GLMix: Generalized Linear Mixed Models For Large-Scale Response Prediction

Xianxing Zhang, Yitong Zhou, Yiming Ma, Bee-Chung Chen, Liang Zhang, Deepak Agarwal
LinkedIn
Mountain View, CA, USA
{xazhang, yizhou, yma, bchen, lizhang, dagarwal}@linkedin.com

ABSTRACT

Generalized linear model (GLM) is a widely used class of models for statistical inference and response prediction problems. For instance, in order to recommend relevant content to a user or optimize for revenue, many web companies use logistic regression models to predict the probability of the user's clicking on an item (e.g., ad, news article, job). In scenarios where the data is abundant, having a more fine-grained model at the user or item level would potentially lead to more accurate prediction, as the user's personal preferences on items and the item's specific attraction for users can be better captured. One common approach is to introduce ID-level regression coefficients in addition to the global regression coefficients in a GLM setting, and such models are called generalized linear mixed models (GLMix) in the statistical literature. However, for big data sets with a large number of ID-level coefficients, fitting a GLMix model can be computationally challenging. In this paper, we report how we successfully overcame the scalability bottleneck by applying parallelized block coordinate descent under the Bulk Synchronous Parallel (BSP) paradigm. We deployed the model in the LinkedIn job recommender system, and generated 20% to 40% more job applications for job seekers on LinkedIn.

1. INTRODUCTION

Accurate prediction of users' responses to items is one of the core functions of many recommendation applications. Examples include recommending movies, news articles, songs, jobs, advertisements, and so forth. Given a set of features, a common approach is to apply generalized linear models (GLM). For example, when the response is numeric (e.g., rating of a user to a movie), a linear regression on the features is commonly used to predict the response. For the binary response scenario (e.g. whether to click or not when a user sees a job recommendation), logistic regression is often used. Sometimes the response is a count (e.g., number of times a user listens to a song), and Poisson regression becomes a natural choice. All of the above models are special cases of GLM.

The features available in recommender systems often include user features (*e.g.,* age, gender, industry, job function) and item features (*e.g.,* title and skills for jobs, title and named entities for news articles). An approach that is widely adopted in industry to model interactions between users and items is to form the outer (cross) product of user and item features, followed by feature selection to reduce the dimensionality and mitigate the problem of overfitting. In reality, we often observe a lot of heterogeneity in the amount of data per user or item that cannot be sufficiently modeled by user/item features alone, which provides an opportunity to improve model accuracy by adding more granularity to the model. Specifically, for a user who has interacted with many items in the past, we should have sufficient data to fit regression coefficients that are specific to that user to capture his/her personal interests. Similarly, for an item that has received many users' responses, it is beneficial to model its popularity and interactions with user features through regression coefficients that are specific to the item.

One common approach to capture such behavior of each individual user and item is to use ID-level features, i.e., the outer product of user IDs and item features, and the outer product of item IDs and user features. Models with ID-level features are usually referred to as generalized linear mixed models (GLMix) in Statistics [15]. Although conceptually simple, it can generate a very large number of regression coefficients to be learned. For example, for a data set of 10 million users, and each user with 1,000 non-zero coefficients on item features, the total number of regression coefficients can easily go beyond 10^{10}. Therefore, fitting GLMix models for big data is computationally challenging. Dimension reduction methods such as feature hashing [1] or principal component analysis can reduce the number of features. However, they also reduce our ability to interpret the model or explain the predictions in the original feature space (*e.g.*, at the user's ID level), making it difficult to debug or investigate system issues or user complaints.

1.1 Our contributions

In this paper we develop a parallel block-wise coordinate descent (PBCD) algorithm under the Bulk Synchronous Parallel (BSP) paradigm [21] for GLMix model, with a novel model score movement scheme that allows both the coefficients and the data to stay in local nodes. We show that our algorithm successfully scales model fitting of GLMix for very large data sets. We empirically demonstrate that this conceptually simple class of models achieves high accuracy

KDD '16 Aug 13–17, 2016, San Francisco, CA, USA

DOI: http://dx.doi.org/10.1145/2939672.2939684

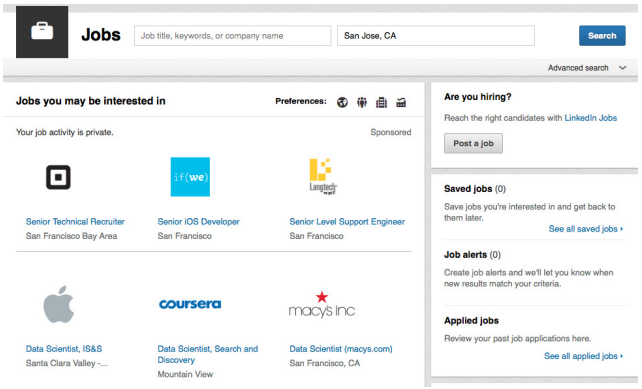

Figure 1: A snapshot of the LinkedIn jobs homepage.

on two public data sets and the LinkedIn job recommendation data set. We also report our successful deployment of GLMix in LinkedIn's job recommendation production system, which improved the total number of job applications from active job seekers by 20% to 40%. To our knowledge, this is the first paper that discusses the practical lessons learned to scale GLMix in real web recommender system applications.

We also extend GLMix by adding a matrix factorization component in order to further leverage recent advancements in scaling up matrix factorization in the distributed environment [24], and see some slight improvement on LinkedIn job recommendation data. Since the parallelization of matrix factorization is well studied, in this paper we mainly focus on the scalability and parallelization of fitting GLMix.

We provide an implementation of the GLMix model with Apache Spark[1], and the code base is open sourced as part of the Photon-ML library[2].

1.2 Organization

The paper is organized as follows. We start by introducing the LinkedIn job recommendation problem as our motivating example and describing the formulation of the GLMix model in such a context in Section 2. The fitting algorithm using PBCD and the implementation details are described in Section 3. We show our offline and online experiments for LinkedIn job recommendation in Section 4, and show results for the other two public data sets in Section 5. We review the literature background of GLMix and related work in Section 6. We conclude the paper in Section 7.

2. OUR MODEL

In this section, we describe the generalized linear mixed model (GLMix) in detail. We start with our motivating example in Section 2.1, the LinkedIn job recommendation problem, to provide the context for GLMix. We introduce the model in Section 2.2 and later describe a general formulation in Section 2.3. Section 2.4 describes extensions of GLMix such as adding a matrix factorization component.

2.1 Job Recommendation at LinkedIn

As the world's largest professional social network, LinkedIn provides a unique value proposition for its over 400M members to connect with all kinds of professional opportunities for their career growth. One of the most important products is the Jobs Homepage, which serves as a central place for members with job seeking intention to come and find good jobs to apply for. Figure 1 is a snapshot of the LinkedIn Jobs Homepage. One of the main modules on the page is "Jobs you may be interested in", where relevant job thumbnails are recommended to members based on their public profile data and past activity on the site. If a member is interested in a recommended job, she can click on it to go to the job detail page, where the original job post is shown with information such as the job title, description, responsibilities, required skills and qualifications. The job detail page also has an "apply" button which allows the member to apply for the job with one click, either on LinkedIn or on the website of the company posting the job. One of the key success metrics for LinkedIn jobs business is the total number of *job application clicks* (i.e the number of clicks on the "apply" button), which is the focus for the job recommendation problem in this paper. Another second-order metric is called "job detail views", which represents the total number of clicks generated from the "Jobs you may be interested in" module to the job details pages. We consider job application clicks to be the key metric to optimize for, instead of job detail views, because the job posters mostly care about how many applications they receive, instead of how many people view their job postings.

2.2 GLMix Model

Now we consider the GLMix model for the job recommendation problem. To measure whether job j is a good match for a member m and to select the best jobs according to this measure, the key is to predict the probability that member m would apply for job j given an impression on the "Jobs you may be interested in" module. Let y_{mjt} denote the binary response of whether member m would apply for job j in context t, where the context usually includes the time and location where the job is shown. We use $\boldsymbol{q}_m$ to denote the feature vector of member m, which includes the features extracted from the member's public profile, e.g., the member's title, job function, education history, industry, etc. We use $\boldsymbol{s}_j$ to denote the feature vector of job j, which includes features extracted from the job post, e.g. the job title, desired skills and experiences, etc. Let $\boldsymbol{x}_{mjt}$ represent the overall feature vector for the (m, j, t) triple, which can include $\boldsymbol{q}_m$ and $\boldsymbol{s}_j$ for feature-level main effects, the outer product between $\boldsymbol{q}_m$ and $\boldsymbol{s}_j$ for interactions among member and job features, and features of the context. We assume that $\boldsymbol{x}_{mjt}$ does not contain member IDs or item IDs as features, because IDs will be treated differently from regular features.

The GLMix model for predicting the probability of member m applying for job j using logistic regression is:

$$g(E[y_{mjt}]) = \boldsymbol{x}'_{mjt}\boldsymbol{b} + \boldsymbol{s}'_j\boldsymbol{\alpha}_m + \boldsymbol{q}'_m\boldsymbol{\beta}_j, \quad (1)$$

where $g(E[y_{mjt}]) = \log \frac{E[y_{mjt}]}{1-E[y_{mjt}]}$ is the link function, $\boldsymbol{b}$ is the global coefficient vector (also called *fixed effect* coefficients in the statistical literature); and $\boldsymbol{\alpha}_m$ and $\boldsymbol{\beta}_j$ are the coefficient vectors specific to member m and job j, respectively. $\boldsymbol{\alpha}_m$ and $\boldsymbol{\beta}_j$ are called *random effects* coefficients, which capture member m's personal preference on different item features and item j's attraction for different member features.

[1]https://spark.apache.org
[2]https://github.com/linkedin/photon-ml

Priors: To mitigate the risk of overfitting due to the large number of parameters and sparsity of the data, we put the following Gaussian priors on both fixed effects and random effects:

$$\boldsymbol{b} \sim N(0, \tfrac{1}{\lambda_b} I), \quad \boldsymbol{\alpha}_m \sim N(0, \tfrac{1}{\lambda_\alpha} I), \quad \boldsymbol{\beta}_j \sim N(0, \tfrac{1}{\lambda_\beta} I), \quad (2)$$

where I is the identity matrix, $\frac{1}{\lambda_b}$ is the prior variance of $\boldsymbol{b}$, and $\frac{1}{\lambda_\alpha}$ and $\frac{1}{\lambda_\beta}$ are the prior variances of $\boldsymbol{\alpha}_m$ and $\boldsymbol{\beta}_j$.

Model behavior: Like many other recommender system applications, we observe a lot of heterogeneity in the amount of data per member or job. There are new members to the site (hence almost no data) as well as members who have strong job search intentions and applied for many jobs in the past. Similarly, for jobs there are both popular and unpopular ones. For a member m with many responses to different items in the past, we are able to accurately estimate her personal coefficient vector $\boldsymbol{\alpha}_m$ and provide personalized predictions. On the other hand, if member m does not have much past response data, the posterior mean of $\boldsymbol{\alpha}_m$ will be close to zero, and the model for member m will fall back to the global fixed effect component $\boldsymbol{x}'_{mjt}\boldsymbol{b}$. The same behavior applies to the per-job coefficient vector $\boldsymbol{\beta}_j$.

2.3 General Formulation

We now consider a more generic formulation of GLMix where more than two sets of random effects can be present in the model. This is useful for scenarios such as multi-context modeling, where we can have random effects to model the interactions between the context id and user/item features. The general formulation of GLMix is provided in the following equation:

$$\begin{aligned} & g(E[y_n]) = \boldsymbol{x}'_n\boldsymbol{b} + \sum_{r\in\mathcal{R}} \boldsymbol{z}'_{rn}\boldsymbol{\gamma}_{r,i(r,n)}, \\ & \boldsymbol{b} \sim p(\boldsymbol{b}), \quad \boldsymbol{\gamma}_{rl} \sim p(\boldsymbol{\gamma}_{rl}), \quad \forall r \in \mathcal{R}, 1 \le l \le N_r \end{aligned} \quad (3)$$

Comparing equation (3) with (1), the following new notations are introduced: Let $\mathcal{R}$ denote the collection of random effect types being modeled. Also let $i(r,n)$ denote an indexing function that retrieves the index of random effect type r in the n-th training sample; for example, if random effect type r corresponds to the per-job random effect, $i(r,n)$ then returns the job ID associated with sample n. Given the definition of the indices, we use $\boldsymbol{\gamma}_{r,i(r,n)}$ to denote random effect coefficient vector and $\boldsymbol{z}_{rn}$ as the corresponding feature vector, for random effect type r in the n-th sample. For the specific case of job recommendations, there are two types of random effects: $\mathcal{R} = \{\text{member}, \text{job}\}$, and n is a per-sample index that represents the triple (m, j, t). For the member-level random effect (i.e., r=member), $\boldsymbol{z}_{rn}$ represents $\boldsymbol{s}_j$ and $\boldsymbol{\gamma}_{r,i(r,n)}$ represents $\boldsymbol{\alpha}_m$. For the job-level random effect (i.e., r=job), $\boldsymbol{z}_{rn}$ represents $\boldsymbol{q}_m$ and $\boldsymbol{\gamma}_{r,i(r,n)}$ represents $\boldsymbol{\beta}_j$.

We generalize the Gaussian priors of fixed effects $\boldsymbol{b}$ and random effects to $p(\cdot)$, and also use N_r to denote the total number of instances for random effect type r, e.g., when r represents member, N_r represents the total number of members in the data set.

2.4 Extensions

We note that GLMix can be extended by incorporating other popular modeling approaches in recommender systems, such as matrix factorization [11]. For example, for the job recommendation application, a simple extension to the model in Equation (1) with an additional matrix factorization component is:

$$g(E[y_{mjt}]) = \boldsymbol{x}'_{mjt}\boldsymbol{b} + \boldsymbol{s}'_j\boldsymbol{\alpha}_m + \boldsymbol{q}'_m\boldsymbol{\beta}_j + \boldsymbol{u}'_m\boldsymbol{v}_j, \quad (4)$$

where $\boldsymbol{u}'_m$ and $\boldsymbol{v}_j$ are k-dimensional member and job factors, respectively.

3. ALGORITHM

In this section we describe our scalable algorithm for the GLMix model using parallel block-wise coordinate descent (PBCD). We start with the algorithm itself in Section 3.1, and in Section 3.2 we discuss the implementation details under the Bulk Synchronous Parallel (BSP) [21] paradigm and lessons learned, such as how to take the network I/O communication complexity into account and how to reduce the size of the random effect coefficients.

Throughout the section, we use the notation of the general formulation described in Section 2.3. Given the set of sample responses $\boldsymbol{y}(\Omega) = \{y_n\}_{n\in\Omega}$, where Ω is the set of sample indices, the main parameters to learn are the fixed effect coefficients $\boldsymbol{b}$ and random effect coefficients $\boldsymbol{\Gamma} = \{\Gamma_r\}_{r\in\mathcal{R}}$, where $\Gamma_r = \{\boldsymbol{\gamma}_{rl}\}_{l=1}^{N_r}$. The hyper-parameters for the priors of $\boldsymbol{b}$ and $\boldsymbol{\Gamma}$, such as λ_b, λ_α and λ_β for the job recommendation case, can be tuned through cross-validation.

For the general formulation of the GLMix model specified in Equation (3), the posterior mode of $\boldsymbol{b}$ and $\boldsymbol{\Gamma}$ can be found through the following optimization problem:

$$\max_{\boldsymbol{b}, \{\Gamma_r\}_{r\in\mathcal{R}}} \sum_{n\in\Omega} \log p(y_n|s_n) + \log p(\boldsymbol{b}) + \sum_{r\in\mathcal{R}} \sum_{l=1}^{N_r} \log p(\boldsymbol{\gamma}_{rl}), \quad (5)$$

where s_n is defined as

$$s_n = \boldsymbol{x}'_n\boldsymbol{b} + \sum_{r\in\mathcal{R}} \boldsymbol{z}'_{rn}\boldsymbol{\gamma}_{r,i(r,n)}, \quad (6)$$

and $p(y_n|s_n)$ is the underlying likelihood function of the link function $g[E[y_n]]$ for response y_n given $\boldsymbol{b}$ and $\boldsymbol{\Gamma}$.

3.1 Parallel Block-wise Coordinate Descent

Traditionally, the fitting algorithms for random effect models require $\boldsymbol{\Gamma}$ to be integrated out either analytically or numerically (literature review can be found in Section 6), which becomes infeasible when facing industry-scale large data sets. Similarly, both deterministic and MCMC sampling based algorithms that operate on $\boldsymbol{\Gamma}$ as a whole become cumbersome.

In Algorithm 1, we propose a parallel block-wise coordinate descent based iterative conditional mode algorithm, where the posterior mode of the random effect coefficients Γ_r for each random effect r is treated as a block-wise coordinate to be optimized in the space of unknown parameters. Given the scores $\boldsymbol{s}$ defined in equation (6), the optimization problems for updating the fixed effects $\boldsymbol{b}$ and the random effects $\boldsymbol{\Gamma}$ are provided in Equation (7) and (8), respectively:

$$\boldsymbol{b} = \arg\max_{\boldsymbol{b}} \left\{ \log p(\boldsymbol{b}) + \sum_{n\in\Omega} \log p(y_n|s_n - \boldsymbol{x}'_n\boldsymbol{b}^{old} + \boldsymbol{x}'_n\boldsymbol{b}) \right\} \quad (7)$$

Algorithm 1: Parallel block-wise coordinate descent (PBCD) for GLMix

```
while not converged do
    Update fixed effect b as in (7)
    foreach n ∈ Ω in parallel do
        Update s_n as in (9)
    foreach r ∈ R do
        foreach l ∈ {1, ..., N_r} in parallel do
            Update random effect γ_rl as in (8)
        foreach n ∈ Ω in parallel do
            Update s_n as in (10)
```

$$\boldsymbol{\gamma}_{rl} = \arg\max_{\boldsymbol{\gamma}_{rl}} \Bigg\{ \log p(\boldsymbol{\gamma}_{rl}) + \sum_{n|i(r,n)=l} \log p(y_n | s_n - \boldsymbol{z}'_{rn}\boldsymbol{\gamma}^{old}_{rl} + \boldsymbol{z}'_{rn}\boldsymbol{\gamma}_{rl}) \Bigg\} \quad (8)$$

As in most coordinate descent algorithms, we perform incremental updates for $\boldsymbol{s} = \{s_n\}_{n\in\Omega}$ for computational efficiency. More specifically, when the fixed effects $\boldsymbol{b}$ get updated, Equation (9) is applied for updating $\boldsymbol{s}$; and when the random effects $\boldsymbol{\Gamma}$ get updated, Equation (10) is used.

$$s_n^{new} = s_n^{old} - \boldsymbol{x}'_n \boldsymbol{b}^{old} + \boldsymbol{x}'_n \boldsymbol{b}^{new} \quad (9)$$

$$s_n^{new} = s_n^{old} - \boldsymbol{z}'_{rn} \boldsymbol{\gamma}^{old}_{r,i(r,n)} + \boldsymbol{z}'_{rn} \boldsymbol{\gamma}^{new}_{r,i(r,n)} \quad (10)$$

Note that Equation (7) and (8) are both standard optimization problems for generalized linear model, for which many numerical optimizers are available for large data sets, such as L-BFGS, SGD, or TRON, and they can be easily parallelized for scalability [9, 14].

3.2 Implementation Details

In this section, we discuss some details on how to implement the parallel block-wise coordinate descent algorithm (PBCD) for GLMix described in Algorithm 1 under the BSP paradigm. Other alternatives for implementing PBCD such as using parameter server are discussed in Section 6.

We first introduce some additional notation, and then discuss the implementation details for updating the fixed effects and the random effects, respectively. An overview of the k-th iteration of the block coordinate descent algorithm for GLMix is given in Figure 2.

3.2.1 Some Notations

At iteration k of Algorithm 1, let $\boldsymbol{s}^k$ denote the current value of $\boldsymbol{s} = \{s_n\}_{n\in\Omega}$. Let P denote the dimension of the fixed effect feature space, i.e., $\boldsymbol{x}_n \in \mathbb{R}^P$, and P_r denote the dimension of the feature space for random effect r, i.e., $\boldsymbol{z}_{rn} \in \mathbb{R}^{P_r}$. And we use C to denote the overall dimension of the feature space, that is:

$$C = P + \sum_{r\in\mathcal{R}} P_r N_r, \quad (11)$$

where N_r denotes the number of random effects of type r (e.g., number of users). For the set of sample responses $\boldsymbol{y}(\Omega) = \{y_n\}_{n\in\Omega}$, we use $|\Omega|$ to denote the size of Ω, i.e. the total number of training samples. We also let $|\mathcal{R}|$ be the number of types of random effects, and M be the number of computing nodes in the cluster. These numbers will be used to compute and discuss the network I/O cost of our proposed approach, which is usually one of the major bottlenecks for an algorithm to scale up in a distributed computing environment [27, 16]

3.2.2 Updating Fixed Effects

We consider the details of updating the fixed effect $\boldsymbol{b}$ at iteration k, following Equation (7). We start with preparing the training data with both the feature set $\boldsymbol{x}_n$ and the latest score s_n^k, and partition them into M nodes. This is identified as Superstep 1 in Figure 2. Given the training data, numerous types of distributed algorithms can be applied to learn $\boldsymbol{b}$, such as [16, 6]. Here we adopt the all-reduce [1] type of implementation for generalized linear models as in MLlib [12]. First, the gradient of $\boldsymbol{b}$ for each sample n is computed and aggregated from each node to the master node. The master node computes the total gradient and updates $\boldsymbol{b}$ using optimizers such as L-BFGS. This corresponds to Superstep 2 in Figure 2. The new coefficients $\boldsymbol{b}^{new}$ are then broadcast back to each node together with $\boldsymbol{b}^{old}$ to update the score $\boldsymbol{s}$ as in Equation (9), in order to prepare for the next effect's update. This phase corresponds to Superstep 3 in Figure 2. Since the main network communication here is the transmission of $\boldsymbol{b}$ from the master node to the worker nodes, the overall network communication cost for one iteration of updating the fixed effects is $\mathcal{O}(MP)$. One trick we found empirically useful is that it sometimes improves convergence to update $\boldsymbol{b}$ multiple times before we update the random effects, for each iteration in Algorithm 1.

In summary, updating the fixed effects consists of three supersteps: (1) Preparing the training data with scores $\boldsymbol{s}$. (2) Updating the coefficients $\boldsymbol{b}$. (3) Updating the scores $\boldsymbol{s}$.

3.2.3 Updating Random Effects

The main challenge in designing a scalable algorithm for GLMix on data sets with a huge number of random effects is that the dimension of the random effect coefficient space can potentially be as large as $N_r P_r$. Therefore, if we naively apply the same approach as the one used in updating fixed effects, the network communication cost for updating the random effects for r becomes MN_rP_r. Given some data of moderate size for example, if $N_r = 10^6$, $P_r = 10^5$ and a cluster with $M = 100$, the network I/O cost amounts to 10^{13}! As a result, the key to making the algorithm scalable is to avoid communicating or broadcasting the random effect coefficient across the computing nodes in the cluster.

Before the random effect updating phase and as a preprocessing step, for each random effect r and ID l, we group the feature vectors $\boldsymbol{z}_{rn}$ to form a feature matrix $\boldsymbol{Z}_{rl}$, which consists of all the $\boldsymbol{z}_{rn}$ that satisfy $i(r,n) = l$. At iteration k and for random effect r, we shuffle the current values of $\boldsymbol{s} = \{s_n^k\}_{n\in\Omega}$ using the same strategy, *i.e.* for ID l we group s_n^k to form a vector $\boldsymbol{S}_l^k$, which consists of all the s_n^k that satisfies $i(r,n) = l$. With the right partitioning strategies, $\boldsymbol{S}_l^k$ can be made to collocate with the corresponding feature matrix $\boldsymbol{Z}_{rl}$, to prepare the training data for updating the random effects r, using Equation (8). This step corresponds to Superstep 4 in Figure 2. With the training data ready for each ID l, any optimizer (*e.g.*, L-BFGS) can be applied again to solve the optimization problem locally, such that the random effects γ_{rl} can be learned in parallel without

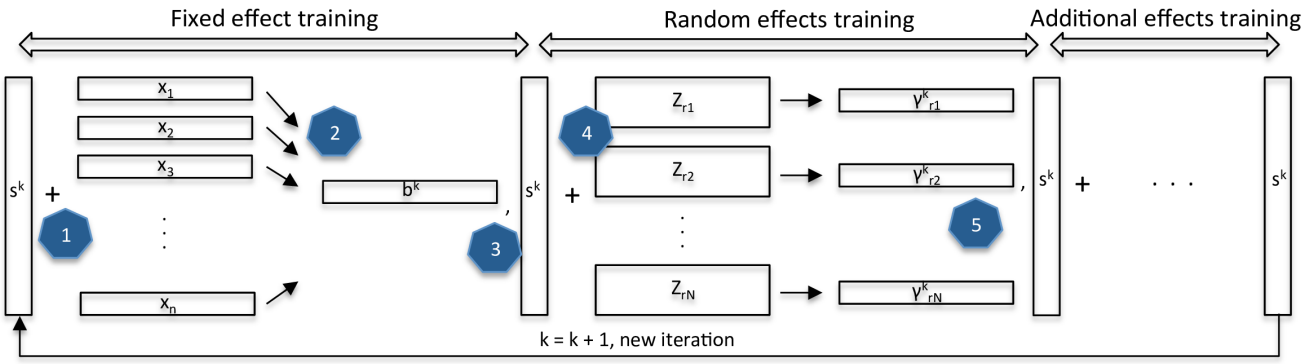

Figure 2: Overview of the kth iteration of the PBCD in BSP paradigm, with five supersteps highlighted in the blue boxes.

any additional network communication cost. Note that because both the coefficients and data collocate in the same node, the scores can be updated locally within the same step, and this phase corresponds to Superstep 5 in Figure 2. Also note that during the whole process, the random effect coefficients $\boldsymbol{\gamma}_{rl}$ live with the data and would never get communicated through the network; only $\boldsymbol{s}$ would get shuffled around the nodes through network I/O. As a result, the overall network communication cost for one iteration of updating one random effects is $\mathcal{O}(|\Omega|)$, and $\mathcal{O}(|\mathcal{R}||\Omega|)$ for $|\mathcal{R}|$ random effects.

Since the optimization problem in Equation (8) can be solved locally, we have an opportunity to apply some tricks that can further reduce the memory complexity C as defined in Equation (11). Note that although the overall feature space size is P_r for random effect r, sometimes the underlying dimension of the feature matrix $\boldsymbol{Z}_{rl}$ could be smaller than P_r, due to the lack of support for certain features. For example, a member who is a software engineer is unlikely to be served jobs with the required skill "medicine". Hence there will not be any data for the feature "job skill=medicine" for this member's random effects, and in such a scenario, $\boldsymbol{Z}_{rl}$ would end up with an empty column. As a result, for each random effect r and ID l, we can condense $\boldsymbol{Z}_{rl}$ by removing all the empty columns and reindexing the features to form a more compact feature matrix, which would also reduce the size of random effect coefficients γ_{rl} and potentially improve the overall efficiency of solving the local optimization problem in Equation (8). An example is shown in Figure 3, where we compare the random effect coefficient size before and after applying such a condensed data storage strategy on a data set consisting of four months' worth of LinkedIn's job recommendations. More details on the data set can be found in Section 4.

In summary, the updating of random effects consists of two supersteps: (1) Prepare the training data with scores $\boldsymbol{s}$. (2) Update the random effect coefficients and also the scores.

3.2.4 Summary

The overall I/O communication cost for our implementation for GLMix is $\mathcal{O}(MP + |\mathcal{R}||\Omega|)$. Since $|\mathcal{R}|$ is a small constant (usually less than 5), the communication cost is mainly bounded by the number of training instances $|\Omega|$.

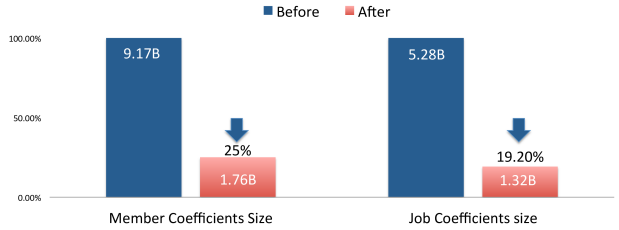

Figure 3: The member and job coefficient size before and after the reindexing and compression with the LinkedIn Job Recommendation data set.

This is a significant improvement compared to the naive implementation with a cost of $\mathcal{O}(MC)$, where C is defined in (11), which could approach $\mathcal{O}(MP_r|\Omega|)$ if we have a large number of random effects N_r that is of the same order as the number of the training instances $|\Omega|$ (*e.g.*, power law distribution). On the other hand, note that even with a large scale data set with *trillions* of training samples, the size of $\boldsymbol{s}$ is bounded to be a couple of terabytes. Given the recent advances in shuffling techniques [26], shuffling terabytes of data is no longer a significant challenge. At the same time, since the total number of random effects N_r no longer depends on the number of nodes in the cluster M, our algorithm has the chance to scale linearly with the number of executors.

4. EXPERIMENTS OF LINKEDIN JOB RECOMMENDATION

In this section, we describe our experiments for the LinkedIn job recommendation problem. Section 4.1 starts with the offline experimental results to compare several variations of GLMix models with a couple of baselines for prediction accuracy, and also experiments on scalability of GLMix in Section 4.1. In Section 4.2 we share implementation details of how we launched GLMix in LinkedIn's production system. We discuss the online A/B test results in Section 4.3.

4.1 Offline Experiments

4.1.1 Setup

All the methods used in the offline experiments are implemented using Apache Spark. In particular, the baseline

methods such as ridge regression, ℓ_2-regularized logistic regression, matrix factorization with alternating least squares are from Spark's open source machine learning library MLlib. The experiments were conducted on a cluster consisting of 135 nodes managed by Apache YARN [3]. Each node has 24 Intel Xeon(R) CPU E5-2640 processors with 6 cores at 2.50GHz each, and every node has 250GB memory. In the following experiments, Spark is running on top of YARN as a client, and the memory for each Spark executor is configured to 10GB, where each executor corresponds to a container (e.g., a core of the node) in YARN.

4.1.2 Our Data

Our offline experiments are conducted using a sample of the four months' worth of data (07/2015 - 11/2015) of member's interactions on the "Jobs You May Be Interested In" module on the LinkedIn jobs homepage. The data include 500M samples with 5M members and 3M jobs. We split the data by time into training (90%), validation (5%) and test (5%).

Impressions that resulted in application clicks provide the positive responses, and impressions that did not result in application clicks provide the negative responses. Note that the detail job views are not considered in this experiment, since that is not the metric to optimize for. There are about $2K$ member features for $\boldsymbol{q}_m$, mostly extracted from public member profiles, e.g. industry, job function, education history, skills, and so forth. There are also around $2K$ job features for $\boldsymbol{s}_j$, e.g. the job title, key words in the description, required skills and qualifications etc. We consider two types of interactions among the member and job features: a) The cosine-similarity of the member and job features in the same domain, e.g. keywords in the summary of the member/job profile, or the skills of the member / job requirement. There are around 100 such cosine-similarity features, and we denote the feature vector as $\boldsymbol{f}_c$. b) The simple outer-product of the member and job features. After applying standard feature selection techniques we obtained 20K outer-product features (denoted as $\boldsymbol{f}_o$).

4.1.3 Experiments of Performance

We consider the following models in our offline experiments (where g denotes the link function $g(E[y_{mjt}])$):

- LR-Cosine: The baseline logistic regression model that only includes the cosine-similarity features $g = \boldsymbol{f}_c'\boldsymbol{b}$.
- LR-Full: The logistic regression model that includes all global features, $\boldsymbol{x}_{mjt} = \{\boldsymbol{q}_m, \boldsymbol{s}_j, \boldsymbol{f}_c, \boldsymbol{f}_o\}$, $g = \boldsymbol{x}_{mjt}'\boldsymbol{b}$.
- MF: A baseline matrix factorization model, $g = \boldsymbol{x}_{mjt}'\boldsymbol{b} + \boldsymbol{u}_m'\boldsymbol{v}_j$.
- GLMix-Member: The GLMix model which includes only the member-level random effects: $g = \boldsymbol{x}_{mjt}'\boldsymbol{b} + \boldsymbol{s}_j'\boldsymbol{\alpha}_m$.
- GLMix-Job: The GLMix model which includes only the job-level random effects: $g = \boldsymbol{x}_{mjt}'\boldsymbol{b} + \boldsymbol{q}_m'\boldsymbol{\beta}_j$.
- GLMix-Full: The full GLMix model as described in Equation (1).
- GLMix-Full+MF: An extension of GLMix which adds an additional matrix factorization component, as in Equation (4).

For each model listed above, we tried multiple values of hyper-parameters for the priors (i.e. λ_b, λ_α and λ_β), picked the ones with the best AUC performance on the validation data, and then report their AUC on the test data in Table 1. It is not surprising that adding the member and job random effects to the model dramatically improves the AUC performance (0.723 for LR-Full vs 0.799 for GLMix-Full). There are also two other interesting observations: a) GLMix-Full significantly outperforms MF, which indicates that for this data, using features to interact with member/job IDs is more effective than the interactions among member IDs and job IDs. We believe this is due to the sparse structure of the data and also because the features are very useful. b) Adding matrix factorization to GLMix-Full provides very little improvement in terms of AUC (0.799 to 0.801). This indicates that GLMix already captures most signals of the data, and adding matrix factorization for additional signals in the residual is not effective.

4.1.4 Experiments of Scalability

In this section, we study a) how GLMix scales up with the increase of both data size and the number of Spark executors used, and b) how GLMix speeds up with the increase of number of Spark executors, while fixing the data size. In this experiment, the size of data is controlled by the number of days used to collect the data, e.g., 60 days of the data is about half of the size of 120 days of the data. The experiment results can be found in Figure 4. In general, we find that our proposed algorithm has good scalability properties, and achieves roughly linear speed-up on the Spark cluster, provided that the amount of data processed per executor does not become so small that system overheads start to dominate.

In Figure 4, we also provide the details on the change of the data complexity C, as defined in Equation (11), as the data size increases. We note that the scalability of our implementation is quite robust to the increasing complexity of the data set. If we had used the naive implementation where the network I/O complexity is proportional to the data complexity, it would have been much worse.

4.2 GLMix in Production System

LinkedIn operates one of the largest job recommendation systems in the world, which serves relevant jobs to members who visit the Job Homepage every day. For each member visit, it is required that we rank millions of jobs using relevance models and serve the top ranked ones in less than a second. Apart from model training challenges, to productionize a GLMix model with a huge number of features and random effects, the scoring and ranking become another critical challenge to overcome.

Our approach to mitigate such a challenge is to have a two-stage ranking strategy. In the first stage of the ranking, we convert it to a search-like problem by leveraging the open-source library Apache Lucene [4]. In this stage, an inverted job index is built, where keys are the job features and values are the corresponding job IDs. We run the baseline model LR-Cosine in this stage, which works efficiently with

[3]https://hadoop.apache.org

[4]https://lucene.apache.org/

Model	LR-Cosine	LR-Full	MF	GLMix-Member	GLMix-Job	GLMix-Full	GLMix-Full+MF
AUC	0.664	0.723	0.772	0.780	0.758	0.799	**0.801**

Table 1: Offline model performance for LinkedIn Job Recommendation data.

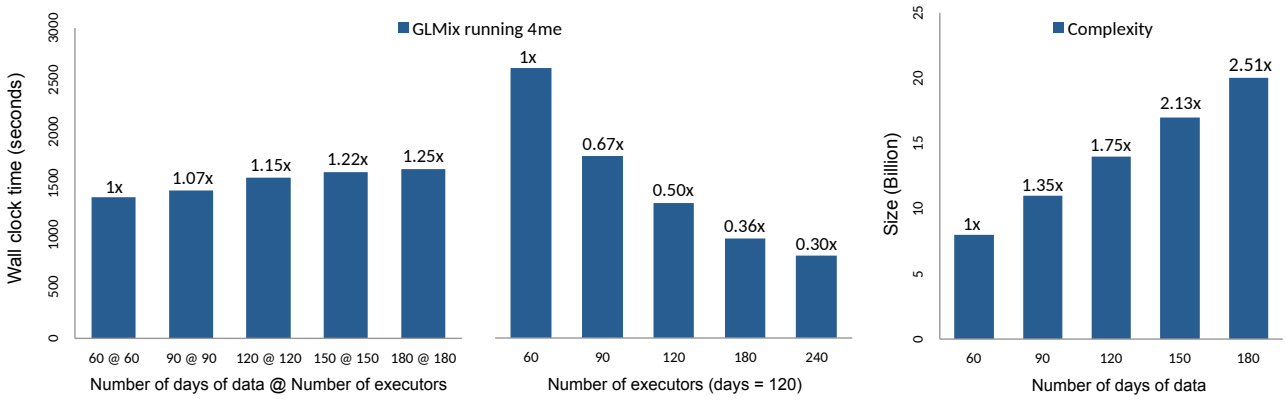

Figure 4: Left and Mid panel: Scale up and speed up results on the Spark cluster. The scale-up experiment is shown on the left panel, where we increase the number of days of data and the amount of Spark executors used at the same time, and the speed-up experiment is shown on the mid panel, where we fix the number of days of data and increase the number of Spark executors used. Right panel: Data complexity as a function of number of days of data.

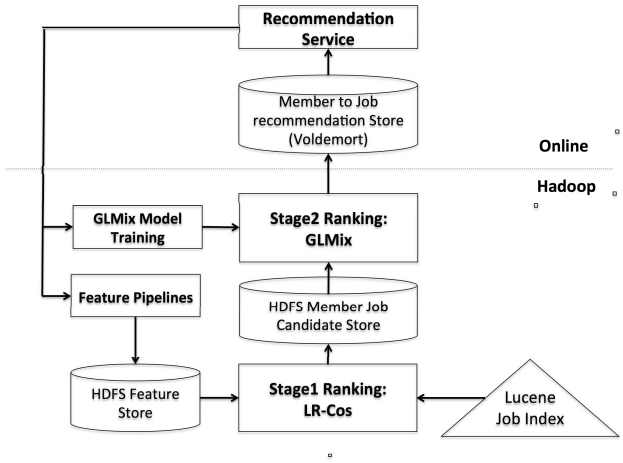

Figure 5: High-level architecture of GLMix in job recommendation production system

Lucene (since all the features are cosine-similarity based), and select the top K results (e.g. K=1000) for the second stage ranking. In the second stage, we re-score the top K results using the full GLMix model and show the top-ranked jobs to the member.

Although such a two-stage strategy can be implemented online, at this time we choose an offline approach as in Figure 5, for more flexibility in experimentation and because of the fact that jobs are usually more evergreen than news articles. Through Hadoop Map-Reduce, we pre-generate the top job recommendations for all members, and push them to an online distributed key-value store using Voldemort [5], where the keys are member IDs and the values are the recommended job IDs. When a member visits the Jobs Homepage, we simply retrieve the recommended jobs from the store, do some post-processing (e.g. freshness control), and show them to the members. This offline-based approach to serve online recommendations has proven to be very valuable in facilitating our experimentation and exploration of new features. Through A/B tests, whatever member or job features that become available on our Hadoop system, we can quickly test them before fully implementing the features in the online retrieval systems, which usually takes much more time.

Note that the two-stage scoring approach is very important even if we move the computation offline: Although it is possible to use Hadoop Map-Reduce to score millions of jobs for each of the 400M LinkedIn members using GLMix, the amount of Hadoop resources it consumes is too high to be cost-efficient.

We also note that it is required to retrain the GLMix model frequently (e.g. every hour or every day). Our offline experiments show that the AUC of GLMix-Full drops from 0.799 to 0.765 if we do not update the model for 20 days.

4.3 Online Experimental Results

We deployed the GLMix-Full model to the LinkedIn job recommendation system as described in Section 4.2 and ran an A/B test to compare with the existing baseline model in the production system, which uses the LR-Cosine model. Each model served a randomly-selected 2% of members, and we report the online experimental results for a week's data in November 2015.

In Figure 6 we show the lift of the job application clicks and job detail views comparing the GLMix model against the baseline LR-Cosine model, for each day of the week. We observe that every day there is a 20%-40% lift of the job application clicks, and interestingly also a consistent 10%-20% lift of job detail views. This shows that a more relevant job recommendation model, which is optimized for job applications, can also dramatically increase the number of times

[5]http://www.project-voldemort.com/voldemort/

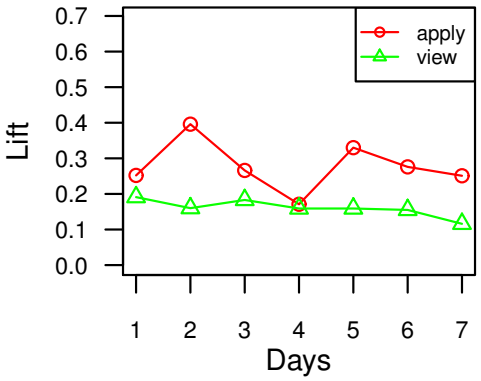

Figure 6: The lift of job application clicks and job detail views comparing GLMix-Full and the baseline model LR-Cosine, for each day of the week.

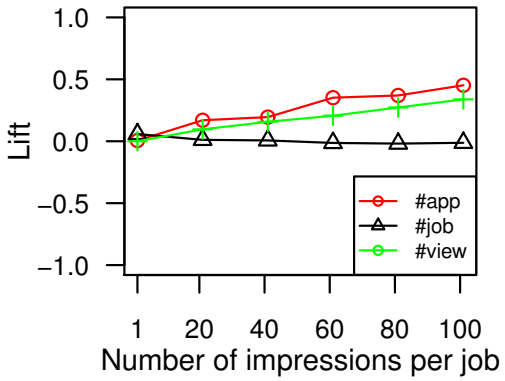

Figure 7: The #app and #view curves are the lift of job application clicks and job detail views comparing GLMix-Full against the baseline model LR-Cosine, for different segments of jobs with different numbers of impressions. The curve of #job shows the lift of the number of jobs per segment comparing GLMix-Full and LR-Cosine.

people click on the job thumbnails to view a given job posting in detail. To further understand the experimental results and investigate the effectiveness of the job random effects in the GLMix model, we segment our online A/B test data by the number of impressions per job, ranging from 1 to 100, and show the lift of job application clicks and detail views comparing GLMix with LR-Cosine in Figure 7. It is interesting to observe that as a job gets more impressions (which is when the per-job random effects become more effective), GLMix generates more lifts compared to the baseline. In the meantime, note that the distribution of the number of impressions per job changes very little, indicating that the GLMix model does not simply serve more popular jobs to members to obtain such a lift in application clicks (in that case the total number of jobs that receive one impression would have dramatically shrunk), instead it tries to serve more relevant jobs to members while keeping the distribution of impressions per job unchanged.

Similarly, we study the effectiveness of the member random effects by segmenting the data into 8 buckets based on the number of visits per member in Figure 8. We observe similar rising patterns for the lift of job application clicks as the number of visits per member increases. There is some slight up patterns of the job detail view curve too but not very significant. The distribution of total number of visits per member remains unchanged since this is a randomized A/B test experiment.

5. EXPERIMENTS ON PUBLIC DATA SETS

In this section, we report the performance of GLMix on two public data sets to demonstrate its modeling capability.

KDD Cup 2012 Track 2: We obtain this data set from https://www.kddcup2012.org/c/kddcup2012-track2. Recall that N_r and P_r denote the random effect size and feature dimension of random effect of type $r \in \mathcal{R}$ respectively, and complexity $= N_r * P_r$, which represents the size of the design matrix resulting from the outer product between the random effect IDs and the features. Basic statistics of the data set is summarized in Table 2.

All the features used through the experiments are raw features from the data set. No feature engineering has been conducted for two reasons: (i) to compare GLMix against models with less granularity but trained on well-engineered features as reported in [22] and [18]; and (ii) to make it more clear whether the lift results from the well-engineered features, or from GLMix itself. Nevertheless, we believe the best performance could come from the combination of well-engineered features and fine-grained modeling. We leave such study as future work. The features provided in the data set include the position of the ad, the user's age and gender, the tokens of the query, the ad keyword, and the title. The details of the features can be found on the website above.

The accuracy results of different models, measured by AUC are shown in Table 3. The GLMix model with all random effects (G-All) significantly outperforms LR and other variants of GLMix with fewer random effects (*e.g.*, G-Query, which only contains the per-query random effect coefficients). This shows that adding random effects can usually improve model accuracy. We also compare GLMix to other existing methods. In particular, G-All outperforms both RankMF [22] and FM [18] based on their published results. When comparing to the best models in the KDD Cup competition, G-All is ranked between top 5 to top 10.

One thing to note is that all of the models that outperform G-All are ensemble methods, and G-All is the best among the single models.

Type	N_r	$N_r P_r$
user	22,023,547	286,306,111
advertiser	14,847	14,450,644,488
ad	641,707	624,575,989,928
query	24,122,076	4.5192951e+12
title	3,735,796	115,809,676
description	2,934,101	90,957,131
keyword	1,188,089	36,830,759

Table 2: Summary of KDD Cup 2012 Track 2 Data

Yahoo! Music: We obtain this data set from Yahoo! Web-

Model	LR	G-Advertiser	G-Ad	G-Query	G-All	RankMF	FM	Top 5	Top 10	Top 50
Public AUC	0.7497	0.7696	0.7716	0.7698	**0.7982**	0.7968	0.7901	0.7987	0.7925	0.7563
Private AUC	0.7503	0.7645	0.7703	0.7729	**0.7940**	-	0.7932	**0.8017**	0.7938	0.7565

Table 3: Model performance on KDD Cup 2012 Track 2. LR stands for logistic regression, G-name stands for GLMix model with random effect of type name included, G-All stands for GLMix model with all random effects included. RankMF is the best single model reported in [22], FM is the factorization machine reported in [18], Top K means the top k model performance reported on the public/private leaderboard.

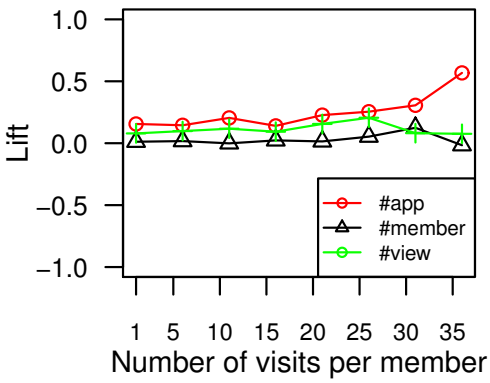

Figure 8: The #app and #view curves are the lift of job application clicks and job detail views comparing GLMix-Full and the baseline model LR-Cosine, for different segments of members with different number of visits. The curve of #member shows the lift of the number of members per segment comparing GLMix-Full and LR-Cosine.

scope (http://webscope.sandbox.yahoo.com): R2 - Yahoo! Music User Ratings of Songs with Artist, Album, and Genre Meta Information, v.1.0. Basic statistics of the data set can be found in Table 4. Features used in this experiment are listed as follows:

- Song attributes: Artist ID, album ID, and genre ID. The total number of such features used is 30,065.
- User latent features and song latent features: Since there is no user feature provided by the data set, we use matrix factorization with rank 30 to generate latent factors for users and songs, and treat them as features.

The root mean squared errors (RMSEs) of these models are reported in Table 5, where the linear model without random effects (RR) does not perform well and MF performs better than GLMix with either only user random effects or only song random effects. However, when both random effects are included, GLMix-Full outperforms MF.

Type	N_r	$N_r P_r$
User	1,623,179	48,833,340,215
Song	136,736	4,113,702,560

Table 4: Summary of Yahoo! Music Data

6. RELATED WORK

Generalized linear mixed models (GLMix) have been thoroughly studied in Statistics [15] and widely applied in many applications. For recommender systems, the full GLMix model that is described in this paper was originally introduced in [3]. There are also many other works with similar or related ideas. For example, [2] used item-level random effects to model the per-ad-campaign interactions with user features,[25] considered recommender system problems such as the Netflix challenge from the perspective of multi-task learning, and described a hierarchical per-item random effect model restricted to the Gaussian case. Our model differs from previous approaches mostly in that our method scales GLMix to industry-scale data sets with an efficient parallel block-wise coordinate descent algorithm.

Traditionally the model fitting approaches for GLMix have relied on Gibbs sampling [3, 10] or trying to integrate the log likelihood over the random effect and obtaining the marginal maximum likelihood estimates of the hyper-parameters [4, 8]. However, for non-Gaussian scenarios, the integral is often not closed form; thus approximations such as adaptive Gauss-Hermite quadrature need to be applied. Note that such approximations only work for scalar random effects, not the multi-dimensional cases where we need to model per-user/item interactions with features. Scaling to large data sets is a problem that none of the above approaches have addressed.

Block coordinate descent algorithms (BCD) have been a natural choice for generalized linear models with certain structures. For example, [17] used BCD for group lasso in logistic regression, and [5] applied BCD for multi-class classification. [19] tried to leverage repeated patterns in the design matrix to create blocks to boost the efficiency of the learning algorithms. [20] used BCD for a mixed effects model that contains only the scalar random effects for Gaussian responses. Therefore, we consider BCD as a natural choice for the GLMix model (more general than the ones with only scalar random effects) in this paper as well.

The parallelization of the BCD algorithms is also a popular area in machine learning research, aiming to speed up parameter learning for loss functions that are separable or partially separable [7]. The ideas are generally to update blocks of parameter estimates in parallel, and the blocks are formed either by natural structure or random sets of parameters. In this paper, since the user/item-level random effects have a natural block structure, the parallelization becomes trivial.

The use of a parameter server [23, 13] is an active research area, which can be used to fit models with a large number of features. In this work we choose to focus on the BSP paradigm and work with Apache Spark because it fits naturally into LinkedIn's distributed computing ecosystem. However, investigating GLMix with parameter server is a promising future direction for research.

7. CONCLUSION

Generalized linear models (GLMs) have been successfully applied to a wide range of applications such as response

Model	MF (rank = 30)	RR	GLMix-User	GLMix-Song	GLMix-Full
RMSE	1.07	1.2734	1.0716	1.0911	**1.0510**

Table 5: Model performance on Yahoo! Music data, MF is trained with Spark MLlib's ALS, RR is a linear model without random effects, GLMix-User and GLMix-Song only models user and song as random effect, respectively, while GLMix-Full models both user and song as random effects.

prediction. However, when the data is abundant at the user/item level, more fine-grained modeling strategies are desired since capturing signals from the user/item's idiosyncrasies in addition to the global pattern could lead to a potentially considerable performance gain. Although it is tempting to apply GLMix on large scale data set, it is also challenging because scalability becomes a problem as the number of parameters of GLMix grows with the size of the data.

In this paper we propose to use GLMix to extend GLM in a way such that the user/item-level idiosyncrasies are captured by the random effects, as a complement to the fixed effect which is responsible for modeling the global pattern. To address the scalability challenge when applying GLMix to the large-scale data, we propose a parallel block-wise coordinate descent algorithm under the bulk synchronous parallel paradigm, and implemented it using Apache Spark. Through extensive experiments we demonstrated the effectiveness, efficiency and scalability of the proposed approach. We also shared our experience of deploying GLMix in the LinkedIn job recommendation system, and showed significant lifts of job applications over the production baseline.

8. REFERENCES

[1] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford. A reliable effective terascale linear learning system. *J. Mach. Learn. Res.*, 2014.

[2] D. Agarwal, B. Long, J. Traupman, D. Xin, and L. Zhang. Laser: A scalable response prediction platform for online advertising. In *WSDM*, 2014.

[3] A. Ansari, S. Essegaier, and R. Kohli. Internet recommendation systems. *Journal of Marketing research*, 2000.

[4] D. Bates, M. Mächler, B. Bolker, and S. Walker. Fitting linear mixed-effects models using lme4. *arXiv preprint arXiv:1406.5823*, 2014.

[5] M. Blondel, K. Seki, and K. Uehara. Block coordinate descent algorithms for large-scale sparse multiclass classification. *Machine learning*, 2013.

[6] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 2011.

[7] J. K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. Parallel coordinate descent for l1-regularized loss minimization. *arXiv preprint arXiv:1105.5379*, 2011.

[8] N. E. Breslow and D. G. Clayton. Approximate inference in generalized linear mixed models. *Journal of the American Statistical association*, 1993.

[9] R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 2008.

[10] P. D. Hoff. Bilinear mixed-effects models for dyadic data. *Journal of the American Statistical association*, 2005.

[11] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 2009.

[12] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan. Mlbase: A distributed machine-learning system. In *CIDR*.

[13] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *OSDI*, 2014.

[14] J. Liu, J. Chen, and J. Ye. Large-scale sparse logistic regression. In *KDD*, 2009.

[15] C. E. McCulloch and J. M. Neuhaus. *Generalized linear mixed models*. 2003.

[16] R. Mcdonald, M. Mohri, N. Silberman, D. Walker, and G. S. Mann. Efficient large-scale distributed training of conditional maximum entropy models. In *NIPS*. 2009.

[17] L. Meier, S. Van De Geer, and P. Bühlmann. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B*, 2008.

[18] S. Rendle. Social network and click-through prediction with factorization machines. In *KDD Cup 2012 Workshop*.

[19] S. Rendle. Scaling factorization machines to relational data. In *PVLDB*, 2013.

[20] J. Schelldorfer, P. Bühlmann, G. DE, and S. VAN. Estimation for high-dimensional linear mixed-effects models using l1-penalization. *Scandinavian Journal of Statistics*, 2011.

[21] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 1990.

[22] K.-W. Wu, C.-S. Ferng, and et al. A two-stage ensemble of diverse models for advertisement ranking in kdd cup 2012. In *KDD Cup 2012 Workshop*.

[23] E. P. Xing, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, A. Kumar, and Y. Yu. Petuum: A new platform for distributed machine learning on big data. In *KDD*, 2015.

[24] H.-F. Yu, C.-J. Hsieh, S. Si, and I. Dhillon. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *ICDM*, 2012.

[25] K. Yu, J. Lafferty, S. Zhu, and Y. Gong. Large-scale collaborative prediction using a nonparametric random effects model. In *ICML*, 2009.

[26] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. *HotCloud*, 2010.

[27] Y. Zhang, M. J. Wainwright, and J. C. Duchi. Communication-efficient algorithms for statistical optimization. In *NIPS*. 2012.