Coupled Group Lasso for Web-Scale CTR Prediction in Display Advertising

Ling Yan YLING0718@SJTU.EDU.CN
Shanghai Key Laboratory of Scalable Computing and Systems, Department of Computer Science and Engineering, Shanghai Jiao Tong University, China

Wu-Jun Li LIWUJUN@NJU.EDU.CN
National Key Laboratory for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, China

Gui-Rong Xue GRXUE@ALIBABA-INC.COM
Alibaba Group, China

Dingyi Han DINGYI.HAN@ALIBABA-INC.COM
Alibaba Group, China

Abstract

In display advertising, click through rate (CTR) prediction is the problem of estimating the probability that an advertisement (ad) is clicked when displayed to a user in a specific context. Due to its easy implementation and promising performance, logistic regression (LR) model has been widely used for CTR prediction, especially in industrial systems. However, it is not easy for LR to capture the nonlinear information, such as the conjunction information, from user features and ad features. In this paper, we propose a novel model, called coupled group lasso (CGL), for CTR prediction in display advertising. CGL can seamlessly integrate the conjunction information from user features and ad features for modeling. Furthermore, CGL can automatically eliminate useless features for both users and ads, which may facilitate fast online prediction. Scalability of CGL is ensured through feature hashing and distributed implementation. Experimental results on real-world data sets show that our CGL model can achieve state-of-the-art performance on web-scale CTR prediction tasks.

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

1. Introduction

Recently, online advertising has become the most popular and effective approach to do brand promotion and product marketing. It is a multi-billion business on the web and accounts for the majority of the income for the major internet companies, such as Google, Yahoo and Alibaba. Display advertising is a big part of online advertising where advertisers pay publishers for placing graphical advertisements (ads) on publishers' web pages (Chapelle et al., 2013). The publishers allocate some positions on their web pages and sell them to different advertisers. Users visit the web pages and can view the published ads. There are some other roles, such as ad agencies and publisher networks, to compose the complex advertising system (Muthukrishnan, 2009). But that is not the focus of this paper. So we will just focus on the scenarios with a user-advertiser-publisher tripartite business, in which three parties have separate goals that can be reduced to a unified task in the end. The advertisers pay more attention on the desired user actions, such as clicks on the ads, subscriptions to the mailing list, or purchases of products. Different advertisers target different kinds of users. For example, a basketball company will be interested in users who bought many sports equipments recently, and a hotel would prefer to display its ads to people who travel frequently. There are different payment options for advertisers, such as cost-per-click (CPC), cost-per-mill (CPM), and cost-per-conversion (CPA) (Mahdian & Tomak, 2007). For the publisher part, their goal is to maximize the revenue from the advertisers and attract more users to their web pages. So they had better precisely display suitable ads to a specific user, and avoid affecting user

experience of the web pages. From the user part, they want to find useful information from the web pages and find ads that they are really interested in.

To satisfy the desire of all three parties, an accurate targeting of advertising system is of great importance, in which the click through rate (CTR) prediction of a user to a specific ad plays the key role (Chapelle et al., 2013). CTR prediction is the problem of estimating the probability that the display of an ad to a specific user will lead to a click. This challenging problem is at the heart of display advertising and has to deal with several hard issues, such as very large scale data sets, frequently updated users and ads, and the inherent obscure connection between user profiles (features) and ad features.

Recently, many models have been proposed for CTR prediction in display advertising. Some models train standard classifiers, such as logistic regression (LR) (Neter et al., 1996) or generalized linear models, on simple concatenation of user and ad features (Richardson et al., 2007; Graepel et al., 2010). Some other models use prior knowledge like the inherent hierarchical information for statistical smoothing in log-linear models (Agarwal et al., 2010) or LR models (Kuang-chih et al., 2012). In (Menon et al., 2011), a matrix factorization method is proposed, but it does not make use of user features. In (Stern et al., 2009), a probabilistic model is proposed to use user and item meta data together with collaborative filtering information, in which user and item feature vectors are mapped into lower-dimensional space and inner product is used to measure similarity. However, it does not have the effect of automatic feature selection from user and item meta features. In addition, inference of the model is too complicated to be used in a large scale scenario. In (Chapelle et al., 2013), a highly scalable framework based on LR is proposed, and terabytes of data from real applications are used for evaluation. Due to its easy implementation and state-of-the-art performance, LR model has become the most popular one for CTR prediction, especially in industrial systems (Chapelle et al., 2013). However, LR is a linear model, in which the features contribute to the final prediction independently. Hence, LR can not capture the nonlinear information, such as the *conjunction* (cartesian product) information, between user features and ad features. In real applications, the conjunction information is very important for CTR prediction. For example, people who have *high buying power* may have more interest in *luxury product* than those with *low buying power*, and *college students* may be more likely to buy *machine learning books* than *high-school students*. Better performance can be expected by exploiting the user-ad two-parts hybrid features through feature conjunction.

In this paper, we propose a novel model, called coupled group lasso (CGL) for CTR prediction in display advertising. The main contributions are outlined as follows:

- CGL can seamlessly integrate the conjunction information from user features and ad features for modeling, which makes it better capture the underlying connection between users and ads than LR.
- CGL can automatically eliminate useless features for both users and ads, which may facilitate fast online prediction.
- CGL is scalable by exploiting feature hashing and distributed implementation.

2. Background

In this section, we introduce the background of our model, including the description of CTR prediction task, LR model, and group lasso (Yuan & Lin, 2006; Meier et al., 2008).

2.1. Notation and Task

We use boldface lowercase letters, such as $\mathbf{v}$, to denote column vectors and v_i to denote the ith element of $\mathbf{v}$. Boldface uppercase letters, such as $\mathbf{M}$, are used to denote matrices, with the ith row and the jth column of $\mathbf{M}$ denoted by $\mathbf{M}_{i*}$ and $\mathbf{M}_{*j}$, respectively. M_{ij} is the element at the ith row and jth column of $\mathbf{M}$. $\mathbf{M}^T$ is the transpose of $\mathbf{M}$ and $\mathbf{v}^T$ is the transpose of $\mathbf{v}$.

While some display advertising systems have access to only some id information for users or ads, in this paper we focus on the scenarios where we can collect both user and ad features. Actually, the publishers can often collect user actions on the web pages, such as click on an ad, buy a product or type in some query keywords. They can analyze these history behaviors and then construct user profiles (features). On the other hand, when advertisers submit some ads to the publishers, they often choose some description words, the groups of people to display the ads, or some other useful features.

We refer to a display of an ad to a particular user in a particular page view as an ad *impression*. Each impression is a case that a user meets an ad in a specific context, such as daytime, weekdays, and publishing position. Hence, each impression contains information of three aspects: the user, the ad, and the context. We use $\mathbf{x}_u$ of length l to denote the feature vector of user u, $\mathbf{x}_a$ of length s to denote the feature vector of ad a. The context information together with some advertiser id or ad id information are composed into a feature vector $\mathbf{x}_o$ of length d. $\mathbf{x}$ is used to denote the feature vector of an expression, with $\mathbf{x}^T = (\mathbf{x}_u^T, \mathbf{x}_a^T, \mathbf{x}_o^T)$. Hence, if we use z to denote the length of vector $\mathbf{x}$, we have $z = l + s + d$. The result of an impression is *click* or

non-click, which makes an instance in the data set.

Given a training set $\{(\mathbf{x}^{(i)}, y^{(i)}) \mid i = 1, ..., N\}$, in which $\mathbf{x}^T = (\mathbf{x}_u^T, \mathbf{x}_a^T, \mathbf{x}_o^T), y \in \{0, 1\}$ with $y = 1$ denoting *click* and $y = 0$ denoting *non-click* in an impression, the CTR prediction problem is to learn a function $h(\mathbf{x}) = h(\mathbf{x}_u, \mathbf{x}_a, \mathbf{x}_o)$ which can be used to predict the probability of user u to click on ad a in a specific context o.

2.2. Logistic Regression

The likelihood of LR is defined as $h_1(\mathbf{x}) = Pr(y = 1|\mathbf{x}, \mathbf{w}) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x})}$, where $\mathbf{w}$ is the parameter (weight vector) to learn. Please note that the bias term of LR has been integrated into $\mathbf{w}$ by adding an extra feature with constant value 1 to the feature vector. Given a training set $\{(\mathbf{x}^{(i)}, y^{(i)}) \mid i = 1, ..., N\}$, the weight vector $\mathbf{w}$ is found by minimizing the following regularized loss function:

$$\min_{\mathbf{w}} \lambda\Omega_1(\mathbf{w}) + \sum_{i=1}^{N} \xi_1(\mathbf{w}; \mathbf{x}^{(i)}, y^{(i)}), \tag{1}$$

$$\xi_1(\mathbf{w}; \mathbf{x}^{(i)}, y^{(i)}) = -\log([h_1(\mathbf{x}^{(i)})]^{y^{(i)}}[1 - h_1(\mathbf{x}^{(i)})]^{1-y^{(i)}}),$$

where $\Omega_1(\mathbf{w})$ is the regularization term.

In real applications, we can use the following L_2-norm for regularization (Golub et al., 1999): $\Omega_1(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||_2^2 = \frac{\mathbf{w}^T\mathbf{w}}{2}$. The resulting model is the standard LR model. We can also use the following L_1-norm for regularization: $\Omega_1(\mathbf{w}) = ||\mathbf{w}||_1 = \sum_{i=1}^{z} |w_i|$, where z is the length of vector $\mathbf{w}$. The resulting model will be *Lasso* which can be used for feature selection or elimination (Tibshirani, 1996).

The optimization function in (1) is easy to implement with promising performance, which makes LR very popular in industry. Please note that in the following content, LR refers to the LR model with L_2-norm regularization, and the LR with L_1-norm will be called *Lasso* as in many literatures (Tibshirani, 1996)

2.3. Group Lasso

The group lasso is a technique to do variable selection on (predefined) groups of variables (Yuan & Lin, 2006; Meier et al., 2008). For a parameter vector $\boldsymbol{\beta} \in \mathbb{R}^z$, the regularization term in group lasso is defined as follows:

$$\sum_{g=1}^{G} ||\boldsymbol{\beta}_{\mathcal{I}_g}||_2, \tag{2}$$

where $\mathcal{I}_g$ is the index set belonging to the predefined gth group of variables, $g = 1, 2, \cdots, G$. The group lasso can be used together with linear regression (Yuan & Lin, 2006) or logistic regression (Meier et al., 2008) as a penalty. It is attractive for its property of doing variable selection at the group level, where all the variables in some groups will be zero after learning.

3. Coupled Group Lasso

Although LR has been widely used for CTR prediction, it can not capture the *conjunction* information between user features and ad features. One possible solution is to manually construct the conjunction features from the original input features as the input of LR. However, as stated in (Chapelle et al., 2013), manual feature conjunction will result in quadratic number of new features, which makes it extraordinarily difficult to learn the parameters. Hence, the modeling ability of LR is too weak to capture the complex relationship in the data.

In this section, we introduce our coupled group lasso (CGL) model, which can easily model the conjunction information between users and ads to achieve better performance than LR.

3.1. Model

The likelihood of CGL is formulated as follows:

$$\begin{aligned} h(\mathbf{x}) &= Pr(y = 1|\mathbf{x}, \mathbf{W}, \mathbf{V}, \mathbf{b}) \\ &= \sigma\left((\mathbf{x}_u^T\mathbf{W})(\mathbf{x}_a^T\mathbf{V})^T + \mathbf{b}^T\mathbf{x}_o\right), \end{aligned} \tag{3}$$

where $\mathbf{W}$ is a matrix of size $l \times k$, $\mathbf{V}$ is a matrix of size $s \times k$, $\mathbf{b}$ is a vector of length d, $\sigma(x)$ is the sigmoid function with $\sigma(x) = \frac{1}{1+\exp(-x)}$. Here, $\mathbf{W}$, $\mathbf{V}$ and $\mathbf{b}$ are parameters to learn, k is a hyper-parameter.

Furthermore, we put regularization on the negative log-likelihood to get the following optimization problem of CGL:

$$\min_{\mathbf{W},\mathbf{V},\mathbf{b}} \sum_{i=1}^{N} \xi\left(\mathbf{W}, \mathbf{V}, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)}\right) + \lambda\Omega(\mathbf{W}, \mathbf{V}), \tag{4}$$

with

$$\begin{aligned} &\xi(\mathbf{W}, \mathbf{V}, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)}) = \\ &\qquad -\log([h(\mathbf{x}^{(i)})]^{y^{(i)}}[1 - h(\mathbf{x}^{(i)})]^{1-y^{(i)}}), \end{aligned} \tag{5}$$

$$\Omega(\mathbf{W}, \mathbf{V}) = ||\mathbf{W}||_{2,1} + ||\mathbf{V}||_{2,1}. \tag{6}$$

Here, $||\mathbf{W}||_{2,1} = \sum_{i=1}^{l} \sqrt{\sum_{j=1}^{k} W_{ij}^2} = \sum_{i=1}^{l} ||\mathbf{W}_{i*}||_2$ is the $L_{2,1}$-norm of the matrix $\mathbf{W}$. Similarly, $||\mathbf{V}||_{2,1}$ is the $L_{2,1}$-norm of the matrix $\mathbf{V}$. From (2), it is easy to find that the $L_{2,1}$-norm is actually a group lasso regularization with each row being a group. Please note that we do not put regularization on $\mathbf{b}$ because from experiments we find that this regularization does not affect the performance.

We can find that there are two group lassos in (4), one for user features and the other for ad features. Furthermore, the two group lassos are coupled together to determine the CTR of an impression. Hence, our model is called coupled group lasso (CGL).

Because $(\mathbf{x}_u^T\mathbf{W})(\mathbf{x}_a^T\mathbf{V})^T = \mathbf{x}_u^T\mathbf{W}\mathbf{V}^T\mathbf{x}_a = \mathbf{x}_u^T(\mathbf{W}\mathbf{V}^T)\mathbf{x}_a$, it is very interesting to see that the term $(\mathbf{x}_u^T\mathbf{W})(\mathbf{x}_a^T\mathbf{V})^T$ in (3) can effectively model the conjunction information between user features and ad features. The number of parameters in $\mathbf{W}$ and $\mathbf{V}$ is only $(l+s)k$, where k is typically a small number (less than 50 in our experiments). On the contrary, if we choose to manually construct the conjunction features for LR (Chapelle et al., 2013), the number of manual conjunction features is $l \times s$, with both l and s being typically tens of thousands in CTR prediction. Hence, the number of parameters of CGL is much less than that of LR with manual conjunction features, which makes CGL more scalable than LR to model conjunction features. Furthermore, the less number of parameters will result in a lower probability to be overfitting for a model.

Another nice property of CGL comes from the regularization term of group lasso. It is easy to find that some rows of $\mathbf{W}$ and $\mathbf{V}$ will be all-zero after learning. Because each row corresponds to a feature of users or ads, we can eliminate the features corresponding to all-zero parameter values. Hence, when we use the learned model for online prediction, it's not necessary to collect those eliminated features for users and ads. This can not only save memory, but also speed up the online prediction procedure.

3.2. Learning

The goal of our learning algorithm is to find the optimal values $\mathbf{b}^* \in \mathbb{R}^d, \mathbf{W}^* \in \mathbb{R}^{l\times k}, \mathbf{V}^* \in \mathbb{R}^{s\times k}$ that minimize the objective function in (4). The coupled part of $(\mathbf{x}_u^T\mathbf{W})(\mathbf{x}_a^T\mathbf{V})^T$ makes the objective function non-convex. We adopt an alternating learning method to learn the parameters. Each time we optimize one parameter with other parameters fixed. Several iterations will be repeated for this procedure until some termination condition is satisfied. More specifically, we first fix the ad parameter $\mathbf{V}$ and use limited-memory BFGS (L-BFGS) (Malouf, 2002; Andrew & Gao, 2007) to optimize the objective function with respect to (w.r.t) $\mathbf{W}$ and $\mathbf{b}$ until convergence. Then we fix the user parameter $\mathbf{W}$, and optimize w.r.t. $\mathbf{V}$ and $\mathbf{b}$ until convergence. Obviously, the objective function is convex in any one of its parameter matrices $\mathbf{W}$ or $\mathbf{V}$.

L-BFGS algorithm is in the family of quasi-Newton methods (Broyden, 1970). L-BFGS stores only a few gradient vectors to approximate the Hessian matrix. Hence, it's more suitable for optimization problems with a large number of variables (Nocedal, 1980; Byrd et al., 1994). To use the L-BFGS algorithm, we only need to compute the gradient of the parameters.

For ease of presentation, we use $\xi(\mathbf{x}, y)$ to denote $\xi(\mathbf{W}, \mathbf{V}, \mathbf{b}; \mathbf{x}, y)$ in (5) by omitting the parameters. For each instance $(\mathbf{x}, y)$, the gradient contributed by this instance can be derived as follows:

$$\frac{\partial \xi(\mathbf{x}, y)}{\partial b_i} = (h(\mathbf{x}) - y)x_{oi}, \tag{7}$$

$$\frac{\partial \xi(\mathbf{x}, y)}{\partial W_{ij}} = x_{ui}(h(\mathbf{x}) - y)\mathbf{x}_a^T\mathbf{V}_{*j}, \tag{8}$$

$$\frac{\partial \xi(\mathbf{x}, y)}{\partial V_{ij}} = x_{ai}(h(\mathbf{x}) - y)\mathbf{x}_u^T\mathbf{W}_{*j}, \tag{9}$$

where x_{ui}, x_{ai}, and x_{oi} denote the ith element in vectors $\mathbf{x}_u$, $\mathbf{x}_a$, and $\mathbf{x}_o$, respectively.

The regularization part in (6) can be expanded as follows:

$$\begin{aligned}\Omega(\mathbf{W}, \mathbf{V}) &= ||\mathbf{W}||_{21} + ||\mathbf{V}||_{21} \\ &= \sum_{i=1}^{l}\sqrt{\sum_{j=1}^{k} W_{ij}^2} + \sum_{i=1}^{s}\sqrt{\sum_{j=1}^{k} V_{ij}^2} \\ &\approx \sum_{i=1}^{l}\sqrt{\sum_{j=1}^{k} W_{ij}^2 + \epsilon} + \sum_{i=1}^{s}\sqrt{\sum_{j=1}^{k} V_{ij}^2 + \epsilon},\end{aligned}$$

where ϵ is a very small positive number to make the regularization term differentiable. Practically, it works well in our application.

The gradient of $\Omega(\mathbf{W}, \mathbf{V})$ can be derived as follows:

$$\frac{\partial \Omega(\mathbf{W}, \mathbf{V})}{\partial W_{ij}} = \frac{W_{ij}}{\sqrt{\sum_{j=1}^{k} W_{ij}^2 + \epsilon}}, \tag{10}$$

$$\frac{\partial \Omega(\mathbf{W}, \mathbf{V})}{\partial V_{ij}} = \frac{V_{ij}}{\sqrt{\sum_{j=1}^{k} V_{ij}^2 + \epsilon}}. \tag{11}$$

We can concatenate the parameter group ($\mathbf{W}$,$\mathbf{b}$) into a parameter vector, and then compute the gradient vector $\mathbf{g}(\mathbf{W}, \mathbf{b})$. Similarly, we can also compute the gradient vector $\mathbf{g}(\mathbf{V}, \mathbf{b})$ of the parameter group ($\mathbf{V}$,$\mathbf{b}$). Assume t is the τ-th parameter in each parameter group, the τ-th element in gradient vector $\mathbf{g}(\cdot)$ has the following form:

$$\mathbf{g}_\tau(\cdot) = \sum_{i=1}^{N} \frac{\partial \xi(\mathbf{x}^{(i)}, y^{(i)})}{\partial t} + \lambda \frac{\partial \Omega(\mathbf{W}, \mathbf{V})}{\partial t}, \tag{12}$$

where $\frac{\partial \xi(\mathbf{x}^{(i)}, y^{(i)})}{\partial t}$ is computed by using (7), (8), or (9), and $\frac{\partial \Omega(\mathbf{W}, \mathbf{V})}{\partial t}$ is computed by using (10) or (11), depending on the value of t. Then we can construct the approximate Hessian matrix $\tilde{\mathbf{H}}$ for L-BFGS.

The whole learning procedure for CGL is summarized in Algorithm 1. In each iteration, the alternating learning algorithm ensures that the objective function value always decreases. Furthermore, the objective function is bounded below by 0. Practically, when the whole loss function tends to be flat and the decrease turns to be flat, we can regard that as convergence. Because the objective function is not jointly convex in both $\mathbf{W}$ and $\mathbf{V}$, the solution is a local optimum. In our implementation, the convergence condition of the algorithm is that the relative decrease of the objective function value turns to be less than a threshold.

Algorithm 1 Alternate Learning for CGL

Input: Data set $\{(\mathbf{x}^{(i)}, y^{(i)}) \mid i = 1, ..., N\}$, and hyper-parameters $k \in \mathcal{N}^+$ and $\lambda \in \mathbb{R}^+$.
Output: $\mathbf{W}^*$, $\mathbf{V}^*$, $\mathbf{b}^*$
Initialize $\mathbf{b} = \mathbf{0}$.
Initialize $\mathbf{W} = random(\mathbb{R}^{l\times k})$, $\mathbf{V} = random(\mathbb{R}^{s\times k})$.
repeat
  Fix $\mathbf{V}$.
  repeat
    Compute gradient $\mathbf{g}(\mathbf{W}, \mathbf{b})$ using (12).
    Compute the approximate Hessian $\tilde{\mathbf{H}}_{\mathbf{W},\mathbf{b}}$ w.r.t. $(\mathbf{W}, \mathbf{b})$.
    $\mathbf{d}(\mathbf{W}, \mathbf{b}) = -\tilde{\mathbf{H}}_{\mathbf{W},\mathbf{b}} * \mathbf{g}(\mathbf{W}, \mathbf{b})$.
    Perform line search in the direction of $\mathbf{d}(\mathbf{W}, \mathbf{b})$ and update $\mathbf{W}, \mathbf{b}$.
  until convergence on $\mathbf{W}, \mathbf{b}$
  Fix $\mathbf{W}$.
  repeat
    Compute gradient $\mathbf{g}(\mathbf{V}, \mathbf{b})$ using (12).
    Compute the approximate Hessian $\tilde{\mathbf{H}}_{\mathbf{V},\mathbf{b}}$ w.r.t. $(\mathbf{V}, \mathbf{b})$.
    $\mathbf{d}(\mathbf{V}, \mathbf{b}) = -\tilde{\mathbf{H}}_{\mathbf{V},\mathbf{b}} * \mathbf{g}(\mathbf{V}, \mathbf{b})$.
    Perform line search in the direction of $\mathbf{d}(\mathbf{V}, \mathbf{b})$ and update $\mathbf{V}, \mathbf{b}$.
  until convergence on $\mathbf{V}, \mathbf{b}$
until convergence

3.3. Complexity Analysis

Let $q = (l + s)k + d$ denote the total number of parameters in $\mathbf{W}$, $\mathbf{V}$ and $\mathbf{b}$. To train the model, we need $O(qN)$ time to compute the gradient $\mathbf{g}(\cdot)$, $O(q^2)$ time to compute the approximate Hessian matrix and $O(q^2)$ time for matrix multiplication and parameter update in each iteration. Hence, the total time complexity is $O(qN + q^2)\mu$ for μ iterations.

4. Web-Scale Implementation

Web-scale applications always contain a huge number of users and ads, with billions of impression instances. Hence, we need a scalable learning framework. In this section, we first introduce the hashing and sub-sampling techniques used for memory saving and class-unbalance handling. Furthermore, we propose a distributed learning framework based on message passing interface (MPI), which can run on clusters with hundreds of computing nodes. By combining these techniques, our learning algorithm is highly scalable for web-scale applications.

4.1. Hashing and Sub-Sampling

We use the hashing technique (Weinberger et al., 2009) for efficient feature mapping and instance generating. The original features used in real CTR prediction systems are mainly categorical, the number of which is typically very large. In order to make the feature mapping (coding) results uniform and enable fast instance generating, we hash user, ad and the other (context) features to three separate subspaces of bit vectors. The structure of the hashing framework is illustrated in Figure 1, in which the *raw representation* of an impression is hashed to the *instance representation* with bit vectors. The raw representation of each impression is composed of several triples, with each triple being *(domain, feature name, feature value)*. The *domain* can be *user, ad, or other*, which refers to the types of the features. For example, the impression in Figure 1 contains k triples, (d1, f1, v1), (d2, f2, v2), ... , (dk, fk, vk). The instance representation of each impression is composed of three subspaces of bit vectors, which are denoted as *User features*, *Ad features* and *Other features* in Figure 1. Given a triple, we can get the index (also called position or key) of its coding in the feature space quickly with a hash function. For example, if an impression has a user (*domain*) feature 'buypower' (*feature name*) with a value '5' (*feature value*), the hash function maps this triple (*user, buypower, 5*) to one position in the subspace of *User features*, and sets the corresponding position to be 1. The <position (key), triple> pairs are stored in the *feature map* for later searching and checking. The hashing technique enables efficient feature engineering and fast prediction together with straightforward implementation. According to (Weinberger et al., 2009), the hashing can not only contract the feature space by collision, but also bring a regularization effect.

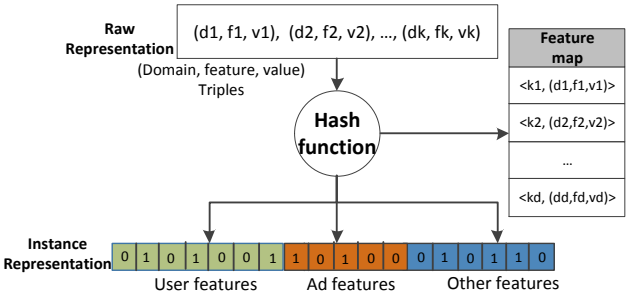


Figure 1. The hashing framework for fast feature mapping and instance generating.

The data sets are typically highly unbalanced, with only a very small proportion of positive instances. In order to reduce the learning complexity with a minimal influence

on accuracy, we sample negative instances with a probability of $\gamma = 10\%$ and keep all the positive instances. After sampling, we give a weight $\frac{1}{\gamma}$ to each negative instance during learning to make the objective calculation unbiased (McMahan et al., 2013).

4.2. Distributed Learning

In each iteration of Algorithm 1, we need to compute the gradient of all the parameters by using (12) and compute the current value of the objective function in (4). Both of these two equations contain summations over all instances, which can be easily parallelized on one machine (node) with multi-thread techniques or can be distributed across several machines in a cluster with MPI.

We implement a distributed learning framework for CGL based on an MPI-cluster with hundreds of nodes (machines) and make full use of the parallel property of gradient computing of the parameters. Let P be the number of nodes. We first evenly distribute the whole data set to each node, and the allocation of data will not be changed as the algorithm proceeds, which can minimize data movement and communication cost.

The *AllReduce* and *BroadCast* interfaces in MPI are used to communicate between master and slaver nodes. Furthermore, a synchronized routine is used to ensure the correctness of our system. We denote the gradient of parameters locally on node p as $\mathbf{g}'_p$. Please note that $\mathbf{g}'_p$ only contains the gradient of the instance part, with the regularization part discarded. The τ-th element in gradient vector $\mathbf{g}'_p$ has the following form:

$$\mathbf{g}'_{p_\tau} = \sum_{i=1}^{p_n} \frac{\partial \xi(\mathbf{x}^{(i)}, y^{(i)})}{\partial t}, \tag{13}$$

where p_n is the number of instances allocated to node p, and t is the τ-th parameter as stated in (12).

The sketch of the distributed learning framework is summarized in Algorithm 2.

Algorithm 2 Distributed Learning Framework for CGL

Input: Data set $\{(\mathbf{x}^{(i)}, y^{(i)}) \mid i = 1, ..., N\}$, and hyper-parameters $k \in \mathcal{N}^+$ and $\lambda \in \mathbb{R}^+$.
Initialize Num of Nodes: P.
Initialize Parameters: $\mathbf{W}, \mathbf{V}, \mathbf{b}$.
Split the data set across P nodes evenly.
repeat
  for all nodes $\{p = 1, 2, \cdots, P\}$ **do in parallel**
    Compute gradient $\mathbf{g}'_p$ locally on node p using (13).
  end for
  Compute gradient $\mathbf{g}' = \sum_{p=1}^{P} \mathbf{g}'_p$ with *AllReduce* .
  Add the gradient of the regularization term to $\mathbf{g}'$ in the master node.
  Take an L-BFGS step in the master node.
  BroadCast the updated parameters to each slaver node.
until convergence

5. Experiment

We have an MPI-cluster with hundreds of nodes, each of which is a 24-core server with 2.2GHz Intel(R) Xeon(R) E5-2430 processor and 96GB of RAM. However, we only use 80 nodes of the cluster for our experiment. One reason is that 80 nodes are enough to handle real web-scale CTR applications. Another reason is that the whole cluster is shared by many groups who are running different jobs. We do not want to interrupt other jobs. However, our method is highly scalable to use more nodes for larger scale problems, which will be verified by our following experiments (refer to Figure 4).

5.1. Data Set

We conduct our experiment on three real-world data sets connected from Taobao of Alibaba group [1]. The training sets of these three data sets contain the log information of display ads across different time periods with different time window sizes, and the subsequent (next) day's log information of each training set is for testing to evaluate the performance of our model. We compose the data sets on weekdays or holidays from different months to make the data sets vary from each other and make the results of our model more convincing. There are billions of impression instances in each data set, and the dimensionality of the feature vector for each impression is tens of thousands.

These three data sets are named as *Dataset-1, Dataset-2* and *Dataset-3*, respectively. *Train 1* and *Test 1* are the training set and test set for *Dataset-1*. Similar names can be got for other two data sets. The characteristics of these three data sets are briefly summarized in Table 1.[2] Please note that the CTR in Table 1 for a data set is computed as $\frac{Number\ of\ Clicks}{Number\ of\ Impressions}$, which is actually the proportion of positive instances in the data set. We can find that all the data sets are highly unbalanced, with only a very small proportion of positive instances. It's easy to find that the data set sizes of our experiments are of web-scale.

We sample 20% of each training set for validation to specify the hyper-parameters of our CGL model and other baselines. The k in CGL is fixed to 50 in our experiment unless otherwise stated.

[1] We build our data sets from the logs of the advertisements displayed in http://www.taobao.com, one of the most famous C2C e-commerce web sites in China. The business model of Taobao is similar to that of eBay (http://www.ebay.com).

[2] Dataset-2 contains the log information of a long holiday, during which people may go outside and have no time surfing the internet. So the number of instances and users is relatively small.

Table 1. Characteristics of the three data sets which contain training data of 4 days, 10 days and 7 days from different time periods, respectively. The subsequent day's log information of each training set is for test set.

DATA SET	# INSTANCES (IN BILLION)	CTR (IN %)	# ADS	# USERS (IN MILLION)	STORAGE (IN TB)
TRAIN 1	1.011	1.62	21,318	874.7	1.895
TEST 1	0.295	1.70	11,558	331.0	0.646
TRAIN 2	1.184	1.61	21,620	958.6	2.203
TEST 2	0.145	1.64	6,848	190.3	0.269
TRAIN 3	1.491	1.75	33,538	1119.3	2.865
TEST 3	0.126	1.70	9,437	183.7	0.233

5.2. Evaluation Metrics and Baseline

5.2.1. METRIC

We can regard CTR prediction as a binary classification problem. Because the data set is highly unbalanced with only a small proportion of positive instances, prediction accuracy is not a good metric for evaluation. Furthermore, neither precision nor recall is a good metric. In this paper, we adopt the area under the receiver operating characteristic curve (AUC) (Bradley, 1997) as a metric to measure the prediction accuracy, which has been widely used in existing literatures for CTR prediction (Chapelle et al., 2013). For a random guesser, the AUC value will be 0.5, which means total lack of discrimination. In order to have a good comparison with baseline models, we first remove this constant part (0.5) from the AUC value and then compute the relative improvement ($RelaImpr$) of our model, which has the following mathematical form:

$$RelaImpr = \frac{AUC(model) - 0.5}{AUC(baseline) - 0.5} \times 100\%.$$

This $RelaImpr$ metric has actually been widely adopted in industry for comparing discrimination of models [3].

Our CGL model has an effect of selecting features or eliminating features for both users and ads. We introduce *group sparsity* ($GSparsity$) to measure the capability of our model in feature elimination: $GSparsity = \frac{\nu}{l+s} \times 100\%$, where ν is the total number of all-zero rows in parameter matrices $\mathbf{W}$ and $\mathbf{V}$, l and s are the number of rows in $\mathbf{W}$ and $\mathbf{V}$ respectively.

5.2.2. BASELINE

Because LR model (with L_2-norm) has been widely used for CTR prediction and has achieved the state-of-the-art performance, especially in industrial systems (Chapelle et al., 2013), we adopt LR as the baseline for comparison. Please note that LR refers to the model in (1) with L_2-norm regularization, and the model in (1) with L_1-norm regularization is called *Lasso* in this paper.

[3] http://en.wikipedia.org/wiki/Receiver_operating_characteristic

5.3. Accuracy of Lasso

Table 2 is the relative improvement ($RelaImpr$) of Lasso w.r.t. the baseline (LR). We can see that there does not exist significant difference between LR and Lasso in terms of prediction accuracy.

Table 2. Relative improvement of Lasso w.r.t. the baseline (LR).

DATA SET	DATASET-1	DATASET-2	DATASET-3
RELAIMPR	−0.019%	−0.096%	+0.086%

5.4. Accuracy of CGL

Please note that in Algorithm 1, the $\mathbf{W}$ and $\mathbf{V}$ are randomly initialized, which may affect the performance. We perform six independent rounds of experiments with different initialization. The mean and variance of the relative improvement of our CGL model (with $k = 50$) w.r.t. the baseline LR are reported in Figure 2. It is easy to find that our CGL model can significantly outperform LR in all three data sets. Furthermore, we can find that the random initialization has an ignorable influence on the performance. Hence, in the following experiments, we won't report the variance of the values.

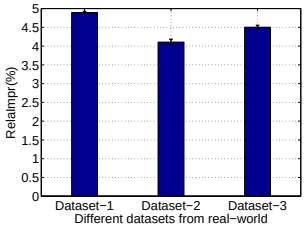
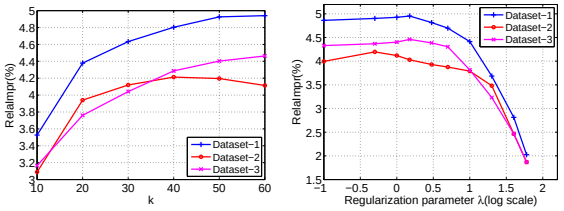


Figure 2. The relative improvement of CGL w.r.t. baseline (LR).

5.5. Sensitivity to Hyper-Parameters

In this subsection, we study the influence of the two key hyper-parameters, k and λ, in our CGL model.

Experiments are conducted for different k and the results are shown in Figure 3 (a). We can find that, with the increasing of k, the performance turns to be better in general. But larger k implies more parameters, which can make the



(a) Influence of k (b) Influence of λ

Figure 3. The influence of hyper-parameters k and λ.

learning much harder in terms of both memory and speed. We find that $k = 50$ is a suitable value for our experiments. Hence, we choose $k = 50$ for all the experiments in this paper.

We vary the values of the hyper-parameter λ and draw the influence on the performance in Figure 3 (b). We can find that very good performance can be achieved when λ is around 1 for all data sets, and our CGL is not sensitive to λ in a relatively large range (from 0.1 to 10).

Actually, the λ controls the tradeoff between the prediction accuracy and number of eliminated features ($GSparsity$). We choose Dataset-2 for demonstration. The relationship of relative improvement and $GSparsity$ on Dataset-2 is shown in Table 3. We can find that our CGL does have the ability to eliminate some features. For this data set, a $GSparsity$ of $3\% - 15\%$ will be a good tradeoff for both feature elimination and prediction accuracy.

Table 3. Tradeoff between relative improvement of performance w.r.t. baseline (LR) and $GSparsity$ on Dataset-2.

GSPARSITY	2%	3%	5%	15%	20%
RELAIMPR	3.90%	3.42%	3.02%	2.5%	1.97%

We take a deep look at the parameter matrices for users and ads, and show the most important features and the most useless features in Table 4. They are all categorical features, in which the most important features for ad part are popular or hot product categories, such as *clothes, skirt* and *dress*. This seems to be reasonable. The useless features for ad part contain *Movie, Act, Takeout, Food booking service*. This is also reasonable because few users buy these products like *food booking service* from Taobao. The most important features for user part are categories they show great interest, such as daily necessities and clothes. The useless features for users are some cold categories and some rare items, such as *stage costume and flooring*.

5.6. Scalability

To study the scalability of our distributed learning framework, we compute the speedup factor relative to the running time with 20 nodes by varying the number of nodes from 20 to 80. Experiments are repeated several times, and the mean and variance of the speedup factor are reported in Figure 4. We can find that the speedup appears to be close to linear, and close to the ideal speedup factors. Hence, our CGL is very scalable for web-scale applications.

Table 4. Feature selection results.

	AD PART	USER PART
IMPORTANT FEATURES	WOMEN'S CLOTHES, SKIRT, DRESS, CHILDREN'S WEAR, SHOES, CELLPHONE	WATCH, UNDERWEAR, FUR CLOTHING, FURNITURE
USELESS FEATURES	MOVIE, ACT, TAKEOUT, FOOD BOOKING SERVICE	STAGE COSTUME, FLOORING, PENCIL, OUTDOOR SOCK

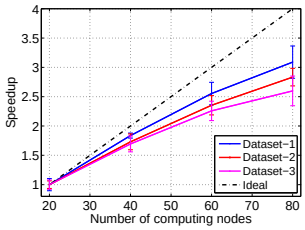


Figure 4. The speedup of the distributed learning framework.

The whole training time (in second) of CGL on 80 nodes is shown in Table 5, from which we can find that CGL is fast enough to handle web-scale applications.

Table 5. Training time (in second).

DATA SET	DATASET-1	DATASET-2	DATASET-3
TIME	$3,184 \pm 431$	$3,296 \pm 387$	$4,281 \pm 541$

6. Conclusion and Future Work

In this paper, a novel model called CGL is proposed to capture the conjunction information between features, which can outperform the state-of-the-art models in web-scale CTR prediction for display advertising. Actually, our CGL is general enough to model other similar applications with outputs determined by two interacting roles. One of our future works is to pursue new applications of our model.

7. Acknowledgement

This work is supported by the NSFC (No. 61100125), the 863 Program of China (No. 2012AA011003), and the Program for Changjiang Scholars and Innovative Research Team in University of China (IRT1158, PCSIRT). Wu-Jun Li is the corresponding author.

References

Agarwal, Deepak, Agrawal, Rahul, Khanna, Rajiv, and Kota, Nagaraj. Estimating rates of rare events with multiple hierarchies through scalable log-linear models. In *KDD*, pp. 213–222, 2010.

Andrew, Galen and Gao, Jianfeng. Scalable training of l_1-regularized log-linear models. In *ICML*, pp. 33–40, 2007.

Bradley, Andrew P. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.

Broyden, C. G. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

Byrd, Richard H., Nocedal, Jorge, and Schnabel, Robert B. Representations of quasi-newton matrices and their use in limited memory methods. *Mathematical Programming*, 63:129–156, 1994.

Chapelle, Oliver, Manavoglu, Eren, and Rosales, Romer. Simple and scalable response prediction for display advertising. *ACM Transactions on Intelligent Systems and Technology*, 2013.

Golub, Gene H, Hansen, Per Christian, and O'Leary, Dianne P. Tikhonov regularization and total least squares. *SIAM Journal on Matrix Analysis and Applications*, 21 (1):185–194, 1999.

Graepel, Thore, Candela, Joaquin Quiñonero, Borchert, Thomas, and Herbrich, Ralf. Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. In *ICML*, pp. 13–20, 2010.

Kuang-chih, Lee, Orten, Burkay, Dasdan, Ali, and Li, Wentong. Estimating conversion rate in display advertising from past performance data. In *KDD*, pp. 768–776, 2012.

Mahdian, Mohammad and Tomak, Kerem. Pay-per-action model for online advertising. In *WINE*, pp. 549–557, 2007.

Malouf, Robert. A comparison of algorithms for maximum entropy parameter estimation. In *CoNLL*, pp. 49–55, 2002.

McMahan, H. Brendan, Holt, Gary, Sculley, David, Young, Michael, Ebner, Dietmar, Grady, Julian, Nie, Lan, Phillips, Todd, Davydov, Eugene, Golovin, Daniel, Chikkerur, Sharat, Liu, Dan, Wattenberg, Martin, Hrafnkelsson, Arnar Mar, Boulos, Tom, and Kubica, Jeremy. Ad click prediction: a view from the trenches. In *KDD*, pp. 1222–1230, 2013.

Meier, Lukas, Van De Geer, Sara, and Bühlmann, Peter. The group lasso for logistic regression. *Journal of the Royal Statistical Society, Series B*, 70(1):53–71, 2008.

Menon, Aditya Krishna, Chitrapura, Krishna Prasad, Garg, Sachin, Agarwal, Deepak, and Kota, Nagaraj. Response prediction using collaborative filtering with hierarchies and side-information. In *KDD*, pp. 141–149, 2011.

Muthukrishnan, S. Ad exchanges: Research issues. In *WINE*, pp. 1–12, 2009.

Neter, John, Wasserman, William, and Kutner, Michael H. *Applied linear statistical models*, volume 4. Irwin Chicago, 1996.

Nocedal, Jorge. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35(151): 773–782, 1980.

Richardson, Matthew, Dominowska, Ewa, and Ragno, Robert. Predicting clicks: estimating the click-through rate for new ads. In *WWW*, pp. 521–530, 2007.

Stern, David H., Herbrich, Ralf, and Graepel, Thore. Matchbox: large scale online bayesian recommendations. In *WWW*, pp. 111–120, 2009.

Tibshirani, Robert. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B*, pp. 267–288, 1996.

Weinberger, Kilian Q., Dasgupta, Anirban, Langford, John, Smola, Alexander J., and Attenberg, Josh. Feature hashing for large scale multitask learning. In *ICML*, pp. 140, 2009.

Yuan, Ming and Lin, Yi. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society, Series B*, 68:49–67, 2006.