Distributed Coordinate Descent for L1-regularized Logistic Regression

Ilya Trofimov
Yandex
trofim@yandex-team.ru

Alexander Genkin
AVG Consulting
alexander.genkin@gmail.com

Abstract

Solving logistic regression with L1-regularization in distributed settings is an important problem. This problem arises when training dataset is very large and cannot fit the memory of a single machine. We present d-GLMNET, a new algorithm solving logistic regression with L1-regularization in the distributed settings. We empirically show that it is superior over distributed online learning via truncated gradient.

1 Introduction

Logistic regression with L1-regularization is the method of choice for solving classification and class probability estimation problems in text mining, biometrics and clickstream data analysis. Despite the fact that logistic regression can build only linear separating surfaces, the performance (i.e., testing accuracy) of it, with proper regularization, has shown to be close to that of nonlinear classifiers such as kernel methods. At the same time training and testing of linear classifiers is much faster. It makes the logistic regression a good choice for large-scale problems. A desirable trait of model is sparsity, which is conveniently achieved with L1 or elastic net regularizer.

A broad survey [15] suggests that coordinate descent methods are the best choice for L1-regularized logistic regression on the large scale. Widely used algorithms that fall into this family are: BBR [6], GLMNET [5], newGLMNET [16]. Software implementations of these methods start with loading the full training dataset into RAM.

Completely different approach is online learning [2, 8, 10, 11]. This kind of algorithms do not require to load training dataset into RAM and can access it sequentially (i.e. reading from disk). Balakrishnan and Madigan [2], Langford et al. [8] report that online learning performs well when compared to batch counterparts (BBR and LASSO).

Nowadays we see the growing number of problems where both the number of examples and the number of features are very large. Many problems grow beyond the capabilities of a single computer and need to be handled by distributed systems. Approaches to distributed training of classifiers naturally fall into two groups by the way they split data across computing nodes: by examples [1] or by features [12]. We believe that algorithms that split data by features can achieve better sparsity while retaining similar or better performance and competitive training speed with those that split by examples. Our experiments so far confirm that belief.

Parallel block-coordinate descent is a natural algorithmic framework if we choose to split by features. The challenge here is how to combine steps from coordinate blocks, or computing nodes, and how to organize communication. When features are independent, parallel updates can be combined straightforwardly, otherwise they may come into conflict and not yield enough improvement to objective; this has been clearly illustrated by Bradley et al. [3]. Bradley et al. [3] proposed Shotgun algorithm based on randomized coordinate descent. They studied how many variables can be up-

1

dated in parallel to guarantee convergence. Ho et al. [7] presented distributed implementation of this algorithm compatible with Stale Synchronous Parallel Parameter Server.

Richtárik and Takáč [13] use randomized block-coordinate descent and also exploit partial separability of the objective. The latter relies on sparsity in data, which is indeed characteristic to many large scale problems. They present theoretical estimates of speed-up factor of parallelization. Peng et al. [12] proposed a greedy block-coordinate descent method, which selects the next coordinate to update based on the estimate of the expected improvement in the objective. They found their GRock algorithm to be superior over parallel FISTA and ADMM.

In contrast, our approach is to make parallel steps on all blocks, then use combined update as a direction and perform a line search. We show that sufficient data for line search have the size $O(n + p)$, where n is the number of examples, p is the number of features, so it can be performed on one machine. Consequently, that's the amount of data sufficient for communication between machines. Overall, our algorithm fits into the framework of CGD method proposed by Tseng and Yun [14], which allows us to prove convergence. Block-coordinate descent on a single machine is performed as a step of GLMNET [5].

When splitting data by examples, online learning comes in handy. A classifier is trained in online fashion on each subset, then parameters of classifiers are averaged and used as a warmstart for the next iteration, and so on [1, 17]. We performed an experimental comparison of our algorithm with distributed online learning.

Our main contributions are the following:

- We propose a new parallel coordinate descent algorithm for L1-regularized logistic regression and guarantee its convergence (Section 2)
- We demonstrate how our algorithm can be efficiently implemented on the distributed cluster architecture (Section 3)
- We empirically show effectiveness of our implementation in comparison with distributed online learning via truncated gradient (Section 4)

The C++ implementation of our algorithm, which we call d-GLMNET, is publicly available at *https://github.com/IlyaTrofimov/dlr*.

2 Parallel coordinate descent algorithm

In case of binary classification the logistic regression estimates the class probability given the feature vector $\mathbf{x}$

$$P(y = +1|\mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\beta}^T \mathbf{x})}$$

This statistical model is fitted by maximizing the log-likelihood (or minimizing the negated log-likelihood) at the training set. Some penalty is often added to avoid overfitting and numerical ill-conditioning. In our work we consider L1-regularization penalty, which provides sparsity in the model. Thus fitting the logistic regression with L1-regularization leads to the optimization problem

$$\boldsymbol{\beta}^* = \operatorname*{argmin}_{\boldsymbol{\beta} \in R^n} f(\boldsymbol{\beta}) \tag{1}$$

$$f(\boldsymbol{\beta}) = L(\boldsymbol{\beta}) + \lambda \|\boldsymbol{\beta}\|_1 \tag{2}$$

where $L(\boldsymbol{\beta})$ is the negated log-likelihood

$$L(\boldsymbol{\beta}) = \sum_{i=1}^{n} \log(1 + \exp(-y_i \boldsymbol{\beta}^T \mathbf{x}_i)) \tag{3}$$

$y_i \in \{-1, +1\}$ are labels, $\mathbf{x}_i \in \mathbb{R}^p$ are input features, $\boldsymbol{\beta} \in \mathbb{R}^p$ is the unknown vector of weights for input features. We will denote by nnz the number of non-zero entries in all x_i.

The first part of the objective - $L(\boldsymbol{\beta})$ is convex and smooth. The second part is L1-regularization term - $\lambda \|\boldsymbol{\beta}\|_1$ is convex and separable, but non-smooth. Hence one cannot use directly efficient

optimization techniques like conjugate gradient method or L-BFGS which are often used for logistic regression with L2-regularization.

Our algorithm is based on building local approximations to the objective (2). A smooth part (3) of the objective has quadratic approximation [5]

$$L_q(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}) \stackrel{\text{def}}{=} L(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + \frac{1}{2}\Delta\boldsymbol{\beta}^T \nabla^2 L(\boldsymbol{\beta})\Delta\boldsymbol{\beta}$$

$$= \frac{1}{2}\sum_{i=1}^{N} w_i(z_i - \Delta\boldsymbol{\beta}^T \mathbf{x}_i)^2 + C(\boldsymbol{\beta}) \tag{4}$$

where

$$z_i = \frac{(y_i + 1)/2 - p(\mathbf{x}_i)}{p(\mathbf{x}_i)(1 - p(\mathbf{x}_i))}$$

$$w_i = p(\mathbf{x}_i)(1 - p(\mathbf{x}_i))$$

$$p(\mathbf{x}_i) = \frac{1}{1 + e^{-\boldsymbol{\beta}^T \mathbf{x}_i}}$$

The core idea of GLMNET and newGLMNET is iterative minimization of the penalized quadratic approximation to the objective

$$\operatorname*{argmin}_{\Delta\boldsymbol{\beta}} \left\{ L_q(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}) + \lambda ||\boldsymbol{\beta} + \Delta\boldsymbol{\beta}||_1 \right\} \tag{5}$$

via cyclic coordinate descent. This form (4) of approximation allows to make Newton updates of the vector $\boldsymbol{\beta}$ without storing the Hessian explicitly. Also the approximation (5) has a simple closed-form solution with respect to a single variable $\Delta\beta_j$

$$\Delta\beta_j^* = \frac{T\left(\sum_{i=1}^{n} w_i x_{ij} q_i, \lambda\right)}{\sum_{i=1}^{n} w_i x_{ij}^2} - \beta_j \tag{6}$$

$$T(x, a) = \operatorname{sgn}(x)\max(|x| - a, 0)$$

$$q_i = z_i - \Delta\boldsymbol{\beta}^T \mathbf{x}_i + (\beta_j + \Delta\beta_j)x_{ij}$$

In order to adapt the algorithm to the distributed settings we replace the full Hessian with its block-diagonal approximation $\tilde{H}$. More formally: let us split p input features into M disjoint sets S_k

$$\bigcup_{k=1}^{M} S_k = \{1, ..., p\}$$

$$S_m \cap S_k = \emptyset, k \neq m$$

Denote by $\tilde{H}$ a block-diagonal matrix

$$(\tilde{H})_{jl} = \begin{cases} (\nabla^2 L(\boldsymbol{\beta}))_{jl}, & \text{if } \exists m : j, l \in S_m \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

Let $\Delta\boldsymbol{\beta} = \sum_{m=1}^{M} \Delta\boldsymbol{\beta}^m$, where $\Delta\beta_j^m = 0$ if $j \notin S_m$. Then

$$L_q(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}^m) = L(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta}^m + \frac{1}{2}\Delta(\boldsymbol{\beta}^m)^T \nabla^2 L(\boldsymbol{\beta})\Delta\boldsymbol{\beta}^m$$

$$= L(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta}^m + \frac{1}{2}\sum_{j,k \in S_m} (\nabla^2 L(\boldsymbol{\beta}))_{jk}\Delta\beta_j\Delta\beta_k$$

$$= \frac{1}{2}\sum_{i=1}^{N} w_i(z_i - (\Delta\boldsymbol{\beta}^m)^T \mathbf{x}_i)^2 + C(\boldsymbol{\beta})$$

by summing this equation over m

$$\sum_{m=1}^{M} L_q(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}^m) = \sum_{m=1}^{M} \left(L(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta}^m + \frac{1}{2} \sum_{j,k \in S_m} (\nabla^2 L(\boldsymbol{\beta}))_{jk} \Delta\beta_j \Delta\beta_k \right)$$

$$= ML(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + \frac{1}{2} \Delta\boldsymbol{\beta}^T \tilde{H} \Delta\boldsymbol{\beta} \qquad (8)$$

From the equation (8) and separability of L1 penalty follows that solving the approximation to the objective

$$\operatorname*{argmin}_{\Delta\boldsymbol{\beta}} \left\{ L(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + \frac{1}{2} \Delta\boldsymbol{\beta}^T \tilde{H} \Delta\boldsymbol{\beta} + \lambda ||\boldsymbol{\beta} + \Delta\boldsymbol{\beta}||_1 \right\}$$

is equivalent to solving M independent sub-problems

$$\operatorname*{argmin}_{\Delta\boldsymbol{\beta}^m} \left\{ L_q(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}^m) + \sum_{j \in S_m} |\beta_j + \Delta\beta_j^m| \ \middle| \ \Delta\beta_j^m = 0 \text{ if } j \notin S_m \right\} \qquad (9)$$

and can be done in parallel over M machines. This is the main idea of the proposed algorithm d-GLMNET. We describe a high-level structure of d-GLMNET in the Algorithm 1.

Algorithm 1 *Overall procedure of d-GLMNET*
$\boldsymbol{\beta} \leftarrow 0$
Split {1, … ,p} into M disjoint sets S_1, … , S_M.
Repeat until convergence:

    1.   Do in parallel over M machines
    2.   Minimize $L_q(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}^m) + ||\boldsymbol{\beta} + \Delta\boldsymbol{\beta}^m||_1$ with respect to $\Delta\boldsymbol{\beta}^m$
    3. $\Delta\boldsymbol{\beta} \leftarrow \sum_{m=1}^{M} \Delta\boldsymbol{\beta}^m$
    4.   Find $\alpha \in (0, 1]$ by the line search procedure (Algorithm 3)
    5. $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \alpha\Delta\boldsymbol{\beta}$

return $\boldsymbol{\beta}$

The downside of using line search is that it can hurt sparsity. We compute the regularization path (Section 4.2) by running Algorithm 1 with decreasing L1 penalty, and the algorithm starts with $\boldsymbol{\beta} = 0$, so absolute values of $\boldsymbol{\beta}$ tend to increase. However there may be cases when $\Delta\beta_j = -\beta_j$ for some j on step 2 of Algorithm 1, so β_j can to go back to 0. In that case, if line search on step 3 selects $\alpha < 1$, then the opportunity for sparsity is lost.

To retain the sparsity our algorithm takes two precautions. First, line search is prevented if $\alpha = 1$ guarantees sufficient decrease in the objective value (step 1 of Algorithm 3). Second, there is a complication in the convergence criterion. It starts by checking if relative decrease in the objective is sufficiently small or maximum number of iteration has been reached. If that turns out true, the algorithm checks if setting α back to 1 would not be too much of an increase in the objective. If that is also true, the algorithms updates $\boldsymbol{\beta}$ with $\alpha = 1$ and then stops.

Algoritm 2 presents our approach for solving sub-problem (9). d-GLMNET makes one cycle of coordinate descent over input features for approximate solving (9). Despite the fact that GLMNET and newGLMNET use multiple passes we found that our approach works well in practice. We also use $\tilde{H} + \nu I$ with small $\nu = 10^{-6}$ instead of $\tilde{H}$ in (8). The fact that matrix $\tilde{H} + \nu I$ is positive definite is essential for the proof of convergence (see Section 2.1).

Like in other Newton-like algorithms a line search should be done to guarantee convergence. The Algorithm 3 describes our line search procedure. We found that selecting α_{init} by minimizing the objective (2) (step 2, Algorithm 3) speeds up the convergence of the Algorithm 1. We used $b = 0.5, \sigma = 0.01, \gamma = 0$ for numerical experiments.

2.1 Convergence

Algorithm d-GLMNET falls into the general framework of block-coordinate gradient descent (CGD) proposed by Tseng and Yun [14]. CGD is about minimization of a sum of a smooth function and

Algorithm 2 *Solving quadratic sub-problem at machine m*
$\Delta\boldsymbol{\beta}^m \leftarrow 0$
Cycle over j in S_m:

      1.   Minimize $L_q(\boldsymbol{\beta}, \Delta\boldsymbol{\beta}^m) + ||\boldsymbol{\beta} + \Delta\boldsymbol{\beta}^m||_1$ with respect to $\Delta\beta_j^m$ using (6)

return $\Delta\boldsymbol{\beta}^m$

Algorithm 3 *Line search procedure*

    1. If $\alpha = 1$ yields sufficient relative decrease in the objective, return $\alpha = 1$.

    2. Find $\alpha_{init} = \operatorname{argmin}_{\delta < \alpha \leq 1} f(\boldsymbol{\beta} + \alpha\Delta\boldsymbol{\beta})$, $\delta > 0$.

    3. Armijo rule: let α be the largest element of the sequence $\{\alpha_{init}b^j\}_{j=0,1,...}$ satisfying

$$f(\boldsymbol{\beta} + \alpha\Delta\boldsymbol{\beta}) \leq f(\boldsymbol{\beta}) + \alpha\sigma D$$

    where $0 < b < 1, 0 < \sigma < 1, 0 \leq \gamma < 1$, and

$$D = \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + \gamma\Delta\boldsymbol{\beta}^T \tilde{H} \Delta\boldsymbol{\beta} + \lambda \left(||\boldsymbol{\beta} + \Delta\boldsymbol{\beta}||_1 - ||\boldsymbol{\beta}||_1 \right)$$

    return α

separable convex function: in our case, negated log-likelihood and L1 penalty. At each iteration CGD solves penalized quadratic approximation problem

$$\underset{\Delta\boldsymbol{\beta}}{\operatorname{argmin}} \left\{ L(\boldsymbol{\beta}) + \nabla L(\boldsymbol{\beta})^T \Delta\boldsymbol{\beta} + \frac{1}{2}\Delta\boldsymbol{\beta}^T H \Delta\boldsymbol{\beta} + \lambda||\boldsymbol{\beta} + \Delta\boldsymbol{\beta}||_1 \right\} \tag{10}$$

where H is positive definite, iteration specific. For convergence it also requires that for some $\lambda_{max}, \lambda_{min} > 0$ for all iterations

$$\lambda_{min}I \preceq H \preceq \lambda_{max}I \tag{11}$$

At each iteration updates are done over some subset of features. (That would always be all features in our case, so the rules of subset selection are irrelevant). After that a line search by the Armijo rule should be conducted. Then Tseng and Yun [14] prove that $f(\boldsymbol{\beta})$ converges as least Q-linearly and $\boldsymbol{\beta}$ converges at least R-linearly.

d-GLMNET inherits the properties of newGLMNET, for which Yuan et al. [16] already proved that it belongs to the CGD framework and inferred the convergence results. That's why we only give the sketch of the proof, outlining the difference. newGLMNET algorithm in (10) for H uses full Hessian $H = \nabla^2 L(\boldsymbol{\beta}) + \nu I$, and Yuan et al. [16] proves (11) for that. Instead, d-GLMNET uses block-diagonal approximation $H = \tilde{H} + \nu I$, where $\tilde{H}$ is defined in (7). That's why CGD iteration (10) for the full set of features is block separable and can be parallelized. To prove (11) for block-diagonal H denote its diagonal blocks by $H^1, ..., H^M$ and represent an arbitrary vector $\mathbf{x}$ as a concatenation of subvectors of corresponding size: $\mathbf{x}^T = (\mathbf{x}_1^T, ..., \mathbf{x}_M^T)$. Then we have

$$\mathbf{x}^T H \mathbf{x} = \sum_{m=1}^{M} \mathbf{x}_m^T H^m \mathbf{x}_m$$

Notice that $H^m = \nabla^2 L(\boldsymbol{\beta}^m) + \nu I$, where $\nabla^2 L(\boldsymbol{\beta}^m)$ is a Hessian over the subset of features S_m. So for each H^m property (11) is already proved in [16]. That means $\lambda_{min}||\mathbf{x}_m||^2 \leq \mathbf{x}_m^T H^m \mathbf{x}_m \leq \lambda_{max}||\mathbf{x}_m||^2$ for $m = 1, ..., M$, and we obtain the required

$$\lambda_{min}||\mathbf{x}||^2 \leq \mathbf{x}^T H \mathbf{x} \leq \lambda_{max}||\mathbf{x}||^2$$

3   Scalable software implementation

Typically most of datasets are stored in "by example" form, so a transformation to "by feature" form is required for d-GLMNET. For large datasets this operation is hard to do on a single machine. We use a Map/Reduce cluster [4] for this purpose. This transformation typically takes 1-5% of time relative to the regularization path calculating (Section 4.2). Training dataset partitioning over

machines is done by means of a Reduce operation. We did not implemented d-GLMNET completely in the Map/Reduce programming model since it is ill-suited for iterative machine learning algorithms [9, 1].

In d-GLMNET machine m solves at each iteration the sub-problem (9). The machine m stores the part X_m of training dataset corresponding to a subset S_m of input features. $X_m = \{L_j | j \in S_m\}$ where $L_j = \{(i, x_{ij}) | x_{ij} \neq 0\}$. Our program expects that input file is already in "by feature" representation, see Table 1. This format of input file allows to read training dataset sequentially

Table 1: Input file format

feature_id	(example_id, value)	(example_id, value)	...	feature_id	(example_id, value)	...

from the disk and make coordinate updates (6) while solving sub-problem (9). Our program stores into the RAM only vectors: $\mathbf{y}$, $(\exp(\boldsymbol{\beta}^T x_i))$, $(\Delta\boldsymbol{\beta}^T x_i)$, $\boldsymbol{\beta}$, $\Delta\boldsymbol{\beta}$. Thus the total memory footprint of our implementation is $O(n + p)$.

Algorithm 4 presents a high-level structure of our software implementation. We consider this as a general framework for distributed block-coordinate descent, which can be used with various types of updates during step 2.

Algorithm 4 *Distributed coordinate descent*
Repeat until convergence:

1. *Do in parallel over M machines*

2. *Read part of training dataset X_m sequentially; make updates of $\Delta\boldsymbol{\beta}^m$, $(\Delta(\boldsymbol{\beta}^m)^T x_i))$*

3. *Sum up vectors $\Delta\boldsymbol{\beta}^m$, $(\Delta(\boldsymbol{\beta}^m)^T x_i)$ using MPI_AllReduce:* [1]

4. $\Delta\boldsymbol{\beta} \leftarrow \sum_{m=1}^{M} \Delta\boldsymbol{\beta}^m$

5. $(\Delta\boldsymbol{\beta}^T \mathbf{x}_i) \leftarrow \sum_{m=1}^{M} (\Delta(\boldsymbol{\beta}^m)^T \mathbf{x}_i)$

6. *Find step size α using line search (Algorithm 3)*

7. $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \alpha\Delta\boldsymbol{\beta}$,

8. $(\exp(\boldsymbol{\beta}^T x_i)) \leftarrow (\exp(\boldsymbol{\beta}^T x_i + \alpha\Delta\boldsymbol{\beta}^T x_i))$

Sequential data reading from disk instead of RAM may slow down the program in case of smaller datasets, but it makes the program more scalable. Also it conforms to the typical pattern of a multi-user cluster system: large disks, many jobs started by different users are running simultaneously. Each job might process large data but it is allowed to use only a small part of RAM at each machine.

Solving sub-problem (9) during step 2 in Algorithm 4 requires $O(nnz)$ operations and it is well suited for large and sparse datasets. The communication cost during step 3 in Algorithm 4 is $O((n + p) \ln M)$. A logarithmic term arises because machines communicate via a tree structure during MPI_AllReduce.

4 Numerical experiments

4.1 Datasets and experimental settings

We used three datasets for numerical experiments. These datasets are from the Pascal Large Scale Learning Challenge 2008 [2]

---

[1]We used an implementation from the Vowpal Wabbit project
$https : //github.com/JohnLangford/vowpal\_wabbit$

[2]http://largescale.ml.tu-berlin.de/

Table 2: Datasets summary

dataset	size	#examples (train/test)	#features	nnz	avg nonzeros
epsilon	12 Gb	0.4×10^6 / 0.1×10^6	2000	8.0×10^8	2000
webspam	21 Gb	0.315×10^6 / 0.035×10^6	16.6×10^6	1.2×10^9	3727
dna	71 Gb	45×10^6 / 5×10^6	800	9.0×10^9	200

- **epsilon** - A synthetic dataset, we used preprocessing and train/test splitting from http://www.csie.ntu.edu.tw/ ~cjlin/libsvmtools/datasets/binary.html
- **webspam** - Webspam classification problem, we used preprocessing and train/test splitting from http://www.csie.ntu.edu.tw/ ~cjlin/libsvmtools/datasets/binary.html
- **dna** - Splice cite recognition problem. We did the same preprocessing as in challenge (see ftp://largescale.ml.tu-berlin.de/largescale/dna/) and did train/test splitting

The datasets are summarized in Table 2. Numerical experiments were carried out at 16 multicore blade servers having Intel(R) Xeon(R) CPU E5-2660 2.20GHz, 32 GB RAM, connected by Gigabit Ethernet. Each server ran one instance of d-GLMNET or Vowpal Wabbit at once.

4.2 Experimental protocol for d-GLMNET

We tested d-GLMNET by solving the problem (1) for a set of regularization parameters, see Algorithm 5.

Algorithm 5 *Computing the regularization path*

Find λ_{max} for which entire vector $\boldsymbol{\beta} = 0$.

For $i = 1$ to 20

    Solve (1) with $\lambda = \lambda_{max} * 2^{-i}$ using previous $\boldsymbol{\beta}$ as a warmstart

For each λ we calculated for a corresponding final $\boldsymbol{\beta}$ the testing quality and the number of non-zero entries. For the "dna" dataset we tested 4 additional regularization parameters $\lambda \in [2730.7, 5461.3]$ because of low density of points in the region with $100 - 300$ non-zero features (Figure 1c).

4.3 Experimental protocol for distributed online learning via truncated gradient

We compared d-GLMNET with the distributed variant of online learning via truncated gradient. The online learning via truncated gradient was presented in [8]. An idea for adapting it to the distributed settings was presented in [1]. We used the first part of [1, Algorithm 2] which proposes to compute a weighted average of classifiers trained at M machines independently. The second part of this algorithm takes the result of the first part as a warmstart for L-BFGS. As we pointed out earlier L-BFGS it not applicable for solving logistic regression with L1-regularization. This algorithm requires training dataset partitioning by examples over M machines.

The Algorithm 2 from [1] is implemented in the Vowpal Wabbit project [3]. We tested the same set of regularization parameters as for d-GLMNET, i.e $\lambda \in \{\lambda_{max}2^{-1}, \lambda_{max}2^{-2}, ..., \lambda_{max}2^{-20}\}$ [4]. Since online learning has many free parameters we made a full search for "epsilon" and "webspam" datasets. We tested jointly learning rates (raging from 0.1 to 0.5), decays of the learning rate (raging from 0.5 to 0.9) for each λ and allowed Vowpal Wabbit to make 50 passes of online learning. After each pass we saved a vector $\boldsymbol{\beta}$. After training we evaluated a quality of all classifiers at the test set and counted the number of non-zero entries in $\boldsymbol{\beta}$.

For the biggest dataset "dna" we did 25 passes and used default learning rate (0.1) and decay (0.5). We also tested additional range of regularization parameter $\lambda \in \{10.7, 10.7 \times 2^{-1}, ..., 10.7 \times 2^{-9}\}$ since Vowpal Wabbit produced only very sparse classifiers with low testing quality.

---

[3] $https://github.com/JohnLangford/vowpal\_wabbit$, we used version 7.5

[4] The parameter λ in (2) is related to the option - *-l1 arg* in Vowpal Wabbit by equation $arg = \lambda/n$ where n is the number of training examples
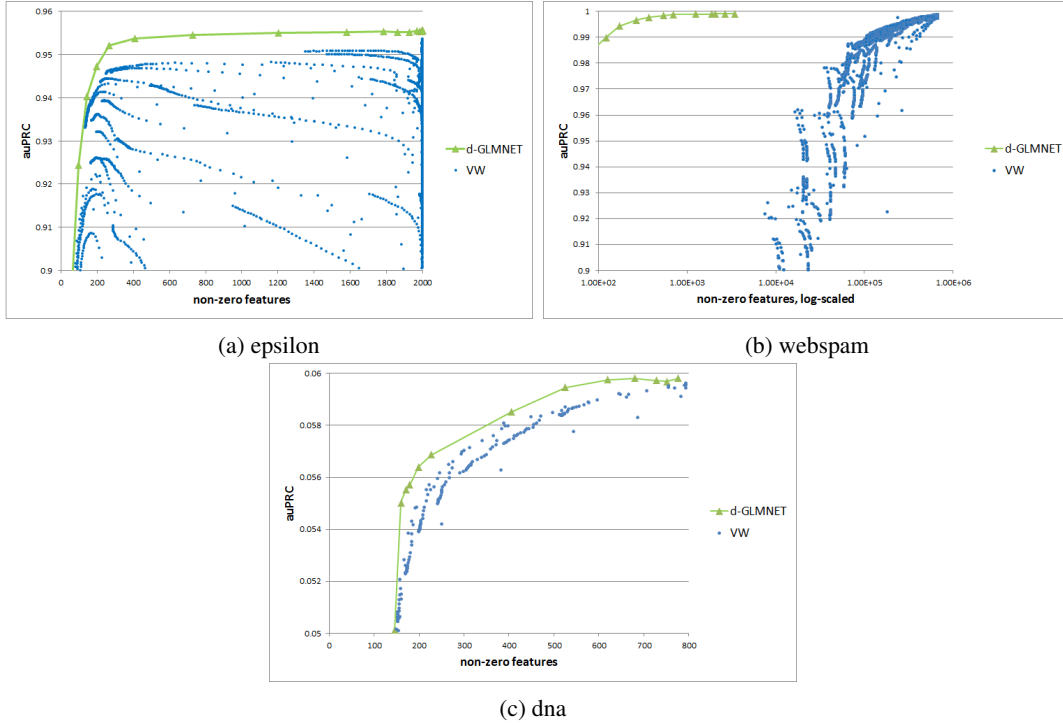
(a) epsilon

(b) webspam

(c) dna

Figure 1: Testing quality (area under Precision-Recall curve) versus non-zero entries count in $\boldsymbol{\beta}$

Table 3: Execution times

dataset	#iter	time, sec	linear search	avg time per iter, sec	avg time per iter, sec
			d-GLMNET		Vowpal Wabbit
epsilon	182	1667	5%	9	30
webspam	269	6318	6%	23	50
dna	123	17626	25%	143	59

4.4 Results

Figure 1 demonstrates results of the experiments: area under Precision-Recall curve on the test set against the number of non-zero components in the $\boldsymbol{\beta}$. We compare results for the whole regularization path of d-GLMNET and each parameter combination and pass number for Vowpal Wabbit. The d-GLMNET algorithm is a clear winner: for each data set, each degree of sparsity, it yields the same or better testing quality. We notice that for online learning different combinations of parameters yield very different results. Online learning is often advertised as a very fast method, but the need to perform a search of good parameters lessens this advantage. At the same time the d-GLMNET algorithm has no free parameters except a regularization coefficient.

Table 3 presents execution times for the whole regularization pass for each dataset, total number of iterations, and average time per iteration. We found that linear search does not hurt much the performance - it takes 5-25% time at different datasets. There is no direct time comparison between d-GLMNET and Vowpal Wabbit because of the parameter search for the latter. The last column in the table gives average time per iteration for Vowpal Wabbit: this can be compared to the same number for d-GLMNET, because one iteration for both algorithms corresponds to one full pass over the training data set, and has the same computational complexity $O(nnz)$.

Acknowledgments

References

[1] Agarwal, A., Chapelle, O., Dudík, M., and Langford, J. (2011). A reliable effective terascale linear learning system. Technical report. http://arxiv.org/abs/1110.4198.

[2] Balakrishnan, S. and Madigan, D. (2007). Algorithms for Sparse Linear Classifiers in the Massive Data Setting. *Journal of Machine Learning Research*, 1:1–26.

[3] Bradley, J. K., Kyrola, A., Bickson, D., and Guestrin, C. (2011). Parallel Coordinate Descent for L1-Regularized Loss Minimization. In *ICML' 11*, Bellevue, WA, USA.

[4] Dean, J. and Ghemawat, S. (2004). MapReduce : Simplified Data Processing on Large Clusters. In *OSDI' 04*, San Francisco.

[5] Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. *Journal of Statistical Software*, 33(1).

[6] Genkin, A., Lewis, D. D., and Madigan, D. (2007). Large-Scale Bayesian Logistic Regression for Text Categorization. *Technometrics*, 49(3):291–304.

[7] Ho, Q., Cipar, J., Cui, H., Kim, J. K., Lee, S., Gibbons, P. B., Gibson, G. A., Ganger, G. R., and Xing, E. P. (2013). More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server. In *NIPS' 13*.

[8] Langford, J., Li, L., and Zhang, T. (2009). Sparse Online Learning via Truncated Gradient. *Journal of Machine Learning Research*, 10:777–801.

[9] Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J. M. (2010). Graphlab: A new framework for parallel machine learning. In *UAI' 10*, Cataline Island, California.

[10] McMahan, H. B. (2011). Follow-the-Regularized-Leader and Mirror Descent : Equivalence Theorems and L1 Regularization. In *AISTATS' 11*.

[11] McMahan, H. B., Holt, G., Sculley, D., Young, M., Ebner, D., Grady, J., Nie, L., Phillips, T., Davydov, E., Golovin, D., Chikkerur, S., Liu, D., Wattenberg, M., Hrafnkelsson, A. M., Boulos, T., and Kubica, J. (2013). Ad Click Prediction: a View from the Trenches. In *KDD' 13*, Chicago, Illinois, USA.

[12] Peng, Z., Yan, M., and Yin, W. (2013). Parallel and Distributed Sparse Optimization. In *STATOS' 13*.

[13] Richtárik, P. and Takáč, M. (2012). Parallel Coordinate Descent Methods for Big Data Optimization. Technical report. http://arxiv.org/abs/1212.0873.

[14] Tseng, P. and Yun, S. (2009). A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117(1-2):387–423.

[15] 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. *Journal of Machine Learning Research*, 11:3183–3234.

[16] Yuan, G.-X., Ho, C.-H., Hsieh, C.-J., and Lin, C.-J. (2012). An Improved GLMNET for L1-regularized Logistic Regression. *Journal of Machine Learning Research*, 13:1999–2030.

[17] Zinkevich, M., Weimer, M., Smola, A., and Li, L. (2010). Parallelized Stochastic Gradient Descent. In *NIPS' 10*.