Learning Large Scale Ordinal Ranking Model via Divide-and-Conquer Technique

Lu Tang
University of Michigan
Ann Arbor, Michigan
lutang@umich.edu

Sougata Chaudhuri
A9.com, Inc.
Palo Alto, California
sougata@a9.com

Abraham Bagherjeiran
A9.com, Inc.
Palo Alto, California
abagher@a9.com

Ling Zhou
University of Michigan
Ann Arbor, Michigan
zholing@umich.edu

ABSTRACT

Structured prediction, where outcomes have a precedence order, lies at the heart of machine learning for information retrieval, movie recommendation, product review prediction, and digital advertising. Ordinal ranking, in particular, assumes that the structured response has a linear ranked order. Due to the extensive applicability of these models, substantial research has been devoted to understanding them, as well as developing efficient training techniques. One popular and widely cited technique of training ordinal ranking models is to exploit the linear precedence order and systematically reduce it to a binary classification problem. This facilitates the usage of readily available, powerful binary classifiers, but *necessitates* an expansion of the original training data, where the training data increases by $K - 1$ times of its original size, with K being the number of ordinal classes. Due to prevalent nature of problems with large number of ordered classes, the reduction leads to datasets which are too large to train on single machines. While approximation methods like stochastic gradient descent are typically applied here, we investigate exact optimization solutions that can scale. In this paper, we present a divide-and-conquer (DC) algorithm, which divides large scale binary classification data into a cluster of machines and trains logistic models in parallel, and combines them at the end of the training phase to create a single binary classifier, which can then be used as an ordinal ranker. It requires no synchronization between the parallel learning algorithms during the training period, which makes training on large datasets feasible and efficient. We prove consistency and asymptotic normality property of the learned models using our proposed algorithm. We provide empirical evidence, on various ordinal datasets, of improved estimation and prediction performance of the model learnt using our algorithm, over several standard divide-and-conquer algorithms.

KEYWORDS

Binary Classification, Ordinal Ranking, Big Data

ACM Reference Format:
Lu Tang, Sougata Chaudhuri, Abraham Bagherjeiran, and Ling Zhou. 2018. Learning Large Scale Ordinal Ranking Model via Divide-and-Conquer Technique. In *WWW '18 Companion: The 2018 Web Conference Companion, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3184558.3191658

1 INTRODUCTION

Structured prediction involves predicting structured objects, rather than scalar discrete or real values. One of the earliest structured prediction problem consists of ordinal labeled data, that is, scalar outcomes which have a ranked order. Learning to rank ordinal outcomes has proven effective in search-based advertising, web search, ratings, and other applications where outcomes are both categorical and ordered in nature [1, 3, 4, 8, 13, 14, 17, 22].

Recognizing that nearly all ordinal regression algorithms are built upon binary classification algorithm, we seek an efficient ordinal approach to use as a black box for our proposed binary classification algorithm [2, 6, 7, 12, 19, 23, 26]. We use the technique discussed in [15] in which the authors reduce ordinal regression to multiple binary classification problems and then solve them jointly as a single binary classifier. A simple step is then used to convert the binary outputs to an ordinal rank, which also leads to an immediate generalization analysis. The main advantages of the proposed scheme is that reduction technique is efficient and only one binary classifier needs to be trained on the expanded classification data; see for example, applications in online advertising [1, 3].

This well-known reduction technique introduces a scalability challenge. Reducing ordinal regression to binary classification expands the instance space by $K - 1$ times of its original size, where K is the number of ordinal outcomes. In the modern era of extreme class problems, where large number of ordered classes is prevalent, the reduction leads to datasets which are too large to train on single machines. Applying this reduction to large-scale modeling problems such as search-based advertising or web search, makes training binary classification models harder. This motivates us to investigate large scale binary classification training algorithms.

The problems outlined above for ordinal regression falls under the “Big Data" regime. Big Data has led to approximation algorithms to cope with data so “big" in dimensions or, as in our case, training instances, that it cannot fit into main memory on a single machine.

Algorithms handle big data by relying on distributed hardware architectures and approximate optimization methods [9, 21, 24, 25, 30]. By distributing computation across a large cluster of machines with increasingly cheap SSD storage and mini-batch computation frameworks such as GPUs, they are able to run multiple passes over the data to eventually arrive at an appoximately close-enough solution. Since these are approximations, they are not guaranteed to solve for the ordinal regression problem. We observe anecdotally that down-sampling, applying the reduction in [15], and then training with a good in-memory batch method such as [11] leads to better and more consistent performance despite the fact that there may be information loss and weak theoretical results.

We seek a large scale exact solver for the convex binary classification optimization problem to use in ordinal regression. We propose a divide-and-conquer strategy called the robust inverse variance weighted average (**RIVWA**) that is ideally suited to ordinal regression and, we believe other structured prediction problems. It provides an exact solution to the binary classification problem, scales to large problems, and involves only a single pass over the data. Solving the problem exactly means that we can obtain ordinal regression models using the reduction described in [15].

We list our contributions below:

- We first outline the reduction step to convert ordinal data to binary data, and the conversion of the trained binary classifier into an ordinal ranker. We then propose a method to divide large binary classification dataset into chunks, train individual regularized logistic classifiers on the blocks of data, and combine the classifiers in an efficient way to get single binary classifier. Our method requires no synchronization between learning algorithms on the data blocks and thus, the learning algorithms can run in parallel, on distributed data frameworks like Hadoop. Moreover, our method is a single pass method, that is, the division step and combinations step are executed only once, and do not need to be repeated. We note that once the data is divided, the actual training of classifiers on individual blocks can mimic any feature of training on the full dataset (assuming feasibility). Thus, the blocks can be loaded from fast SSDs, trained on CPU or fast GPU and use standard optimization algorithms, neural networks etc.
- We prove that the model coefficients from our method is consistent with the true underlying parameter and asymptotically normally distributed.
- We compare our model on multiple datasets (public and proprietary), and show improvement in estimation and prediction performance, as well as reduction in training time in some cases, over several competing divide and conquer methods.

2 RELATED WORK

Here, we give an overview of related work in divide-and-conquer paradigm for fitting logistic classification models. DC algorithms for logistic classification mainly refer to methods of partitioning the full dataset into M separate parts (*splitting*), obtaining coefficient estimates from each part, and combining the M sets of estimates to get the final result (*merging*). A DC algorithm is more efficient if no synchronization is required between individual learning algorithms, leading to true parallel learning. Although there are many relevant work that studies DC algorithms from various perspectives, we specifically emphasize the estimation consistency and asymptotic normality properties of a DC estimator . Thus, we only include variance dependent methods that we are aware of in the literature.

The most straightforward DC method is to take the simple average (**SA**) of the estimates as:

$$\text{SA:}\quad \hat{\theta} = \tfrac{1}{M}\textstyle\sum_{m=1}^{M}\hat{\theta}_m$$

where $\hat{\theta}_m$ is the logistic classifier coefficient estimate from the m-th data partition. This method has been studied by others and shown to produce high variance in the combined estimator [18, 30].

Another popular DC method that has been studied extensively under the topic of meta-analysis [27] is to take the inverse variance weighted average (**IVWA**) of the separate estimates as

$$\text{IVWA:}\quad \hat{\theta} = (\textstyle\sum_{m=1}^{M}\hat{\Sigma}_m^{-1})^{-1}\sum_{m=1}^{M}\hat{\Sigma}_m^{-1}\hat{\theta}_m$$

where $\hat{\Sigma}_m$ is the estimated variance-covariance matrix of $\hat{\theta}_m$. For logistic regression, $\hat{\Sigma}_m = \{X_m^\top V_m(\hat{\theta}_m)X_m\}^{-1}$, where X_m is the m-th block feature matrix, and $V_m(\theta)$ is a diagonal matrix with diagonal element $v_m(\theta)_{i,i} = \sigma((X_m^\top\theta)_i)\{1 - \sigma((X_m^\top\theta)_i)\}$, where $\sigma(x) = 1/(1+e^{-x})$ is the sigmoid function. This estimator guarantees theoretical efficiency in the sense that the DC estimator can achieve the smallest variance possible, which is the variance achieved by the benchmark of directly training on the full data. However, because of overfitting issue, due to lack of regularization, the empirical results usually show larger variance (and hence worse performance) than the benchmark. Although for ordinal regression there are many established methods (for example [16]) that can be used as baseline, we specifically choose to compare with the full data maximum likelihood estimator since we are interested in not only the prediction, but also the efficiency for inference.

To enforce sparsity, a majority voting (**MV**) method was proposed to select the most frequently identified features from lasso regressions across all data divisions [5]. This method returns nonzero weights only for features that are identified by lasso regression across a majority of data parts, and lets the rest be zero. The combination step takes the form

$$\text{MV:}\quad \hat{\theta} = A(\textstyle\sum_{m=1}^{M}A^\top\hat{\Sigma}_m^{-1}A)^{-1}\sum_{m=1}^{M}A^\top\hat{\Sigma}_m^{-1}AA^\top\hat{\theta}_m$$

where A is a column-wise slicing of an identify matrix I_D, selecting columns $\{j : \sum_{m=1}^{M} 1[\hat{\theta}_{m,j} \neq 0] > v\}$ for some voting threshold v (D is the feature dimension). Here, $\hat{\theta}_m$ are lasso regularized estimates obtained from each data block m, unlike in **SA** and **IVWA** methods, where the estimates are not regularized. Due to the sparseness of $\hat{\theta}_m$, this method is numerically robust. However, it requires tuning of two parameters, the lasso regularization parameter and the voting threshold v. Additionally, the combined estimator $\hat{\theta}$ is biased due to the biasedness of $\hat{\theta}_m$, $m = 1, \ldots, M$.

As aforementioned, a major challenge for regularized estimators is that they are biased. Thus the combined estimate often lack theoretical guarantees in terms of consistency. Previous work has been done using the bootstrap subsampling idea [10] to estimate and correct the bias of DC estimators [30]. However, in this paper, we derive the closed-form expression of the bias of ℓ_1 regularized logistic regression, and directly correct the bias within each data part before combining the results. The importance of correcting for bias will be highlighted in the section discussing the theoretical merits of our method.

3 ORDINAL TO BINARY CLASSIFICATION REDUCTION

We outline the critical points of the method of reduction from ordinal ranking to binary classification given by a previous work [15]. Ordinal outcomes naturally inspire a binary classification approach for training models. As an example, consider the satisfaction level of a user for a product, with five possible levels. By asking the question "is the satisfaction level for the user greater than level k", one can get a binary classification problem for a fixed k, since the answer would be *yes* or *no* (1 or 0). By varying $k = 1, 2, 3, 4$, for each user, one can get 4 different binary classification problems. The approach then reduces to a question of how the classification models be trained and combined to obtain an ordinal ranking model. The main advantage of reducing ordinal ranking problem into binary classification problem is that it facilitates the usage of well-tuned binary classifiers available with standard libraries. We focus on logistic loss as the classification objective function since our theoretical results are derived for the same.

For a binary logistic regression, with instance $x \in \mathbb{R}^D$ and label $y \in \{0, 1\}$, the binary classifier $f(x)$ to be learnt is parameterized by $\beta \in \mathbb{R}^D$, i.e., $f(x) = x^\top \beta$. The loss (or the negative log likelihood) function of a training dataset is

$$\textstyle\sum_{i=1}^{N} \left\{\log\left(1 + e^{f(x_i)}\right) - y_i f(x_i)\right\} \tag{1}$$

where N is the training sample size, and the estimated coefficient vector $\hat{\beta}$ is the minimizer to (1).

A K class ordinal ranking problem is defined by an instance $x \in \mathcal{X} \subseteq \mathbb{R}^D$ and label $y \in \mathcal{Y} = \{1, 2, \ldots, K\}$, where $1 \leq 2 \leq \ldots \leq K$. The objective is to learn a ranking rule $r : \mathcal{X} \mapsto \mathcal{Y}$, which will minimize a weighted point-wise loss function with weights defined by some cost $C_{y,r(x)}$. Each instance and label pair (x_i, y_i) is reduced to a binary classification pair (along with introduction of a weight) by the following technique:

$$\begin{aligned} x_i^k &= (x_i^\top, e_k^\top)^\top \in \mathbb{R}^{D+K-1}, \\ y_i^k &= 1[k < y], \\ w_i^k &= |C_{y_i,k} - C_{y_i,k+1}|, \end{aligned} \tag{2}$$

for $k = 1, \ldots, K-1$, where $C_{y,k}$ is the cost for assigning an outcome of k when the actually value is y, and e_k is the standard basis vector in dimension $K - 1$. *As a result, the original sample size expands from N to* $(K-1)N$. Then, a logistic classifier $f(\cdot)$ can be trained on the expanded training set by minimizing the new loss function

$$\textstyle\sum_{i=1}^{N} \sum_{k=1}^{K-1} w_i^k \left\{\log\left(1 + e^{f(x_i^k)}\right) - y_i^k f(x_i^k)\right\}. \tag{3}$$

This can be viewed as the loss (negative log likelihood) of a training data with sample size $\tilde{N} = (K-1)N$, feature dimension $\tilde{D} = D+K-1$, and sample weights specified by w_i^k. The solution to (3) would lead to a classifier $f(\cdot)$ of the form $f(\cdot) = (g(\cdot), b_1, b_2, \ldots, b_{K-1})$, where g is defined by a parameter vector $\beta \in \mathbb{R}^D$ ($g(x) = x^\top \beta \mapsto \mathbb{R}$) and $\{b_1, \ldots, b_{K-1}\}$ are bias terms. Thus, $f(\cdot)$ can be represented as a linear function with parameter $\theta \in \mathbb{R}^{\tilde{D}}$ as $\theta = [\beta, b_1, \ldots, b_{K-1}]^\top$, with $f(x^k) = x^{k\top}\theta = x^\top \beta + b_k$. The authors guaranteed (Thm.2, [15]) when $C_{y,r(x)}$ is convex, the bias terms are *rank monotone* such that $b_1 \geq b_2 \geq \cdots \geq b_{K-1}$, therefore $f(x^1) \geq f(x^2) \geq \cdots \geq f(x^{K-1})$. This justifies the ranking rule of predicting the ordinal class of a new instance $x_* \in \mathbb{R}^D$ by

$$r(x_*) = 1 + \textstyle\sum_{k=1}^{K-1} 1[f(x_*^k) > 0]. \tag{4}$$

In this paper, we consider the convex absolute loss $C_{y,r(x)} = |y - r(x)|$ in the reduction to binary classification to ensure the biases to be rank monotone as described by the authors. As a result, $w_i^k = 1$ for all i, k.

4 DIVIDE AND CONQUER ALGORITHM FOR LARGE SCALE LOGISTIC CLASSIFICATION

The critical part of the reduction technique is the expansion of the training set, as evidenced in (2). The training set increases by $K - 1$ times its original size. Even for moderately large datasets, such expansions can lead to greater computational burden. It might become impossible to store the expanded dataset in a single machine, or at least, load it into the main memory, which would lead to substantial increase in training time.A natural technique might be to partition the full training dataset and learn a classifier coefficient on the individual parts and somehow combine the coefficients to get a final, global coefficient. However, each part may have insufficient sample size to yield a stable coefficient estimate due to overfitting. Additionally, regularized methods that prevent overfitting usually give biased estimates such that the combined estimate is also biased. In this section, we propose the robust inverse variance weighted average (**RIVWA**) method to address these challenges.

First, we divide the full ordinal data into M parts. For ease of exposition, we assume data is divided equally, with each part containing $n = N/M$ of the original training set (Note that the algorithm can handle data parts that are not equally sized; smaller parts will have smaller inverse variance weights; thus smaller contribution to the final result. The equal division is for technical convenience). We assume that the feature dimension D is fixed so that coefficient estimates from separate parts can be combined. We advise to choose M not too large to ensure $n > D$, and M not too small to ensure benefiting from DC.

Let (Y_m, X_m) denote the m-th part after the expansion by (2), with instance space dimension being $\tilde{n} = (K-1)n$ and feature space dimension being $\tilde{D} = D + K - 1$. Thus X_m is an $\tilde{n} \times \tilde{D}$ matrix where each row is an expanded instance x_i^k, i.e., $X_m = [x_{m,1}^1, \ldots, x_{m,1}^{K-1}, \ldots, x_{m,n}^1, \ldots, x_{m,n}^{K-1}]^\top$, and Y_m is a vector of length $\tilde{n}$, i.e., $Y_m = [y_{m,1}^1, \ldots, y_{m,1}^{K-1}, \ldots, y_{m,n}^1, \ldots, y_{m,n}^{K-1}]^\top$.

For convenience of representation, we use iterator $l = 1, \ldots, \tilde{n}$ to iterate through X_m and Y_m. For each of the data block, we consider the ℓ_1 regularized logistic regression [29] which employs the lasso penalty on the loss function to learn the coefficient vector θ:

$$\hat{\theta}_m = \operatorname{argmin}_{\theta \in \mathbb{R}^{\tilde{D}}} \{\tfrac{1}{\tilde{n}} \textstyle\sum_{l=1}^{\tilde{n}} \left\{\log\left(1 + e^{x_{m,l}{}^\top \theta}\right) - y_{m,l} x_{m,l}{}^\top \theta\right\} + \lambda ||\theta||_1\}, \tag{5}$$

where $|| \cdot ||_1$ is the ℓ_1 norm and λ is the penalty factor. From (5), we get a sparse estimate of regression coefficients $\hat{\theta}_m$ for the m-th block. In our experiments, we use the Python library `sklearn` with the `liblinear` solver to obtain $\hat{\theta}_m$ [11].

Next, we obtain the robust inverse variance by

$$\hat{\Sigma}_m^{-1}(\hat{\theta}_m) = X_m^\top V_m(\hat{\theta}_m) X_m, \tag{6}$$

where $V_m(\hat{\theta}_m)$ is an $\tilde{n} \times \tilde{n}$ diagonal matrix with diagonal elements $v_{l,l} = \sigma(x_{m,l}^\top \hat{\theta}_m)(1 - \sigma(x_{m,l}^\top \hat{\theta}_m))$, $l = 1, \ldots, \tilde{n}$.

The de-biased coefficient vector is obtained by

$$\hat{\theta}_m^c = \hat{\theta}_m + \hat{\Sigma}_m(\hat{\theta}_m) X_m^\top (Y_m - \hat{Y}_m), \tag{7}$$

where $\hat{Y}_m = [\hat{y}_{m,1}, \ldots, \hat{y}_{m,\tilde{n}}]^\top$ with $\hat{y}_{m,l} = \sigma(x_{m,l}^\top \hat{\theta}_m)$, $l = 1, \ldots, \tilde{n}$. The de-biased coefficient vector $\hat{\theta}_m^c$ is an approximation to the coefficient estimated when $\lambda = 0$. Eq. (7) provides a convenient way to quickly compute $\hat{\theta}_m^c$ instead of solving (5) again at $\lambda = 0$. We obtain $\hat{\theta}_m^c$ and $\hat{\Sigma}_m^{-1}(\hat{\theta}_m)$ for each of the data part, $m = 1, \ldots, M$, in parallel. Finally, we get the combined RIVWA estimate:

$$\hat{\theta} = \left\{ \sum_{m=1}^M \hat{\Sigma}_m^{-1}(\hat{\theta}_m) \right\}^{-1} \left\{ \sum_{m=1}^M \hat{\Sigma}_m^{-1}(\hat{\theta}_m) \hat{\theta}_m^c \right\}. \tag{8}$$

Note that when estimating the inverse variance weights, we plug in the sparse regularized estimates $\hat{\theta}_m$ to avoid overfitting and ensure numerical robustness of $\{\hat{\Sigma}_m\}_{m=1}^M$. The computation of $\hat{\theta}_m$ is mainly for the purpose of stabilizing the robust inverse variances. However, the average is taken across the de-biased estimates $\{\hat{\theta}_m^c\}_{m=1}^M$ to ensure unbiasedness and consistency of $\hat{\theta}$. Eq. (7) gives a direct and simple way to use the penalized $\hat{\theta}_m$ to compute the unpenalized counterpart $\hat{\theta}_m^c$ when $\lambda = 0$.

In contrast, the classic inverse variance weighted average (IVWA) (see related work section) uses the unregularized coefficient estimates in calculation of the inverse variance weights, which often leads to overfitting in individual data parts, resulting in predicted class probabilities being very close to the boundary (i.e., 0 or 1).

5 THEORETICAL RESULTS

In this section, we provide the consistency and asymptotic normality properties of the RIVWA estimator $\hat{\theta}$ in (8). First, we establish the asymptotic properties of the de-biased estimators $\{\hat{\theta}_m^c\}_{m=1}^M$ from individual parts to show that the bias correction recovers the loss of asymptotic due to regularization. Then, we show that the combined estimator $\hat{\theta}$ achieves asymptotic normality with the most efficient variance. Let θ_0 be the unknown true underlying coefficient, which is the limiting value of the benchmark coefficient $\hat{\theta}_{BM}$ (obtained from training on the entire, undivided dataset), as $\tilde{N} \to \infty$.

Theorem 5.1. *Under conditions listed in the Appendix, for the m-th block, $\hat{\theta}_m^c$ in* (7) *is consistent, i.e., $\hat{\theta}_m^c \xrightarrow{p} \theta_0$, and asymptotically normally distributed, i.e., $\sqrt{\tilde{n}}(\hat{\theta}_m^c - \theta_0) \xrightarrow{d} \mathcal{N}(0, \Sigma_m(\theta_0))$, where $\Sigma_m^{-1}(\theta_0) = X_m^\top V_m(\theta_0) X_m$.*

Theorem 5.2. *Under the same conditions in Theorem 5.1, the combined estimator $\hat{\theta}$ in* (8) *has the same consistentency and asymptotically normality property as $\hat{\theta}_{BM}$, i.e., $\hat{\theta} \xrightarrow{p} \theta_0$ and $\sqrt{\tilde{N}}(\hat{\theta} - \theta_0) \xrightarrow{d} \mathcal{N}(0, \Sigma(\theta_0))$, with $\Sigma^{-1}(\theta_0) = \sum_{m=1}^M \Sigma_m^{-1}(\theta_0)$.*

Theorem 5.1 states that the de-biased estimator (7) regains the asymptotic normality lost due to regularization of the likelihood. Theorem 5.2 establishes that the loss of efficiency due to lack of communication between data parts is ignorable in an asymptotic sense. The proofs we provide are motivated from a recent work on divide-and-conquer in generalized linear models [28]. Here we specifically focus on the logistic model, and also show that the parameters due to data expansion enjoy the same properties, which is crucial to outcome prediction and other applications in ordinal ranking. In addition, using results from Theorem 5.2, we can conduct statistical tests on the bias terms $b1, \ldots, b_{K-1}$ to compare the mean expected probabilities of different outcome levels.

6 EMPIRICAL RESULTS

In this section, we present results of extensive experiments on public and a proprietary ordinal dataset and a public classification dataset. The datasets span the domain of insurance industry, digital advertising, e-commerce and movie rating. Specifically, we compare our method with the benchmark and state-of-the-art DC methods.

6.1 Summary of Methods

We compare the following methods (citations, wherever required, are provided in related work section):

- Single memory training on full data with batch gradient descent method (benchmark method - BM)- the benchmark method trains a logistic classifier on the full dataset, by solving Eq. 1, [11].
- Single memory training on sub-sampled data with batch gradient descent method (50% subsampling- BM 50%)- this is same as the benchmark method, except that 50% of the data is randomly subsampled
- Single memory training on sub-sampled data with batch gradient descent method (20% subsampling- BM 20%)- same as above, with even more aggressive subsampling
- Single memory training on full data with follow-the-regularized-leader method (FTRL) [20];
- Single memory training with mini-batch gradient descent method (MBGD);
- DC with simple average (SA) [18, 30];
- DC with inverse variance weighted average (IVWA) [27];
- DC with majority voting (MV) [5];
- DC with robust inverse variance weighted average and bias correction (RIVWA), our proposed method.

Important points about the competing methods: BM (full and sub-sampled versions) and FTRL requires in-memory access to the full training data. MBGD, while ideally suited when we have in-memory access to full data, can be executed by reading mini-batches from secondary storage (as we have done). FTRL and MBGD are stochastic methods, that requires synchronization and data access at every iteration; thus they cannot be parallelized without paying extensive overhead charge. For FTRL, we iterate through all instances; for MBGD, we let the mini-batches to be 1/100 of the full training data. BM does not involve regularization, as it will not have the limiting distribution as shown in Theorem 5.2. Moreover, FTRL represents the regularized version of single memory, non-divided training method. All our experiments were conducted on machines with SSD for fast data reading and writing, but we did not involve GPUs due to facility limitation.

Table 1: Result on Prudential insurance dataset. Number of divisions $M = 100$ for DC methods. d_1: absolute differences between coefficient estimates of other methods and the benchmark; d_2: squared differences between coefficient estimates of other methods and the benchmark; change in abs_loss with respect to benchmark (%): the relative percentage change in absolute prediction loss with respect to benchmark (the smaller the better); time: computation time in seconds. Results are averaged across 10 repeated experiments and reported as mean ± sd. Best results are highlighted in bold.

Methods	$d_1(\hat{\theta}, \hat{\theta}_{BM})$	$d_2(\hat{\theta}, \hat{\theta}_{BM})$	Change in abs_loss w.r.t BM (%)	Time (sec)
Single memory				
BM	0.00	0.00	0.00	63.3 ± 1.9
BM 50%	2.68 ± 0.37	0.16 ± 0.07	0.49 ± 0.21	22.7 ± 0.6
BM 20%	5.82 ± 1.00	0.79 ± 0.66	2.14 ± 0.64	6.4 ± 0.3
FTRL	5.56 ± 0.21	0.51 ± 0.03	1.72 ± 0.40	109.8 ± 1.0
MBGD	4.63 ± 0.29	0.32 ± 0.04	0.64 ± 0.18	117.3 ± 20.2
DC with $M = 100$				
SA	11.45 ± 0.78	2.09 ± 0.29	1.23 ± 0.51	1.0 ± 0.1
IVWA	4.24 ± 0.19	0.33 ± 0.02	0.88 ± 0.15	1.0 ± 0.1
MV	4.23 ± 0.15	0.38 ± 0.01	0.72 ± 0.18	1.7 ± 0.4
RIVWA	**1.60** ± 0.06	**0.05** ± 0.002	**0.29** ± 0.14	**0.4** ± 0.1

6.2 Performance Evaluation

For performance evaluation, we report the following metrics: 1) absolute difference d_1 of an estimated coefficient $\hat{\theta}$ to that of the benchmark $\hat{\theta}_{BM}$:

$$d_1(\hat{\theta}, \hat{\theta}_{BM}) = ||\hat{\theta} - \hat{\theta}_{BM}||_1$$

2) squared difference d_2 of an estimated coefficient $\hat{\theta}$ to that of the benchmark $\hat{\theta}_{BM}$:

$$d_2(\hat{\theta}, \hat{\theta}_{BM}) = ||\hat{\theta} - \hat{\theta}_{BM}||_2^2$$

3) absolute prediction loss of an estimated $\hat{\theta}$ evaluated on the testing set, defined by

$$\text{abs\_loss}(Y_{test}, \hat{\theta}) = \frac{1}{n}\sum_{i=1}^{n} C_{y_i, \hat{y}_i(\hat{\theta})} = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i(\hat{\theta})|,$$

where y_i is the true ordinal label of the i-th instance in Y_{test} and $\hat{y}_i(\hat{\theta}) = 1 + \sum_{k=1}^{K-1} 1[x_i^{k\top}\hat{\theta} > 0]$ is its predicted label given $\hat{\theta}$ (this boils down to $0 - 1$ loss for binary classification) and
4) computation time in seconds, including the time used to read data from divided parts. Time of DC methods is calculated by the maximum time across all parallel procedures plus time used to combine the results. We also compare the above metrics at different choices of M.

Tuning parameters are selected to maximize the absolute prediction loss on the validation set such that for tuning parameter(s) $\lambda \in \mathcal{L}$, we select λ^* such that

$$\lambda^* = \arg\min_{\lambda \in \mathcal{L}} \text{abs\_loss}(Y_{valid}, \hat{\theta}_\lambda)$$

where $\hat{\theta}_\lambda$ is the coefficient vector obtained at tuning value λ. We use a grid search for $\lambda \in \{10^{-4}, 10^{-3}, \ldots, 10^{3}\}$.

6.3 Prudential Life Insurance Data

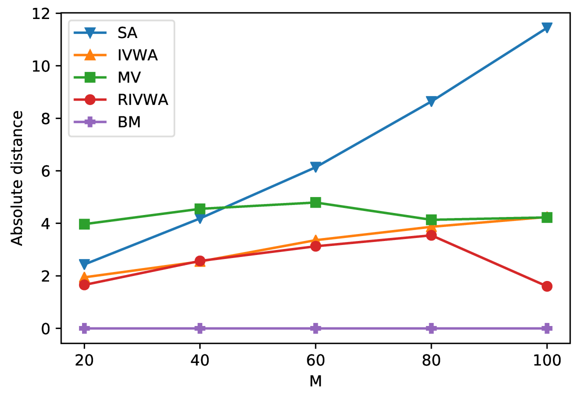
The Prudential dataset [1] is publicly available with an 8-level ordinal outcome of interest related to some undisclosed decision associated with an application in Prudential Financial, Inc. The dataset contains 59,381 labeled instances and has 144 features. We randomly split the dataset into 60% for training, 10% for validation, and 30% for testing.

Table 1 shows extensive experiment results comparing different methods. Results are averaged from 10 repetitions of the experiment. Our proposed RIVWA method produces the closest coefficient estimates to the benchmark in terms of d_1 and d_2, and achieves good prediction performance in terms of abs_loss$(Y_{test}, \hat{\theta})$. Additionally, our computational time is less than 1/100 that of BM, and similar to other DC methods. Figure 1 shows the change in d_1, d_2, abs_loss and time at different values of M. We can see that performance of RIVWA remains stable against different choices of M whereas other DC methods show worse performance as M increases, especially for SA and IVWA. The computation time for batch and stochastic methods usually don't scale well for large data, thus is not shown in Figure 1(d). DC methods only require one pass of the entire data, thus are much faster than the iterative methods. The time of DC methods usually depends on the number of parallel jobs. In this case, given N fixed, the computation time in general has a decreasing trend as M increases, assuming M parallel jobs can be executed all at once. It is also interesting to note that when M is small, the performance of different DC methods are very similar, because the sample size in each data division is large enough to provide good estimates. An extreme case is when $M = 1$, where all DC methods perform as similar as BM.

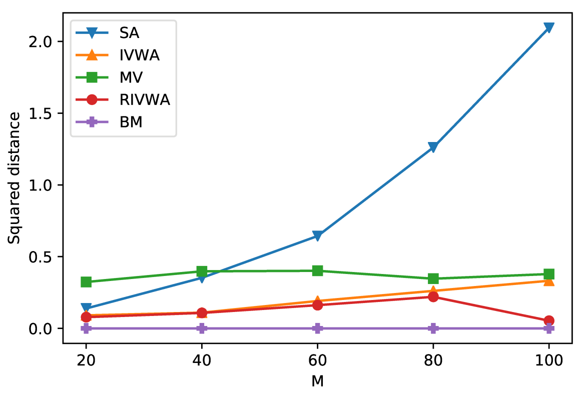
6.4 Additional Data Experiments

6.4.1 MovieLens Data . This is a popular public movie rating dataset [2] containing $20,000,263$ movie ratings by $138,493$ users of

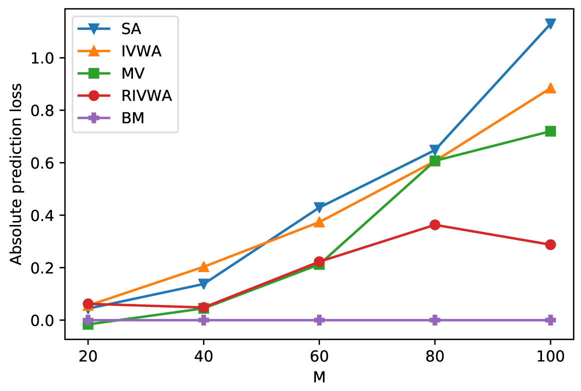
[1]Prudential Financial insurance data source https://www.kaggle.com/c/prudential-life-insurance-assessment
[2]MovieLens movie rating data source https://grouplens.org/datasets/movielens/20m/



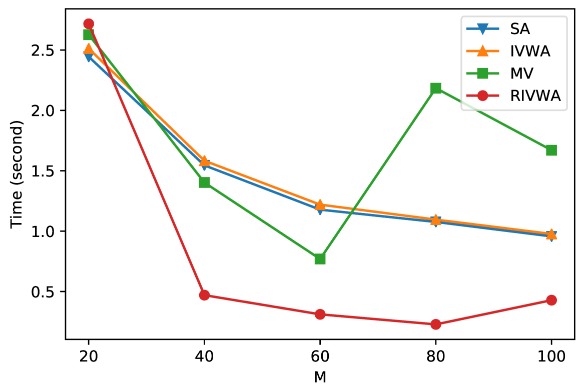
(a) Absolute distance from BM, d_1



(b) Squared distance from BM, d_2

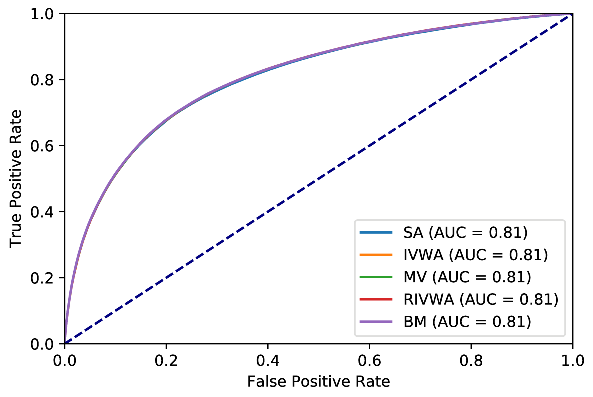


(c) Percentage change in absolute prediction loss, abs_loss(Y_{test}, $\hat{\theta}$), with respect to BM

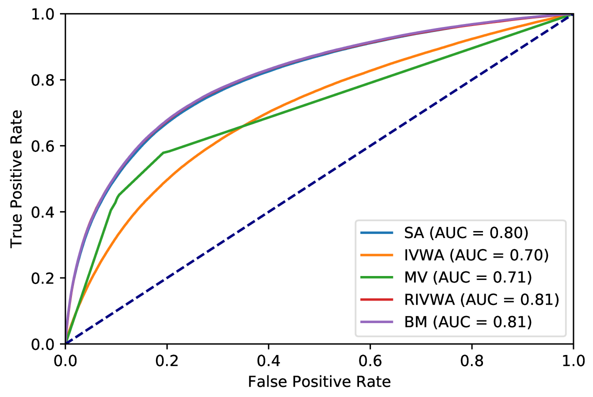


(d) Computation time in seconds

Figure 1: Performance of different methods on the Prudential insurance dataset at $M = 20, 40, 60, 80, 100$ for DC methods. Results are averaged across 10 repeated experiments. Metrics d_1, d_2 and abs_loss of BM is not affected by the choice of M. Computation time for BM is not displayed because it is much larger in scale as shown in Table 1.



(a) $M = 200$



(b) $M = 1000$

Figure 2: Area under curves results of Criteo dataset across multiple DC methods in comparison to BM.

Table 2: Result on the MovieLens movie rating dataset, Criteo conversion dataset, and an E-commerce advertising funnel dataset. Smaller the $d_1(\hat{\theta}, \hat{\theta}_{BM})$, $d_2(\hat{\theta}, \hat{\theta}_{BM})$ and percentage change in abs_loss, the better. Larger the AUC, the better. AUC is only available for the Criteo dataset because it has binary (two level) outcomes. Best results are highlighted in bold.

Methods	$d_1(\hat{\theta}, \hat{\theta}_{BM})$	$d_2(\hat{\theta}, \hat{\theta}_{BM})$	Change in abs_loss w.r.t. BM (%)	AUC	Time (sec)
MovieLens					
BM	0.00	0.00	0.00	—	953.2
DC with $M = 1000$					
SA	976.5	3245.64	7.21	—	11.1
IVWA	89.22	22.17	3.12	—	11.6
MV	79.59	17.07	**-0.02**	—	0.8
RIVWA	**8.40**	**0.47**	0.04	—	1.7
Criteo					
BM	0.00	0.00	0.00	0.81	71.1
DC with $M = 200$					
SA	61.00	8.40	0.31	0.81	0.7
IVWA	40.30	3.13	0.19	0.81	1.0
MV	36.34	2.61	0.19	0.81	3.6
RIVWA	**19.60**	**0.72**	**0.14**	0.81	2.4
DC with $M = 1000$					
SA	2850.43	15755.89	0.95	0.80	0.6
IVWA	172.53	53.08	35.55	0.70	1.1
MV	180.84	61.38	25.42	0.71	0.1
RIVWA	**46.30**	**4.76**	**-0.15**	**0.81**	1.2
E-commerce					
BM	0.00	0.00	0.00	—	635.4
DC with $M = 1000$					
SA	746.08	1979.43	-12.44	—	1.3
IVWA	**73.71**	36.18	199.76	—	1.8
MV	108.11	**29.01**	-18.96	—	1.0
RIVWA	79.09	29.75	**-27.47**	—	1.4
SkillCraft					
BM	0.00	0.00	0.00	—	3.5
M=10					
SA	4530.56	220837.20	50.75	—	7.6
IVWA	90.75	91.35	38.56	—	7.1
MV	69.36	60.41	-3.61	—	0.1
RIVWA	**20.66**	**5.01**	**-0.75**	—	0.2

27, 278 movies from 1995 − 2015. We use the following features for modeling: user Id, movie Id, rate year, movie year, genre categories, user tags and genome tags with relevance above 0.8.

Different from the Prudential dataset where the feature space dimension is fixed and small; the total number of features in this example is much larger than N and highly sparse. In order to estimate the variance of coefficients, we apply the hashing trick so that the features are reduced to a space of fixed dimension, which we fix at $2^{10} = 1,024$. Note that having a fixed feature space with lower dimension is critical for all types of inverse variance weighted methods, i.e., IVWA, MV and RIVWA, because they need to store the inverse variance matrix from each data division, which is a $\tilde{D} \times \tilde{D}$ matrix. If $\tilde{D}$ grows with $\tilde{N}$, the challenge in storing $\tilde{D} \times \tilde{D}$ weighting matrices compromises the scalability of these DC methods. There are DC methods that avoid using inverse variance weighting, however, they cannot achieve the asymptotic properties as presented in our theorems.

Each movie has a score from 0.5 to 5.0 with 0.5 increments (10 ordinal levels). We use a subsample of around 1 million instances for our experiments, which expands to 9 million binary instances. Similarly, we split the data as training, validation and testing set, with 60% for training, 10% for validation, and 30% for testing. We divide the training data into $M = 1000$ parts. Results are shown in Table 2 where we report similar performance metrics as before and compare between the different types of DC methods and the benchmark. We do not show other single memory methods besides BM because they require much longer training time and generally do not provide better performance than BM. From Table 2, we can see that both RIVWA and MV have predictions very close to BM, and RIVWA has the closest coefficient estimate to BM.

6.4.2 *Criteo Advertisement Data .*

While our paper is motivated by an ordinal ranking ranking problem (because it naturally leads to an expanded dataset), as we have mentioned at the beginning, the DC technique we proposed is applicable to general logistic regression for binary classification. We apply this method to a public advertisement dataset released by Criteo [3], which only has two outcomes, conversion versus no conversion. The dataset has 15 million instances. Due to memory limit restriction when computing BM, we randomly sampled $N = 2,120,698$ for training, $212,070$ for validation and $848,277$ for testing. We bucketize the continuous features into the nearest integer of their natural logged values. Then we apply the same hashing trick and map all features into a size of $D = 1,024$. Model tuning parameters are selected to maximize the AUC.

Table 2 shows results for BM and DC methods at $M = 200$ and $M = 1000$. Since the result is binary, we report both AUC and absolute prediction loss. When $M = 200$, the prediction performance is very similar for all types of DC methods. However, when $M = 1000$, only RIVWA and SA preserves comparable performance as BM. In all cases, RIVWA has the smallest deviation from BM in terms of coefficient estimation. Figure 2 shows the ROC curves of the different methods at $M = 200$ and $M = 1000$. Our proposed RIVWA method achieves identical performance in prediction as that of BM in both cases.

6.4.3 *E-commerce Advertisement Data.*

This dataset records online advertisement from a major internet based business. The dataset consists of 3 ordinal levels: an ad impression on a publisher website which did not lead to a click ($k = 1$), an ad impression which led to a click but did not lead to any product purchase ($k = 2$) and an ad impression which led to a click followed by a product purchase ($k = 3$). Naturally, a purchase is valued more than a click, which is valued more than an impression which did not lead to a click (thus, a natural ordinal ranking is induced).

For training, we collected the impression, click and purchase data over a period of 1 week (with 2 day click attribution and 7 day purchase attribution). We took a single day's data for validation and another day's data for testing. The number of instances and features are in the millions and we apply the same hashing trick to project the original feature space into a fixed dimensional space. The training data are randomly divided into $M = 1000$ parts.

Table 2 shows results of different DC methods as well as the benchmark. We can see that RIVWA has the best prediction accuracy in terms of absolute prediction loss, and its estimate is close to that of benchmark. Although IVWA yields similar performance in terms of d_1 and d_2 to ours, it has a much larger prediction loss.

6.4.4 *SkillCraft Data.*

SkillCraft is a dataset [4] for users gaming league rank prediction. The objective of this data is to predict the league of each player using some features related to gaming behaviors. Including all two way interactions, we have 152 total number of features and 3395 instances. Because the data size is small, we report results for $M = 10$ in Table 2. Again, we do a train-validation-testing split according to the ratios 60% : 10% : 30%.

7 DISCUSSION

RIVWA places heavy theoretical emphasis on coefficient estimation because many downstream analyses are dependent on the estimation results, such as ranking prediction, calibration and estimation of probabilities. In such cases, RIVWA is more superior than methods that purely focus on prediction. A limitation of our method, as is for all variance-dependent DC methods, is that it requires the feature space to be fixed in dimension. For feature space that are not known in advance, an additional feature reduction step, such as hashing or embedding, is needed to project the unknown dimensional space into a fixed and lower dimensional feature space.

In the application examples considered in this paper, the numbers of unique class labels are small. Indeed, it is of great interest to extend the setting of classic multi-label ordinal/multinomial classification considered in this paper to the case where the number of labels K could reach hundres, thousands or even more. In principle, K being large will not compromise the computational efficiency of our divide-and-conquer algorithm because the number of data parts M can also increase to accomodate the large scale data expansion. However, challenges may arise pertaining to the issue of rare and unbalanced labels. For example, some data parts might not contain training samples from some of the rare classes, resulting in some parameters to be unidentifiable. Although a quick-and-dirty approach can be to bin the rare class labels into larger classes in the divided training steps, and then train another classifier for the binned rare labels on the combined dataset, a more systematic way for such extreme multi-label classification problem is worth exploring in future works under the divide-and-conquer framework.

[3]Criteo advertisement conversion data source http://research.criteo.com/criteo-releases-first-public-dataset-conversion-logs/

[4]SkillCraft gaming data source http://archive.ics.uci.edu/ml/datasets/skillcraft1+master+table+dataset

REFERENCES

[1] Abraham Bagherjeiran, Andrew O Hatch, and Adwait Ratnaparkhi. 2010. Ranking for the conversion funnel. In *Proceedings of SIGIR*. ACM, 146–153.

[2] Kuang-Yu Chang, Chu-Song Chen, and Yi-Ping Hung. 2010. A ranking approach for human ages estimation based on face images. In *Pattern Recognition (ICPR), 2010 20th International Conference on*. IEEE, 3396–3399.

[3] Sougata Chaudhuri, Abraham Bagherjeiran, and James Liu. 2017. Ranking and calibrating click-attributed purchases in performance display advertising. In *Proceedings of the 2017 AdKDD and TargetAd Workshop*. 145–152.

[4] Po-Lung Chen, Chen-Tse Tsai, Yao-Nan Chen, Ku-Chun Chou, Chun-Liang Li, Cheng-Hao Tsai, Kuan-Wei Wu, Yu-Cheng Chou, Chung-Yi Li, Wei-Shih Lin, et al. 2011. A linear ensemble of individual and blended models for music rating prediction. In *Proceedings of the 2011 International Conference on KDD Cup 2011-Volume 18*. JMLR. org, 21–60.

[5] Xueying Chen and Min-ge Xie. 2014. A split-and-conquer approach for analysis of extraordinarily large data. *Statistica Sinica* (2014), 1655–1684.

[6] Wei Chu and S Sathiya Keerthi. 2005. New approaches to support vector ordinal regression. In *Proceedings of the 22nd international conference on Machine learning*. ACM, 145–152.

[7] Koby Crammer and Yoram Singer. 2002. Pranking with ranking. In *Advances in neural information processing systems*. 641–647.

[8] Wan-Yu Deng, Qing-Hua Zheng, Shiguo Lian, Lin Chen, and Xin Wang. 2010. Ordinal extreme learning machine. *Neurocomputing* 74, 1 (2010), 447–456.

[9] John C Duchi, Alekh Agarwal, and Martin J Wainwright. 2012. Dual averaging for distributed optimization: Convergence analysis and network scaling. *IEEE Transactions on Automatic control* 57, 3 (2012), 592–606.

[10] Bradley Efron and Robert J Tibshirani. 1994. *An introduction to the bootstrap*. CRC press.

[11] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A Library for Large Linear Classification. *J. Mach. Learn. Res.* 9 (June 2008), 1871–1874.

[12] Eibe Frank and Mark Hall. 2001. A simple approach to ordinal classification. In *European Conference on Machine Learning*. Springer, 145–156.

[13] Bin Gu, Victor S Sheng, Keng Yeow Tay, Walter Romano, and Shuo Li. 2015. Incremental support vector learning for ordinal regression. *IEEE Transactions on Neural networks and learning systems* 26, 7 (2015), 1403–1416.

[14] Yehuda Koren and Joe Sill. 2011. OrdRec: an ordinal model for predicting personalized item rating distributions. In *Proceedings of the fifth ACM conference on Recommender systems*. ACM, 117–124.

[15] Ling Li and Hsuan-Tien Lin. 2007. Ordinal regression by extended binary classification. *Advances in neural information processing systems* 19 (2007), 865.

[16] P. Li, C. Burges, and Q. Wu. 2007. McRank: Learning to Rank Using Multiple Classification and Gradient Boosting. In *Advances in Neural Information Processing Systems*, Vol. 19. The MIT Press, 897–904.

[17] Hsuan-Tien Lin and Ling Li. 2012. Reduction from cost-sensitive ordinal ranking to weighted binary classification. *Neural Computation* 24, 5 (2012), 1329–1367.

[18] Gideon S Mann, Ryan Mcdonald, Mehryar Mohri, Nathan Silberman, and Dan Walker. 2009. Efficient large-scale distributed training of conditional maximum entropy models. In *Advances in Neural Information Processing Systems*. 1231–1239.

[19] Peter McCullagh. 1980. Regression models for ordinal data. *Journal of the royal statistical society. Series B (Methodological)* (1980), 109–142.

[20] Brendan McMahan. 2011. Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 525–533.

[21] Angelia Nedic and Asuman Ozdaglar. 2009. Distributed subgradient methods for multi-agent optimization. *IEEE Trans. Automat. Control* 54, 1 (2009), 48–61.

[22] E Michael Nussbaum and Gregory Schraw. 2007. Promoting argument-counterargument integration in students' writing. *The Journal of Experimental Education* 76, 1 (2007), 59–92.

[23] Shyamsundar Rajaram, Ashutosh Garg, Xiang Sean Zhou, and Thomas S Huang. 2003. Classification approach towards ranking and sorting problems. In *European Conference on Machine Learning*. Springer, 301–312.

[24] S Sundhar Ram, Angelia Nedić, and Venugopal V Veeravalli. 2010. Distributed stochastic subgradient projection algorithms for convex optimization. *Journal of optimization theory and applications* 147, 3 (2010), 516–545.

[25] Steven L Scott, Alexander W Blocker, Fernando V Bonassi, Hugh A Chipman, Edward I George, and Robert E McCulloch. 2016. Bayes and big data: The consensus Monte Carlo algorithm. *International Journal of Management Science and Engineering Management* 11, 2 (2016), 78–88.

[26] Amnon Shashua and Anat Levin. 2003. Ranking with large margin principle: Two approaches. In *Advances in neural information processing systems*. 961–968.

[27] Alexander J Sutton and Julian Higgins. 2008. Recent developments in meta-analysis. *Statistics in medicine* 27, 5 (2008), 625–650.

[28] Lu Tang, Ling Zhou, and Peter X.-K. Song. 2016. Method of divide-and-combine in regularised generalised linear models for big data. *arXiv:1611.06208v1 [stat.ME]* (2016).

[29] Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* (1996), 267–288.

[30] Yuchen Zhang, Martin J Wainwright, and John C Duchi. 2012. Communication-efficient algorithms for statistical optimization. In *Advances in Neural Information Processing Systems*. 1502–1510.