Parallel Sublinear Algorithms for Penalized Logistic Regression in Massive Datasets

No Author Given

No Institute Given

Abstract. Penalized logistic regression (PLR) is a widely used supervised learning model. In this paper, we present a parallel implementation of sublinear algorithms for PLR to efficiently improve scalability. We develop concrete parallel algorithms for PLR with ℓ_2-norm and ℓ_1-norm penalties, respectively with MapReduce. Due to the intrinsic random nature of the proposed parallel algorithms, they can provide fault recovery for lengthy distributed computations. They also enjoy the benefits of sublinear dependency on both the volume and dimensionality of training data, and can be widely applied to many large real-world datasets. We have released MapReduce-PSUBPLR to open source at http://code.google.com/p/psubplr.

1 Introduction

The penalized logistic regression (PLR) model [9] plays an important role in machine learning and data mining. The model serves for classification problems, and enjoys a substantial body of supporting theories and algorithms. PLR is competitive with the support vector machines (SVMs) [16], because it has both high accuracy and interpretability (PLR can directly estimate a conditional class probability).

Recently, large-scale applications have emerged from many modern massive datasets. A key characteristic of these applications is that the size of their training data is very large and data dimensionality is very high. For example, in medical diagnostic applications [15], both doctors and patients would like to take the advantage of millions of records over hundreds of attributes. More evidently, search engines on texts or multimedia data must handle data volume in the billion scale and each data instance is characterized by a feature space of thousands of dimensions [8]. Large data volume and high data dimensionality pose computational challenges to machine learning problems.

We tackle these challenges via stochastic approximation approaches. Stochastic approximation methods, such as stochastic gradient descent [19] and stochastic dual averaging [18], obtain optimal generalization guarantees with only a single pass or a small number of passes over the data. Therefore, they can achieve a desired generalization with runtime linear to the dataset size. We further speed up the runtime, and propose sublinear algorithms for PLR via the use of stochastic approximation idea. Our algorithms work at the same level of performance with traditional learning methods for PLR, but require much shorter running time. Our methods access a single feature of training vectors instead of entire training vectors at each iteration. This *sampling* approach brings much improved computational efficiency by eliminating a large number of vector multiplication operations. By devising clever randomized algorithms, we can also enjoy the benefits of taking less number of iterations and hence accessing less number of features. Such reduction in accessing features can substantially reduce running time as pointed out by [11].

Our algorithms can be easily applied to distributed storage systems [12] with parallel updates on all instances. We can achieve good scalability in massive datasets with a MapReduce implementation.

The rest of the paper is organized as follows: Section 2 discusses some related work. In Section 3, we review the penalized logistic regression model along with the sublinear algorithms. In Section 4, we present the parallel framework of our sublinear algorithms for PLR. In Section 5, we depict detailed algorithms and analysis. Section 6 describes the datasets and the baseline of our experiments and presents the experimental results. Finally, we offer our concluding remarks in Section 7.

2 Related Work

There are many existing techniques that address logistic regression with ℓ_1-penalty in the literature.

The *Reduced Memory Multi-pass* (RMMP) algorithm, proposed by Balakrishnan and Madigan [2], is one of the most accurate and fastest convergent algorithms. RMMP trains sparse linear classifiers on high-dimensional datasets in a multi-pass manner. However, this algorithm has computational complexity and memory requirements that make learning on large-scale datasets extremely computational expensive. The central idea of the work is a straightforward quadratic approximation to the likelihood function. When the dimensionality of the data gets large, the cost of many vector-vector multiplication operations increases significantly. Also, the quadratic approximation is added together for all instances in each iteration, and such computation inevitably requires global reduction in a distributed storage system.

The *Hybrid Iterative Shrinkage* (HIS) algorithm, proposed by Shi et al. [14], is also computationally efficient without loss of classification accuracy. This algorithm includes a fixed point continuation phase and an interior point phase. The first phase is based completely on memory efficient operations such as matrix-vector multiplications, while the second phase is based on a truncated Newton's method. Thus, HIS is in the scope and constraints of traditional way of solving the optimization problem. As RMMP has relatively better scalability and performance, we choose to use RMMP instead of HIS as our baseline for the empirical comparison in this paper.

Recently, Clarkson et al. [4] proposed a new method by taking advantage of randomized algorithms. They presented sublinear-time approximation algorithms for optimization problems arising in machine learning, such as linear classifiers and minimum enclosing balls. The algorithm uses a combination of a novel sampling techniques and a new multiplicative update algorithm. They also proved lower bounds which show the running times to be nearly optimal on the unit-cost RAM model.

Hazan et al. [11] exploited sublinear approximation approach to the linear SVM with ℓ_2-penalty, from which we were inspired and borrowed some of the ideas (We generally refer to them as the ETN framework in Section 4). Later on, Cotter et al. [5] extended the work to kernelized SVM cases. In [10], Hazan et al. applied the sublinear approximation approach for solving ridge (ℓ_2-regularized) and lasso (ℓ_1-regularized) linear regression. Garber and Hazan [7] developed the method in semidenfinite programming (SDP).

In this paper, we mainly target the size of the training data at the Gigabytes level. These datasets require us to develop machine learning algorithms towards a more efficient and more parallelized end. In the Hadoop programming environment, researchers have already done much work to cater for the needs of massive datasets.

Early work like PSVM (*Parallel Support Vector Machines*) [3] employed an approximation matrix decomposition method based on row selections to reduce the memory when running the algorithm. It can then increase the number of participated parallel computation nodes to several hundreds. Later, the work of PLDA (*Parallel Latent Dirichlet Allocation*) [17] further improve the computational efficiency through the use of sampling methods. The proposed parallel algorithm in Hadoop is robust, for it has fault tolerance of machines, taking the advantage of Hadoop. Recently, the work by Dean et al. [6] showed that the advantage of parallelization is fully taken in deep learning algorithms. It pushed the limits of parallel computational nodes to a hundred-million level, and in the meantime, achieved a best learning performance ever. Besides Hadoop, there are many other trials on parallelization in machine learning. GraphLab [13], a recent developed tool in CMU (*Carnegie Mellon University*)for large-scale machine learning, tried to tackle with intrinsic problems of Hadoop when applied to machine leaning algorithms. However, the original design purpose is to improve efficiency in a single-machine multi-core situation, which is a drawback in a distributed system.

3 Learning Algorithms for Penalized Logistic Regression Models

Logistic regression is a widely used method for solving classification problems. In this paper, we are mainly concerned with the binary classification problem. Suppose that we are given a set of training data $\mathcal{X} = \{(\mathbf{x}_i, y_i) : i = 1, \ldots, n\}$ where $\mathbf{x}_i \in \mathbb{R}^d$ are input samples and $y_i \in \{-1, 1\}$ are the corresponding labels. For simplicity, we let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^T$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)^T$. In the logistic regression model, the expected value of y_i is given by

$$P(y_i|\mathbf{x}_i) = \frac{1}{1+\exp(-y_i(\mathbf{x}_i^T\mathbf{w}+b))} \triangleq g_i(y_i),$$

where $\mathbf{w} = (w_1, \ldots, w_d)^T \in \mathbb{R}^d$ is a regression vector and $b \in \mathbb{R}$ is an offset term.

3.1 Penalized Logistic Regression Models

We assume that $\mathbf{w}$ follows a Gaussian distribution with mean $\mathbf{0}$ and covariance matrix $\lambda\mathbf{I}_d$ where $\mathbf{I}_d$ is the $d\times d$ identity matrix, i.e. $\mathbf{w} \sim N(\mathbf{0}, \lambda\mathbf{I}_d)$. In this case, we can formulate the optimization problem as

$$\max_{\mathbf{w},b} \left\{F(\mathbf{w}, b|\mathcal{X}) - \frac{\lambda}{2}\|\mathbf{w}\|_2^2\right\}. \tag{1}$$

(1) shows us that the problem reduces to an optimization problem with an ℓ_2-penalty.

In another case, we impose a Laplace prior for $\mathbf{w}$, whose density is given by

$$\log p\,(\mathbf{w}) = d\,\log\frac{\gamma}{2} - \gamma\|\mathbf{w}\|_1.$$

With this prior, we formulate an optimization problem with the ℓ_1-penalty. ℓ_1-penalty logistic regression can serve for both classification and feature selection simultaneously.

$$\max_{\mathbf{w},b} \left\{F(\mathbf{w}, b|\mathcal{X}) - \gamma\|\mathbf{w}\|_1\right\}. \tag{2}$$

3.2 Sublinear Algorithms

The framework of sublinear algorithms is a hybrid method to handle hard margin and soft margin separately and simultaneously. It enjoys the property of fast convergence for both hard margin and soft margin.

Each iteration of the method works in two steps. The first one is the *stochastic primal update*:

(1) An instance $i \in \{1, \ldots, n\}$ is chosen according to a probability vector $\mathbf{p}$;
(2) The primal variable $\mathbf{w}$ is updated according to the derivative of $f_i(\mathbf{w}, b)$ and the soft margin, via an online update with regret.

The second one is the *stochastic dual update*:

(1) A stochastic estimate of $f_i(\mathbf{w}, b)$ plus the soft margin is obtained, which can be computed in $O(1)$ time per term;
(2) The probability vector $\mathbf{p}$ is updated based on the above computed terms by using the *Multiplicative Updates* (MW) framework [1] for online optimization over the simplex.

We use the following notations in our algorithms and analysis.
$clip(\cdot)$ is a projection function defined as follows:

$$clip(a,b) \triangleq \max(\min(a,b), -b) \quad a,b \in \mathbb{R}.$$

$\operatorname{sgn}(\cdot)$ is the sign function; namely,

$$\operatorname{sgn}(x) = \begin{cases} +1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0. \end{cases}$$

$g(\cdot)$ is the logistic function; namely,

$$g(x) = \frac{1}{1+e^{-x}}$$

We let Λ be the $\mathbb{R}_n$ Euclidean space which meets the following conditions:

$$\Lambda = \{\xi \in \mathbb{R}_n \mid \forall i,\ 0 \le \xi_i \le 2,\ \|\xi\|_1 \le \nu n\}.$$

3.3 Sequential Sublinear Algorithm for ℓ_2-Penalty Logistic Regression

Algorithm 1 SLLR-L2

```
1: Input: $\varepsilon > 0, 0 \le \nu \le 1, X \in \mathbb{R}^{n\times d}, Y \in \mathbb{R}^n$
2: Let $T \leftarrow 1000^2\varepsilon^{-2}\log n, \eta \leftarrow \sqrt{\log(n)/T}$
3:     $\mathbf{u}_0 \leftarrow \mathbf{0}_d, \mathbf{w}_1 \leftarrow \mathbf{0}_d, \mathbf{q}_1 \leftarrow \mathbf{1}_n, b_1 \leftarrow 0$
4: for $t = 1$ to $T$ do
5:     $\mathbf{p}_t \leftarrow \mathbf{q}_t/\|\mathbf{q}_t\|_1$
6:     Choose $i_t \leftarrow i$ with probability $\mathbf{p}(i)$
7:     Let $coef = y_{i_t} g\left(-y_{i_t}\left({\mathbf{w}_t}^T \mathbf{x}_{i_t} + b_t\right)\right)$
8:     Let $\mathbf{u}_t \leftarrow \mathbf{u}_{t-1} + \frac{coef}{\sqrt{2T}}\mathbf{x}_{i_t}$
9:         $\xi_t \leftarrow \operatorname{argmax}_{\xi\in\Lambda}\left({\mathbf{p}_t}^T\xi\right)$
10:        $b_t \leftarrow \operatorname{sgn}\left({\mathbf{p}_t}^T\mathbf{y}\right)$
11:      $\mathbf{w}_t \leftarrow \mathbf{u}_t/\max\{1, \|\mathbf{u}_t\|_2\}$
12:      Choose $j_t \leftarrow j$ with probability $\mathbf{w}_t(j)^2/{\|\mathbf{w}_t\|_2}^2$
13:      for $i = 1$ to $n$ do
14:          $\sigma \leftarrow \mathbf{x}_i(j_t)\,{\|\mathbf{w}_t\|_2}^2/\mathbf{w}_t(j_t) + \xi_t(i) + y_i b_t$
15:          $\hat{\sigma} \leftarrow clip(\sigma, 1/\eta)$
16:          $\mathbf{q}_{t+1}(i) \leftarrow \mathbf{q}_t(i)\left(1 - \eta\hat{\sigma} + \eta^2\hat{\sigma}^2\right)$
17:      end for
18: end for
19: Output: $\bar{\mathbf{w}} = \frac{1}{T}\sum_t \mathbf{w}_t, \bar{b} = \frac{1}{T}\sum_t b_t$
```

In Algorithm 1, we give the sublinear approximation procedure for ℓ_2-penalty logistic regression.

3.4 Sequential Sublinear Algorithm for ℓ_1-Penalty Logistic Regression

Algorithm 2 SLLR-L1

```
1: Input: $\varepsilon > 0, \gamma > 0, X \in \mathbb{R}^{n \times d}, Y \in \mathbb{R}^n$
2: Let $T \leftarrow 1000^2 \varepsilon^{-2} \log n, \eta \leftarrow \sqrt{\log(n)/T}$
3:     $\mathbf{u}_0 \leftarrow \mathbf{0}_d, \mathbf{wavg}_0 \leftarrow \mathbf{0}_d, \mathbf{q}_1 \leftarrow \mathbf{1}_n, b_1 \leftarrow 0$
4: for $t = 1$ to $T$ do
5:     $\mathbf{p}_t \leftarrow \mathbf{q}_t / \|\mathbf{q}_t\|_1$
6:     $\mathbf{uprev}_t \leftarrow \mathbf{u}_{t-1}$
7:     Choose $i_t \leftarrow i$ with probability $\mathbf{p}(i)$
8:     Let $coef = y_{i_t} g\left(-y_{i_t}\left(\mathbf{wavg}_{t-1}{}^T \mathbf{x}_{i_t} + b_t\right)\right)$
9:     Let $\mathbf{u}_t \leftarrow \mathbf{u}_{t-1} + \frac{coef}{\sqrt{2T}} \mathbf{x}_{i_t}$
10:        $b_t \leftarrow \mathrm{sgn}\left(\mathbf{p}_t{}^T \mathbf{y}\right)$
11:    for $j = 1$ to $d$ do
12:        if $\mathbf{uprev}_t(j) > 0$ and $\mathbf{u}_t(j) > 0$
13:            $\mathbf{u}_t(j) = \max(\mathbf{u}_t(j) - \gamma, 0)$
14:        if $\mathbf{uprev}_t(j) < 0$ and $\mathbf{u}_t(j) < 0$
15:            $\mathbf{u}_t(j) = \min(\mathbf{u}_t(j) + \gamma, 0)$
16:    end for
17:    $\mathbf{w}_t \leftarrow \mathbf{u}_t / \max\{1, \|\mathbf{u}_t\|_2\}$
18:    $\mathbf{wavg}_t \leftarrow \frac{t-1}{t} \mathbf{wavg}_{t-1} + \frac{1}{t} \mathbf{w}_t$
19:    Choose $j_t \leftarrow j$ with probability $\mathbf{w}_t(j)^2 / \|\mathbf{w}_t\|_2{}^2$
20:    for $i = 1$ to $n$ do
21:        $\sigma \leftarrow \mathbf{x}_i(j_t) \|\mathbf{w}_t\|_2{}^2 / \mathbf{w}_t(j_t) + y_i b_t$
22:        $\hat{\sigma} \leftarrow clip(\sigma, 1/\eta)$
23:        $\mathbf{q}_{t+1}(i) \leftarrow \mathbf{q}_t(i)\left(1 - \eta\hat{\sigma} + \eta^2 \hat{\sigma}^2\right)$
24:    end for
25: end for
26: Output: $\mathbf{wavg}_t, \bar{b} = \frac{1}{T}\sum_t b_t$
```

4 Parallel Framework of PSUBPLR

In this section, we develop an approach to solve sublinear learning for penalized logistic regression using the architecture of MapReduce.

5 Algorithms and Analysis

Our proposed algorithms are fast convergent and intrinsic fault-tolerant.

Procedure PSUBPLR

```
Input parameters: $\varepsilon, X, Y, n, d$
                  $\nu$ // (for $\ell_2$-penalty)
                  $\gamma$ // (for $\ell_1$-penalty)
Let $T \leftarrow \frac{1}{n}1000^2\varepsilon^{-2}\log n, \eta \leftarrow \sqrt{\log(n)/T}$
    $\mathbf{w}_1 \leftarrow \mathbf{0}_d, \mathbf{p}_1 \leftarrow \mathbf{1}_n, b_1 \leftarrow 0$
for $t = 1$ to $T$ do
    HDFS-File("paraw") $\leftarrow$ Write-in-HDFS-File($\mathbf{w}_t$)
    add-to-distributed-cache(HDFS-File("paraw"))
    HDFS-File("parap") $\leftarrow$ Write-in-HDFS-File($\mathbf{p}_t$)
    add-to-distributed-cache(HDFS-File("parap"))
    Primal-MR $\leftarrow$ new MapReduce-Job()
    Primal-MR.configuration.setparameters($T, n, d, b_t$)
    Primal-MR.configuration.setOutputFilePath("sublinear/tmp/primal$t$")
    Primal-MR.start()
    Primal-MR.waitForCompletion()
    $(\mathbf{w}_{t+1}, b_{t+1})$$\leftarrow$PrimalUpdate($\mathbf{w}_t, b_t$)
    soft-thresh-holding($\mathbf{w}_{t+1}$) // for $\ell_1$-penalty only
    $j_t \leftarrow$ samplingFrom($d$) // with probability vector $\mathbf{w}_{t+1}(j)^2/\|\mathbf{w}_{t+1}\|_2{}^2$, for $j \in \{1,\ldots,d\}$
    HDFS-File("paraw") $\leftarrow$ Write-in-HDFS-File($\mathbf{w}_{t+1}$)
    add-to-distributed-cache(HDFS-File("paraw"))
    Dual-MR $\leftarrow$ new MapReduce-Job()
    Dual-MR.configuration.setparameters($d, jt, b_{t+1}, \eta$)
    Dual-MR.configuration.setOutputFilePath("sublinear/tmp/dual$t$")
    Dual-MR.start()
    Dual-MR.waitForCompletion()
    $\mathbf{p}_{t+1}$$\leftarrow$DualUpdate($\mathbf{p}_t$)
end for
Output: $\bar{\mathbf{w}} = \frac{1}{T}\sum_t \mathbf{w}_t, \bar{b} = \frac{1}{T}\sum_t b_t$
```

5.1 Convergence Analysis

We now formally describe the MW algorithm, then state and prove theorems for convergence our algorithms.

Definition 1. *(MW algorithm) [4]. Consider a sequence of vectors $\mathbf{v}_1, ..., \mathbf{v}_T \in \mathbb{R}^d$ and a parameter $\eta > 0$. The Multiplicative Weights (MW) algorithm is defined as follows: let $\mathbf{w}_1 \leftarrow \mathbf{1}_n$, and for $t \geq 1$,*

$$\mathbf{p}_t \leftarrow \mathbf{w}_t/\|\mathbf{w}_t\|_1, \quad \text{and} \quad \mathbf{w}_{t+1}(i) \leftarrow \mathbf{w}_t(i)\left(1 - \eta\mathbf{v}_t(i) + \eta^2\mathbf{v}_t(i)^2\right).$$

The following lemma establishes a regret bound for the MW algorithm.

Lemma 1. *(The Variance MW Lemma) from [4]. The MW algorithm satisfies*

$$\sum_{t=1}^{T} \mathbf{p}_t^T \mathbf{v}_t \leq \min_{i\in\{1,\ldots,n\}} \sum_{t=1}^{T} \max\{\mathbf{v}_t(i), -\frac{1}{\eta}\} + \frac{\log n}{\eta} + \eta \sum_{t=1}^{T} \mathbf{p}_t^T \mathbf{v}_t^2$$

Procedure Primal-Map(key, values)

```
Configuration.$getparameters(T, n, d, b_t)$
$\mathbf{w}_t \leftarrow$ Read-from-HDFS-File(HDFS-File(”paraw”))
$\mathbf{p}_t \leftarrow$ Read-from-HDFS-File(HDFS-File(”parap”))
$i_t, \mathbf{x}_{i_t}, y_{i_t} \leftarrow$ Extract(values)
$r \leftarrow$ random(seed)
if $\mathbf{p}_t(i_t) > \frac{r}{n}$
    $tmp\_coef = \mathbf{p}_t(i_t) y_{i_t} g\left(-y_{i_t}\left({\mathbf{w}_t}^T \mathbf{x}_{i_t} + b_t\right)\right)$
else
    $tmp\_coef = 0$
for $j = 1$ to $d$ do
    $Outkey \leftarrow j$
    $Outvalue \leftarrow \frac{tmp\_coef}{\sqrt{2T}} \mathbf{x}_{i_t}(j)$
    $EmitIntermediate(Outkey, Outvalue.toString())$
end for
```

Procedure Primal-Reduce(key, values)

```
$Outkey \leftarrow key$
$Outvalue \leftarrow 0$
for each v in values
    $Outvalue \leftarrow Outvalue + ParseDouble(v)$
end for
$Emit(Outkey, Outvalue.toString())$
```

Main Theorem needed here to support for the theoretical analysis
It will be like sth below:

Theorem 1. *The proposed parallel algorithm returns an ε-approximate solution to the optimization problem of (**??**) with probability at least $1/?$. It's critical to determine T*

5.2 Fault-Tolerance Analysis

A Theorem needed here to support for the theoretical analysis
It will be like sth below:

Theorem 2. *The proposed parallel algorithm returns an ε-approximate solution to the optimization problem of (**??**) with probability at least $1/?$ when there are a certain percentage of faulty Primal-Maps. T has to be changed here.*

6 Experiments

Current Plan
Datasets and Measurements Description

```
**Procedure** PrimalUpdate($\mathbf{w}_t, b_t$)
$\Delta\mathbf{w}_t \leftarrow$ Read-from-HDFS(HDFS-Path("sublinear/tmp/primalt"))
$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \Delta\mathbf{w}_t$
$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_{t+1} / \max\{1, \|\mathbf{w}_{t+1}\|_2\}$
$b_{t+1} \leftarrow \mathrm{sgn}\left(\mathbf{p}_t{}^T\mathbf{y}\right)$
```

```
**Procedure** Dual-Map(key, values)
Configuration.getparameters($d, j_t, b_{t+1}, \eta$)
$\mathbf{w}_{t+1} \leftarrow$ Read-from-HDFS-File(HDFS-File("paraw"))
$i_t, \mathbf{x}_{i_t}, y_{i_t} \leftarrow$ Extract(values)
$Outkey \leftarrow i_t$
$\sigma \leftarrow \mathbf{x}_{i_t}(j_t)\,\|\mathbf{w}_{t+1}\|_2{}^2/\mathbf{w}_{t+1}(j_t) + y_{i_t} b_{t+1}$
$\hat{\sigma} \leftarrow clip(\sigma, 1/\eta)$
$Outvalue \leftarrow 1 - \eta\hat{\sigma} + \eta^2\hat{\sigma}^2$
$Emit(Outkey, Outvalue.toString())$
```

6.1 Analysis of Performance

on URL Reputation dataset (from UCI) and KDD CUP 2012 dataset

More datasets to be added

Test Error VS Core Number

Test Error VS Iterations

Time Used VS Core Number

6.2 Analysis of Convergence

MAP VS Core Number

MAP VS Iterations

6.3 Analysis of Fault Tolerance

Test Error VS Failed Core Number

Iterations Number VS Failed Core Number

Time Used VS Failed Core Number

Current URL-reputation Dataset Status

7 Conclusion

Also, the multi-class version is ready in matlab codes. It's quite trivial to obtain such a version from previously developed binary classification problem.

Procedure DualUpdate($\mathbf{p}_t$)

```
$\mathbf{var} \leftarrow$ *Read-from-HDFS*(HDFS-Path("sublinear/tmp/dual$t$"))
**for** $j = 1$ to $n$ **do**
    $\mathbf{p}_{t+1}(j) \leftarrow \mathbf{p}_t(j) * \mathbf{var}(j)$
**end for**
```

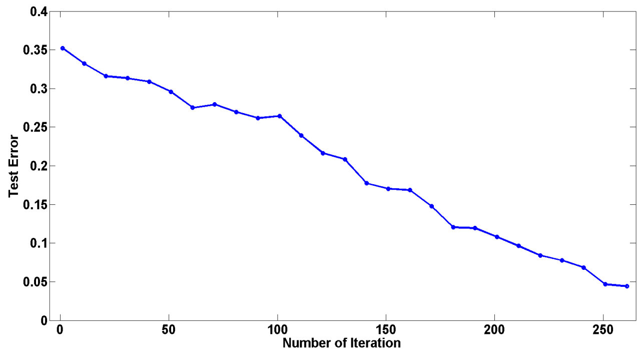

Fig. 1. URL-reputation Dataset, Performance Result

References

1. S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta algorithm and applications. *Manuscript, 2005. Preliminary draft of paper available online at http://www.cs.princeton.edu/ arora/pubs/MWsurvey.pdf*, 2005.
2. S. Balakrishnan and D. Madigan. Algorithms for sparse linear classifiers in the massive data setting. *The Journal of Machine Learning Research*, 9:313–337, 2008.
3. Edward Y Chang, Kaihua Zhu, Hao Wang, Hongjie Bai, Jian Li, Zhihuan Qiu, and Hang Cui. Psvm: Parallelizing support vector machines on distributed computers. *Advances in Neural Information Processing Systems*, 20:16, 2007.
4. K.L. Clarkson, E. Hazan, and D.P. Woodruff. Sublinear optimization for machine learning. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 449–457. IEEE Computer Society, 2010.
5. A. Cotter, S. Shalev-Shwartz, and N. Srebro. The kernelized stochastic batch perceptron. *Arxiv preprint arXiv:1204.0566*, 2012.
6. Jeffrey Dean, Greg S Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V Le, Mark Mao, Marc'Aurelio Ranzato, Andrew W Senior, Paul Tucker, Ke Yang, and Andrew Y Ng. Large scale distributed deep networks. *Advances in Neural Information Processing Systems*, 2012.
7. D. Garber and E. Hazan. Approximating semidefinite programs in sublinear time. In *Advances in Neural Information Processing Systems*, 2011.
8. A. Genkin, D.D. Lewis, and D. Madigan. Large-scale bayesian logistic regression for text categorization. *Technometrics*, 49(3):291–304, 2007.
9. T. Hastie, R. Tishirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer-Verlag, New York, 2001.

10. E. Hazan and T. Koren. Optimal algorithms for ridge and lasso regression with partially observed attributes. *Arxiv preprint arXiv:1108.4559*, 2011.
11. E. Hazan, T. Koren, and N. Srebro. Beating sgd: Learning svms in sublinear time. In *Advances in Neural Information Processing Systems*, 2011.
12. C. Hogan, L. Cassell, J. Foglesong, J. Kordas, M. Nemanic, and G. Richmond. The livermore distributed storage system: Requirements and overview. In *Digest of Papers. Tenth IEEE Symposium on Mass Storage Systems*, pages 6–17. IEEE, 1990.
13. Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. Graphchi: Large-scale graph computation on just a pc. OSDI, 2012.
14. J. Shi, W. Yin, S. Osher, and P. Sajda. A fast hybrid algorithm for large scale l1-regularized logistic regression. *Journal of Machine Learning Research*, 1:8888, 2008.
15. S. Tsumoto. Mining diagnostic rules from clinical databases using rough sets and medical diagnostic model. *Information sciences*, 162(2):65–80, 2004.
16. V. Vapnik. *Statistical Learning Theory.* John Wiley and Sons, New York, 1998.
17. Yi Wang, Hongjie Bai, Matt Stanton, Wen-Yen Chen, and Edward Chang. Plda: Parallel latent dirichlet allocation for large-scale applications. *Algorithmic Aspects in Information and Management*, pages 301–314, 2009.
18. L. Xiao. Dual averaging methods for regularized stochastic learning and online optimization. *The Journal of Machine Learning Research*, 11:2543–2596, 2010.
19. T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116. ACM, 2004.