IEICE TRANS. INF. & SYST., VOL.E99–D, NO.4 APRIL 2016

PAPER Special Section on Data Engineering and Information Management

BLM-Rank: A Bayesian Linear Method for Learning to Rank and Its GPU Implementation*

Huifeng GUO[†a)], **Dianhui CHU**[††b)], ***Nonmembers***, **Yunming YE**[†c)], ***Member***, **Xutao LI**[†d)], ***and*** **Xixian FAN**[†††], ***Nonmembers***

SUMMARY Ranking as an important task in information systems has many applications, such as document/webpage retrieval, collaborative filtering and advertising. The last decade has witnessed a growing interest in the study of learning to rank as a means to leverage training information in a system. In this paper, we propose a new learning to rank method, i.e. BLM-Rank, which uses a linear function to score samples and models the pairwise preference of samples relying on their scores under a Bayesian framework. A stochastic gradient approach is adopted to maximize the posterior probability in BLM-Rank. For industrial practice, we have also implemented the proposed algorithm on Graphic Processing Unit (GPU). Experimental results on LETOR have demonstrated that the proposed BLM-Rank method outperforms the state-of-the-art methods, including RankSVM-Struct, RankBoost, AdaRank-NDCG, AdaRank-MAP and ListNet. Moreover, the results have shown that the GPU implementation of the BLM-Rank method is ten-to-eleven times faster than its CPU counterpart in the training phase, and one-to-four times faster in the testing phase.

key words: *ranking, Bayesian Personalized Ranking, stochastic gradient method, GPU*

1. Introduction

Ranking plays a central role in information systems. It can be used in a wide variety of applications, including document/webpage retrieval, popularity analysis of celebrities or news, collaborative filtering and advertising, to name just a few. In the literature, many ranking methods have been developed based on different theories and methodologies. For instance, PageRank [1] and HITS [2] are two well-known ranking methods based on link structures. Recently, these two methods have been extended by using multi-relational link structures [3], [4]. In addition, some researchers have proposed to design ranking algorithms in terms of language model [5], probability model [6] as well as BM25 model [7]. However, all these methods produce rankings in an unsupervised manner without including a learning process.

In recent years, researchers are becoming to pay their attentions to supervised machine learning techniques by leveraging training information such that more accurate ranking functions can be accomplished. Methods of such kind are referred to as *learning to rank*. It has become one of the core technologies in modern retrieval systems. Many algorithms relying on this technology have been proposed.

In general, these algorithms can be classified into three categories, including pointwise approaches, pairwise approaches and listwise approaches. Pointwise approaches define objective functions regarding the relevance score of an individual webpage relying on regression models [8]. Pairwise approaches define objective functions regarding the preferences of pairwise webpages. As a result, the ranking function learning problem is transferred into a binary classification problem. RankSVM [9], RankBoost [10], RankNet [11] are algorithms belonging to this category. Different from these two categories, listwise approaches define objective functions to optimize directly or indirectly information retrieval measures. For example, the AdaRank algorithm optimizes the NDCG/MAP directly by employing boosting techniques [12]. Instead of optimizing information retrieval evaluation metrics directly, the ListNet algorithm minimizes the KL divergence of two probability distributions that are constructed from training list with a permutation model and a top k model [13].

In this paper, we propose BLM-Rank, a Bayesian linear method aiming to learn ranking functions from a training set. The basic idea of this approach is to use a linear function evaluating the scores of all training samples with respect to a query. However, instead of fitting directly the relevance scores of samples in the training set as pointwise approaches do, we incorporate this function into a Bayesian framework to model the probability of each preference pair of training samples. As a sequel, Bayesian parameter estimation is performed to fit this linear function. Compared to aforementioned learning to rank algorithms, the proposed method has two distinguished advantages. First, the linear function is obtained by fitting pairwise preferences of training samples rather than relevance scores of them, which has potential to be applied in many other applications. The reason is

Manuscript received June 18, 2015.
Manuscript revised October 2, 2015.
Manuscript publicized January 14, 2016.

[†]The authors are with the Shenzhen Key Laboratory of Internet Information Collaboration, Shenzhen Graduate School, Harbin Institute of Technology, China.
[††]The author is with the Department of Computer Science and Technology, Harbin Institute of Technology at Weihai, China.
[†††]The author is with Baidu Inc., China.

*This research was supported in part by NSFC under Grant No. 61572158 and No. 61272538, and Shenzhen Science and Technology Program under Grant No. JCYJ20140417172417128, and research Award fund for outstanding young scientists in Shandong Province No. 2014BSA10016, and Shandong Province Science and Technology major projects No. 2015ZDXX0201B02.

a) E-mail: huifengguo@yeah.org
b) E-mail: cdh2004@sina.com (Corresponding author)
c) E-mail: yeyunming@hit.edu.cn
d) E-mail: lixutao@hitsz.edu.cn

DOI: 10.1587/transinf.2015DAP0001

Copyright © 2016 The Institute of Electronics, Information and Communication Engineers

that pairwise preferences are easier to obtain than relevance scores in real applications. For example, a click through data in search engine helps us identify which webpage is more important than others, rather than give the relevance scores of all webpages. Second, Bayesian learning is adopted, which has good generalization ability to overcome overfitting problem. For real applications, in this paper we also give the GPU implementation of the proposed method. Experimental results on LETOR data sets have demonstrated the proposed method outperforms RankSVM-Struct, RankBoost, AdaRank-NDCG, AdaRank-MAP and ListNet on retrieval performance. Moreover, experimental results have shown that the GPU implementation of the proposed method is ten to eleven times faster than its CPU counterpart in the training phrase, and one to four times faster in the testing phrase.

Our main contributions in this paper can be summarized as the following three folds:

- We propose a new linear learning to rank method based on Bayesian Personalized Ranking criterion. The method is a pairwise approach and capable of learning ranking function for documents/items with their relative relevancies, which has great potential to be applied in many applications because relative relevancies are easier to obtain than precise relevance scores.
- We develop a GPU implementation of the proposed method, which substantially improves the efficiency in both training and testing phases.
- Extensive experiments on various publicly available datasets are conducted, and the results demonstrate the proposed method outperforms state-of-the-art methods in terms of retrieval accuracy. Moreover, experimental results manifest the developed GPU implementation can speed up the proposed method considerably.

The rest of this paper is organized as follows. In Sect. 2, related work is briefly reviewed. In Sect. 3, we present the proposed Bayesian linear method for learning to rank. Section 4 discusses the GPU implementation. Experimental results are given in Sect. 5. Finally, we give the concluding remarks in Sect. 6.

2. Related Work

2.1 Learning to Rank

As aforementioned, learning to rank methods are mainly divided into three categories, namely, pointwise approaches, pairwise approaches and listwise approaches. Here, we give a brief review on them respectively.

2.1.1 Pointwise Approaches

Pointwise approaches are the earliest ways to learn ranking functions. Its central idea is to map this ordinal scales of samples into numerical values and then consider this problem as ordinal regression. For example, PRank model proposed in [14] is a classical method in this category. This model considers query/document pair as an input instance for a perceptron network, and trains this network by minimizing the difference between the predicted rank level and the ground truth one of each instance. Similar to PRank, RankProp [15] is another neural network method in this category. In RankProp, the neural network has two phases: an MSE regression phase on the target numerical values, and an adjustment phase of target values themselves to reflect the ranking.

2.1.2 Pairwise Approaches

Pairwise approaches treat each document pair with preference information as a training instance, and then transfer the learning problem into a classification problem by regarding the pairwise preferences as class labels. For instance, given a document pair $(\mathbf{d}_1, \mathbf{d}_2)$ as an instance, if the document $\mathbf{d}_1$ has a higher relevance score than the document $\mathbf{d}_2$, this instance is labeled as a positive example, otherwise it is labeled as a negative example. In the literature, there are many algorithms falling into this category. In [10], RankBoost algorithm has been proposed based on AdaBoost so as to learn ranking functions by boosting weak classifiers. RankSVM [9] learns ranking functions by exploiting SVM to maximize the margin between positive and negative examples constructed from pairwise preferences. RankNet [11] employs a neural network to learn ranking functions by minimizing the cross entropy that measures the difference between the modeled probabilities of pair preferences and the ground truth ones. Besides, LambdaRank [16], FRank [17] and Multiple-nested-ranker [18] are all pairwise approaches.

2.1.3 Listwise Approaches

Compared to the above two categories, listwise approaches are relatively new. Listwise approach addresses this problem by considering the ranking lists as instances and defining a listwise objective function. In this case, some information retrieval evaluation metric can be incorporated directly or indirectly into such objective function for optimization. For example, in [12], an AdaRank algorithm optimizes directly the NDCG/MAP by employing boosting techniques. It repeatedly constructs "weaker rankers" based on re-weighted training samples and finally combines them linearly for ranking. In [13], a ListNet algorithm employing a neural network optimizes a listwise objective function based on the top k probability. ListNet is very similar to RankNet in pairwise category. The major difference is that ListNet uses ranking lists as instances to define loss function, while RankNet uses document pairs as instances.

2.2 Bayesian Personalized Ranking

Bayesian Personalized Ranking (BPR) [19] is a generic optimization criterion to consider the personalized ranking from the perspective of Bayesian analysis. Its basic idea is to

maximize the posterior estimators based on training data. For optimization, stochastic gradient method is employed in this method. Due to its good performance and interpretation, BPR has been applied to design different algorithms for various applications. In [20], a tensor factorization model has been proposed based on BPR framework for tag recommendation. In [21], a collaborative group ranking method has been established using BPR framework for product recommendation purpose. In [22], Artus et al. have proposed a multi-relational matrix factorization model relying on the BPR criterion for analyzing social network data. In [23], Lerche et al. have proposed to incorporate implicit feedback signals at a more fine-grained level in the personalized ranking process. Different from these existing methods, we make use of BPR framework to solve the learning to rank problems in this paper.

3. BLM-Rank: A Bayesian Linear Method

In this section, we introduce the proposed Bayesian Linear Method for learning to rank. Despite this method belonging to pairwise approaches as well, it does not transfer the learning problem into a classification problem but a Bayesian learning problem employing BPR criterion.

3.1 Construction of Pairwise Training Samples

Let $\mathcal{Q} = \{q_1, q_2, \cdots, q_n\}$ denote the set of n queries in the training data, and let $\mathcal{D} = \{d_1, d_2, \cdots, d_m\}$ denote the set of m documents. Let a $m \times 1$ vector $\mathbf{y}^{(i)} = [y_1^{(i)}, y_2^{(i)}, \cdots, y_m^{(i)}]^T$ be the relevance scores or grades of these documents with respect to the i-th query. Let $\phi(\cdot) : \mathcal{Q} \times \mathcal{D} \to R^p$ be a feature function that is used to construct features for a particular document with respect to a query. In other words, given the i-th query, the j-th document can be represented as a $p \times 1$ vector $\mathbf{x}_{i,j} = \phi(q_i, d_j)$. Many features can be used or constructed for learning to rank purpose, such as PageRank feature, BM25 feature, anchor text feature [24]. As a result, the training data can be represented as $\mathcal{T} = \left\{\left(\{\mathbf{x}_{i,j}\}_{j=1}^m, \mathbf{y}^{(i)}\right)\right\}_{i=1}^n$.

As we aim to model the pairwise preference relationship of documents, it is necessary to construct pairwise training samples from $\mathcal{T}$. In information systems, the pairwise preferences of documents are usually query dependent. For example, document d_1 may be more relevant than document d_2 for query q_1, while this may not be true for other queries. Therefore, a query dependent partial order for documents needs to be defined ahead of the construction of pairwise training samples.

Definition 1: Given a query $q_i \in \mathcal{Q}$ and two documents $d_j, d_k \in \mathcal{D}$, we define $d_j \succ_{q_i} d_k$ if $y_j^{(i)} > y_k^{(i)}$, where $\succ_q$ indicates a query dependent partial order relationship of documents.

Following this partial order relationship, we are able to create a training set $\mathcal{T}^+$ of positive preference pairs from $\mathcal{T}$:

$$\mathcal{T}^+ = \left\{(\mathbf{x}_{i,j}, \mathbf{x}_{i,k}) | d_j \succ_{q_i} d_k, q_i \in \mathcal{Q} \text{ and } d_j, d_k \in \mathcal{D}\right\}.$$

Clearly, the instances in $\mathcal{T}^+$ are a set of document pairs that have partial order relationship with respect to the queries in the original training data. In each instance, the former document is more relevant than the latter one regarding a particular query.

Even though the pairwise training samples are constructed based on a partial order defined on the relevance scores of documents, it is worth pointing out that the definition of partial order can be very general, as long as this definition indicates the relative importance of them. For example, based on the click through data in search engine, we can infer the relative importance of webpages. One webpage is more important than others for a query if it has a higher clicking rate. As a result, pairwise training samples can be constructed based on clicking rate for this example.

3.2 BLM-Rank Model

In this section, we aim to learn a probability model from the training data such that it has the largest posterior probability to generate the preference pairs in the training data. According to the Bayesian theory, the posterior probability can be conceptually expressed as:

$$p(\mathbf{w}|\mathcal{T}^+) = \frac{p(\mathcal{T}^+|\mathbf{w})p(\mathbf{w})}{p(\mathcal{T}^+)} \tag{1}$$

where $\mathbf{w}$ represents the parameters for arbitrary models, and $\mathcal{T}^+$ is the training set. Thus, we would like to maximize

$$p(\mathbf{w}|\mathcal{T}^+) \propto p(\mathcal{T}^+|\mathbf{w})p(\mathbf{w}), \tag{2}$$

as $p(\mathcal{T}^+)$ is a constant for a given training data.

To achieve the solution in Eq. (2), we introduce how to model the likelihood $p(\mathcal{T}^+|\mathbf{w})$ next. In particular, we first employ a linear function to estimate the relevance score of each document d_j with respect to a query q_i as follows:

$$F(\phi(q_i, d_j), \mathbf{w}) = F(\mathbf{x}_{i,j}, \mathbf{w}) = \mathbf{w}^T \mathbf{x}_{i,j} \tag{3}$$

where $\mathbf{w}$ is a $p \times 1$ parameter vector to be learned. With this linear function, we are able to compute the difference of relevance scores for any two documents d_j and d_k with respect to a query q_i:

$$\begin{aligned} r_{i,j,k} &= F(\phi(q_i, d_j), \mathbf{w}) - F(\phi(q_i, d_k), \mathbf{w}) \\ &= \mathbf{w}^T \mathbf{x}_{i,j} - \mathbf{w}^T \mathbf{x}_{i,k}. \end{aligned} \tag{4}$$

For a given instance $(\mathbf{x}_{i,j}, \mathbf{x}_{i,k}) \in \mathcal{T}^+$, by using the logistic sigmoid function, we thus can model the probability to generate the instance as:

$$\begin{aligned} p((\mathbf{x}_{i,j}, \mathbf{x}_{i,k})|\mathbf{w}) &= p(d_j \succ_{q_i} d_k|\mathbf{w}) \\ &= \frac{1}{1 + e^{-r_{i,j,k}}} \\ &= \frac{1}{1 + e^{\mathbf{w}^T \mathbf{x}_{i,k} - \mathbf{w}^T \mathbf{x}_{i,j}}}. \end{aligned} \tag{5}$$

Assuming that all the instances in $\mathcal{T}^+$ are independent and identically distributed (i.i.d.), we obtain the likelihood as

follows:

$$p(\mathcal{T}^+|\mathbf{w}) = \prod_{(\mathbf{x}_{i,j},\mathbf{x}_{i,k})\in\mathcal{T}^+} p((\mathbf{x}_{i,j},\mathbf{x}_{i,k})|\mathbf{w}) = \prod_{(\mathbf{x}_{i,j},\mathbf{x}_{i,k})\in\mathcal{T}^+} \frac{1}{1+e^{\mathbf{w}^T\mathbf{x}_{i,k}-\mathbf{w}^T\mathbf{x}_{i,j}}} \quad (6)$$

Although the i.i.d. assumption may not be exactly correct (due to the way we construct preference pairs, some pairs can be dependent), it has been widely used in many similar machine learning models (e.g., BPR model [19], which assumes that preference pairs are i.i.d. as we do despite they are not i.i.d.), and yields very promising performance. Hence, we follow previous work and make such an assumption here.

Equation (6) is the likelihood function of generating the preference pairs in $\mathcal{T}^+$ relying on the linear function $F(\cdot)$. As we learn the linear function $F(\cdot)$ using Bayesian learning framework, it is necessary to introduce a prior density function $p(\mathbf{w})$ for $F(\cdot)$. We assume that $p(\mathbf{w})$ is a normal distribution with zero mean and covariance matrix $\Sigma_\mathbf{w}$, namely,

$$p(\mathbf{w}) \sim N(\mathbf{0}, \Sigma_\mathbf{w}) \quad (7)$$

Since we have no prior knowledge about the correlation between p features as well as their importance in the linear function, it is reasonable to assume the covariance matrix $\Sigma_\mathbf{w}$ is a diagonal matrix with equal elements, i.e., $\Sigma_\mathbf{w} = \frac{1}{\lambda}I$, where I is a $p \times p$ identity matrix.

Substituting Eqs. (7) and (6) into Eq. (2) and using logarithm form, we have the following objective function to maximize for our BLM-Rank model:

$$\begin{aligned}\ln p(\mathbf{w}|\mathcal{T}^+) &\propto \ln p(\mathcal{T}^+|\mathbf{w}) + \ln p(\mathbf{w}) \\ &= \sum_{(\mathbf{x}_{i,j},\mathbf{x}_{i,k})\in\mathcal{T}^+} \ln\frac{1}{1+e^{\mathbf{w}^T(\mathbf{x}_{i,k}-\mathbf{x}_{i,j})}} - \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \\ &= \sum_{(\mathbf{x}_{i,j},\mathbf{x}_{i,k})\in\mathcal{T}^+} \ln\frac{1}{1+e^{(-r_{i,j,k})}} - \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \quad (8)\end{aligned}$$

where λ is a regularization parameter to be specified by users of BLM-Rank model.

3.3 The Algorithm

In order to maximize the posterior probability in (8) for BLM-Rank model, we adopt the gradient method because of its simplicity and effectiveness. Taking derivative with respect to $\mathbf{w}$ in (8), we have

$$\nabla_\mathbf{w} = \sum_{(\mathbf{x}_{i,j},x_{i,k})\in\mathcal{T}^+} \frac{-e^{\mathbf{w}^T(\mathbf{x}_{i,k}-\mathbf{x}_{i,j})}}{1+e^{\mathbf{w}^T(\mathbf{x}_{i,k}-\mathbf{x}_{i,j})}}(\mathbf{x}_{i,k}-\mathbf{x}_{i,j}) - \lambda\cdot\mathbf{w} \quad (9)$$

There are two possible ways to perform gradient optimization: one is the full gradient method; the other is the stochastic gradient method. For the first method, we compute the full gradient $\nabla_\mathbf{w}(\ln p(\mathbf{w}))$ based on all the training pairs in $\mathcal{T}^+$ and then update the parameter $\mathbf{w}$ in BLM-Rank model as follows:

$$\mathbf{w} = \mathbf{w} + \alpha\cdot\nabla_\mathbf{w} \quad (10)$$

where $\alpha > 0$ is the learning rate. This method is capable of leading to the solution for BLM-Rank model. However, it suffers from the slow convergence problem because in each iteration we need to consider all training pairs so as to compute the gradient "direction" for updating $\mathbf{w}$, which is time consuming.

For the stochastic gradient method, we compute the gradient based on each training pair $(\mathbf{x}_{i,j},\mathbf{x}_{i,k})\in\mathcal{T}^+$ and then perform an immediate updating for parameter $\mathbf{w}$:

$$\mathbf{w} = \mathbf{w} + \alpha\cdot\left(\frac{-e^{\mathbf{w}^T(\mathbf{x}_{i,k}-\mathbf{x}_{i,j})}}{1+e^{\mathbf{w}^T(\mathbf{x}_{i,k}-\mathbf{x}_{i,j})}}(\mathbf{x}_{i,k}-\mathbf{x}_{i,j}) - \lambda\cdot\mathbf{w}\right) \quad (11)$$

In general, stochastic gradient method has a better convergence speed compared to full gradient method because it has fewer computations in each iteration of updating. In spite of this advantage, it needs to be pointed out that the order of training pairs to be traversed is crucial because a biased order may lead to poor convergence and solution. For example, if the training pairs are traversed in document-wise order, i.e., visiting the pairs related to the same document continuously, there are too many consecutive updates on the same document, which may introduce a bias when updating $\mathbf{w}$ and affect the convergence. To address this concern, we need to randomly shuffle the training pairs in $\mathcal{T}^+$ ahead of the computation.

The BLM-Rank training algorithm we proposed is based on stochastic gradient method, and it is summarized in Algorithm 1. After training with this algorihtm, we learn a linear function $F(\mathbf{x}_{i,j},\mathbf{w})$ to evaluate the relevance score of the j-th document with respect to the i-th query. In the testing phrase, this learned linear function is employed to predict the relevance scores of documents, and then documents

Algorithm 1 The BLM-Rank Training Algorithm

Input: a training data $\mathcal{T} = \left\{\left(\{\mathbf{x}_{i,j}\}_{j=1}^m, \mathbf{y}^{(i)}\right)\right\}_{i=1}^n$, parameters λ and α

Output: a linear function $F(\mathbf{x}_{i,j},\mathbf{w}) = \mathbf{w}^T\mathbf{x}_{i,j}$

Procedure:

1: Construct pairwise training samples and build $\mathcal{T}^+$ from $\mathcal{T}$;
2: Shuffle the training pairs in $\mathcal{T}^+$ such that the order to be traversed is unbiased and random;
3: Randomly initialize $\mathbf{w}$;
4: Set $\tilde{\mathbf{w}} = \mathbf{w}$;
5: For each sample $(\mathbf{x}_{i,j},\mathbf{x}_{i,k})\in\mathcal{T}^+$

$$\mathbf{w} = \mathbf{w} + \alpha\cdot\left(\frac{-e^{\mathbf{w}^T(\mathbf{x}_{i,k}-\mathbf{x}_{i,j})}}{1+e^{\mathbf{w}^T(\mathbf{x}_{i,k}-\mathbf{x}_{i,j})}}(\mathbf{x}_{i,k}-\mathbf{x}_{i,j}) - \lambda\cdot\mathbf{w}\right)$$

EndFor;
6: If $\|\tilde{\mathbf{w}} - \mathbf{w}\| > 0.0001$, goto Step 4; otherwise, return $\mathbf{w}$.

Algorithm 2 The BLM-Rank Testing Algorithm

Input: a set of documents $\{d_1, d_2, \cdots, d_m\}$ and their feature representations $\{\mathbf{z}_j = \phi(q, d_j)\}_{j=1}^{m}$ for a particular query q
Output: ranking results of documents $\{d_1, d_2, \cdots, m\}$
Procedure:

1: For $j = 1$ to m

$$s_j = F(\phi(q_i, d_j), \mathbf{w}) = \mathbf{w}^T \mathbf{z}_j$$

EndFor

2: Apply bitonic sort to rank m documents $\{d_1, d_2, \cdots, m\}$ in terms of their relevance scores s_j in descending order.

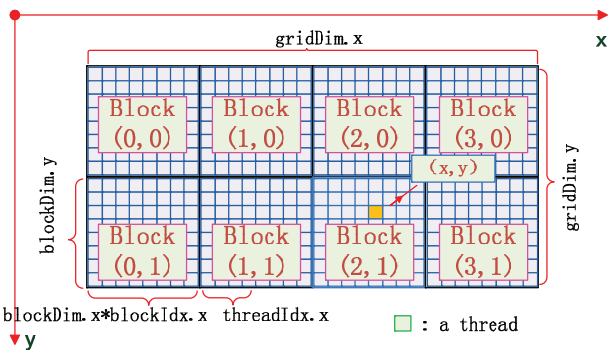


Fig. 1 The demonstration of the structure of GPU thread grid.

are ranked in terms of their relevance scores, as shown in Algorithm 2.

4. Parallel Implementation on GPUs

In recent years, big data have attracted increasing attentions of researchers from both industry and academia. Meanwhile, parallel computation with MapReduce or GPUs is becoming a hot research topic. In particular, ranking problems in real application usually face a huge number of samples and computations. For example, a webpage search engine needs to rank millions or billions of webpages in real time so as to answer a particular query. In this case, it is extremely important and useful to study the parallel implementation of the ranking algorithm. Therefore, for real-world applications, we present in this section the GPU implementation of the proposed BLM-Rank algorithm.

4.1 Scheme of Parallel Computing on GPU

The central idea of GPU parallel implementation is to decompose the computations into independent ones and then employs the grid thread of GPU to compute them in parallel. In a bird's view, the grid thread of GPU consists of two parts, i.e., the blocks to be paralleled and the threads within each block to be paralleled. Figure 1 gives an example to demonstrate this grid structure. We can see from this example there are 8 blocks and 8×8 threads in each block. In this grid, each small square represents a thread. Each thread is identified with the position (x, y) of the corresponding small square in this grid, which is computed as follows:

$$x = blockIdx.x \times blockDim.x + threadIdx.x \quad (12)$$

$$y = blockIdx.y \times blockDim.y + threadIdx.y \quad (13)$$

where $blockIdx.x$ and $blockIdx.y$ are the x-position and y-position of the block that current thread lies in respectively, $blockDim.x$ and $blockDim.y$ are the sizes of the block in x-dimension and y-dimension respectively, $threadIdx.x$ and $threadIdx.y$ are the x-position and y-position of current thread in this block. Moreover, this position (x, y) can be transferred into an offset in global to identify this thread from others:

$$offset = x + y \times gridDim.xWidth, \quad (14)$$

where $gridDim.xWidth$ is the number of threads in x-dimension in this grid. For example, the thread marked with yellow color is identified with the position (x, y), where

$$x = 2 \times 8 + 4,$$
$$y = 1 \times 8 + 2.$$

Also, we can compute its global offset:

$$offset = 20 + 10 \times 32 \quad (15)$$

where $gridDim.xWidth = 32$.

The GPU parallel computations are achieved by using its grid threads to perform decomposable computing in parallel. Conceptually, we need to define a GPU "kernel" function as follows:

```
_global_ void kernel()
{
    1. Calculate the offset for current thread;
    2. Perform computing related to current thread;
}
```

As this function will be executed in each thread, it needs to calculate the offset first in step 1 so that relevant thread can be identified and activated to perform computing in step 2. After declaration, this function can be called as follows:

$$kernel <<< BlockNum, ThreadNum >>> ();$$

where $BlockNum$ and $ThreadNum$ specify the number of blocks and threads for this function respectively.

4.2 Parallel BLM-Rank on GPU

Following the scheme of parallel computing on GPU, we introduce the detail to speed up the training and testing phase of BLM-Rank. In the training phase, the computational cost mainly comes from computing gradient for training pairs in $\mathcal{T}^+$, i.e., step 5 in Algorithm 1. In the implementation on GPU, the gradient is computed in parallel for all training pairs on GPU. The BLM-Rank training phase can be speeded up by executing this algorithm whenever the gradient is needed. In the testing phase, the computational cost

mainly comes from the computation of scores for testing documents, i.e., step 1 in Algorithm 2. In GPU implementation, these scores are calculated in parallel by using GPU. Therefore, calling this algorithm in step 1 of Algorithm 2 leads to a faster testing phase for BLM-Rank.

5. Experimental Results

In this section, we present two experimental results to demonstrate the performance of the proposed BLM-Rank method. In the first experiment, we use LETOR data sets to show the retrieval performance of the BLM-Rank method. For comparison, the results of RankSVM-Struct, RankBoost, AdaRank-NDCG, AdaRank-MAP and ListNet are given as well. In the second experiment, we test the GPU implementation of the BLM-Rank method and make a comparison with its CPU counterpart on algorithm efficiency. In all the experiments, the learning rate α is set to be 0.001. Table 1 shows the settings of the computer on which the experiments are performed.

5.1 Data Sets and Evaluation Metric

The data sets used in this paper are from LETOR provided by Microsoft Research Asia, which is a benchmark data collection for learning to rank [25]. In this data collection, there are currently five different versions, including LETOR 1.0, 2.0, 3.0, 4.0 and a new data set released recently. Each of the data sets of different versions has been divided into five-folds for cross-validation purpose.

We use the Normalized Discounted Cumulative Gain (NDCG) [26] as an evaluation metric to test the retrieval performance. In information retrieval, to emphasize the high-ranking relevant documents, the discounted cumulative gain DCG@k is defined as a measure to evaluate the effectiveness of ranking results. DCG@k discounts the contribution of low-ranking relevant documents. It is calculated by

$$DCG@k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{log_2(1+i)}$$

where rel_i is the relevance grade of the document ranked number i with respect to the query input. NDCG@k is a normalized version of this measure:

$$NDCG@k = \frac{DCG@k}{IDCG@k}$$

where IDCG@k refers to the ideal discounted cumulative gain at position k, which is computed by presenting an ideal ranking list. Since we have a set of queries, we report the average NDCG@k scores of all queries as final results.

Table 1 Settings of computer.

Components	Information
CPU	Intel(R) Core(TM) i7 3.40GHz
RAM	16GB
Disk	7200r/min 300MB/s DTR
GPU	NVIDIA GeForce GTX570
Operating system	Windows7 64bit

5.2 Experiment 1

In this experiment, we use two data sets from LETOR 4.0 and one from LETOR 3.0, namely, MQ2007, MQ2008 and OHSUMED to test the retrieval performance of the BLM-Rank method. As aforementioned, each of these data sets is divided into five folds. The pairwise training set $\mathcal{T}^+$ is then constructed through internal 5 fold-cross validation. The detailed descriptions of these data can be found in Table 2 (#(Docs) is the number of documents, #(Pairs) is the number of pairs in $\mathcal{T}^+$). In both MQ2007 and MQ2008, each document is represented as a 46-dimensional vector (while each document of OHSUMED is a 45-dimensional vector), i.e., $p = 46$.

5.2.1 Parameter Tuning Test

In order to find the optimal parameters for the BLM-Rank method, we test first how its performance changes with respect to the tuning of the regularization parameter λ. We tune the parameter λ from 0 to 1 and evaluate its performance with the average result of five-fold cross-validation. Figures 2 show how the performance of BLM-Rank changes against the change of parameter λ for three data, respectively. We observe from these figures that $\lambda = 0.5$ yields the best results for MQ2007 and MQ2008, and yields the relative better results for OHSUMED. Therefore, we use this value for λ in this experiment. For a more detailed demonstration, we show in Fig. 3 the five-fold results of BLM-Rank on three data sets when $\lambda = 0.5$.

5.2.2 Performance Comparison Test

In this subsection, we compare the performance of the proposed BLM-Rank method and some learning to ranking algorithms of state-of-the-art, including RankSVM-Struct, RankBoost, AdaRank-NDCG, AdaRank-MAP and ListNet. Figures 4 show the retrieval performance of comparison algorithms on MQ2007/2008 and OHSUMED data sets respectively. We observe from these figures that the performance of BLM-Rank algorithm is better than that of the other algorithms. In particular, there is a large improvement for BLM-Rank on MQ2008 compared to the other al-

Table 2 Descriptions of MQ2007, MQ2008 and OHSUMED.

MQ2007					
	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
#(Docs)	42,158	41,958	41,320	41,478	41,955
#(Pairs)	246,015	244,151	237,273	277,410	290,459
MQ2008					
	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
#(Docs)	9,630	9,404	8,643	8,514	9,442
#(Pairs)	52,325	46,631	44,450	54,729	63,736
OHSUMED					
	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
#(Docs)	9,219	10,187	10,494	9,491	9,096
#(Pairs)	367,663	422,716	378,087	297,131	285,703

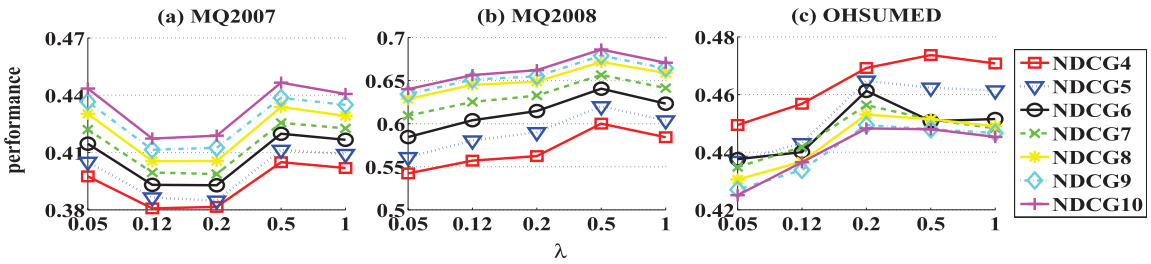


Fig. 2 The parameter tuning test. (a) Test on MQ2007; (b) Test on MQ2008; (c) Test on OHSUMED.

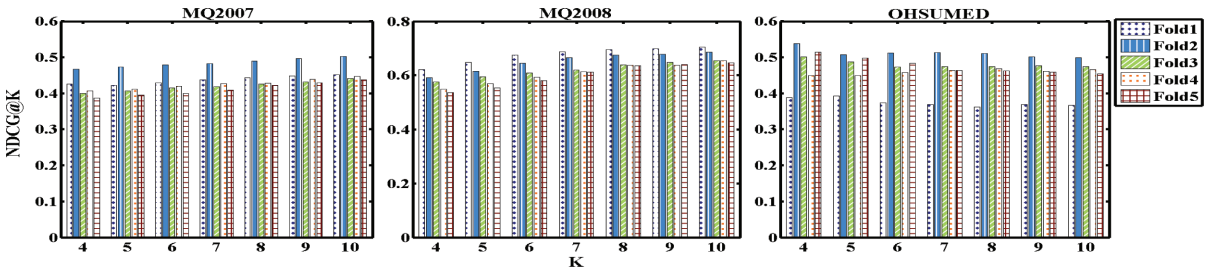


Fig. 3 The detailed results of BLM-Rank on five folds when $\lambda = 0.5$. Left: MQ2007; center: MQ2008; right: OHSUMED.

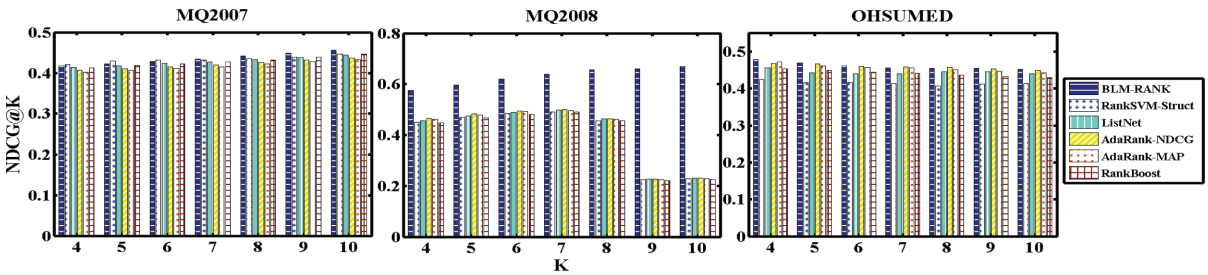


Fig. 4 The retrieval performance of the BLM-Rank method and other algorithms. Left: MQ2007; center: MQ2008; right: OHSUMED.

gorithms (the performance of baseline methods was cited from the LETOR websites[†]). These observations suggest that the proposed BLM-Rank method has a better generalization ability, which can be attributed to the simplicity of its model and the advantage of Bayesian parameter estimation.

Moreover, we find that the proposed method performs much better on MQ2008 than on MQ2007, although both datasets are produced from similar data sources. This is presumably due to the fact that MQ2007 has much more samples than MQ2008. As a result, the data distribution may be more complex for MQ2007 than MQ2008. Since our ranking function is a linear model, it is more suitable to handle simple data distribution and thus obtain strikingly better performance on MQ2008 than on MQ2007. However, we note that our model is a general ranking framework and one can incorporate Kernel function into the linear model to tackle datasets with more complex distributions.

[†]http://research.microsoft.com/en-us/um/beijing/projects/letor/

5.3 Experiment 2

In this experiment, we test the GPU implementation of the proposed BLM-Rank method and compare it with the CPU implementation in both training and testing phases to demonstrate its efficiency. As we aim to test the efficiency, two recently-released large data sets in LETOR are used, including MSLR-WEB10K and MSLR-WEB30K. These data sets are collected by the search engine Bing of Microsoft, and 136 features are selected by experts to represent each document. The relevances of documents are graded into five levels. Again, each of these data sets is divided into five folds, and pairwise training set $\mathcal{T}^+$ is constructed based on them. The detailed descriptions of both data sets are shown in Table 3. In this experiment, we set both *BlockNum* and *ThreadNum* to 512.

5.3.1 Speedup Test in Training Phase

First, we test how much speedup we can obtain by using GPU implementation in the training phase for BLM-Rank.

Table 6 The comparison of running time for GPU and CPU in the testing phase.

	$u = 1$						$u = 10$					
t-value	10^4	10^5	5×10^5	10^6	1.5×10^6	2×10^6	10^4	10^5	5×10^5	10^6	1.5×10^6	2×10^6
GPU(S)	0.062	0.132	0.388	0.724	1.095	1.306	0.164	0.750	3.295	6.542	10.356	13.011
CPU(S)	0.016	0.219	1.077	2.370	4.508	5.486	0.210	2.242	11.562	24.998	48.163	54.453
Speedup ratio	0.285	1.659	2.776	3.273	4.117	4.201	1.280	2.989	3.508	3.821	4.651	4.185
	$u = 50$						$u = 100$					
t-value	10^4	10^5	5×10^5	10^6	1.5×10^6	2×10^6	10^4	10^5	5×10^5	10^6	1.5×10^6	2×10^6
GPU(S)	0.540	3.478	16.006	32.416	51.613	66.020	1.147	6.504	32.517	65.228	100.086	130.568
CPU(S)	1.060	10.935	56.519	125.121	240.703	274.041	2.140	21.628	109.317	246.918	466.909	527.312
Speedup ratio	1.963	3.144	3.531	3.860	4.650	4.151	1.866	3.325	3.362	3.785	4.665	4.039

Table 3 Descriptions of MSLR-WEB10K and MSLR-WEB30K.

	MSLR-WEB10K		MSLR-WEB30K	
	# (Docs)	# (Pairs)	# (Docs)	# (Pairs)
Fold 1	235,259	8,853,518	747,218	14,934,127
Fold 2	241,521	9,327,539	753,611	14,960,363
Fold 3	241,988	9,226,117	756,149	14,959,521
Fold 4	239,093	9,021,995	753,394	14,952,642
Fold 5	242,331	9,331,327	760,753	15,308,771

Table 4 The comparison of running time for GPU and CPU on MSLR-WEB10K.

	CPU(S)	GPU(S)	Speedup ratio
Fold 1	626.500	57.125	10.967
Fold 2	620.856	58.602	10.594
Fold 3	623.405	58.372	10.679
Fold 4	651.985	59.921	10.880
Fold 5	706.341	60.305	11.712
Average	645.817	58.865	10.971

Table 5 The comparison of running time for GPU and CPU on MSLR-WEB30K.

	CPU(S)	GPU(S)	Speedup ratio
Fold 1	2949.037	272.879	10.807
Fold 2	3257.075	273.819	11.895
Fold 3	3080.138	272.875	11.287
Fold 4	3160.088	272.856	11.581
Fold 5	3257.075	274.806	11.852
Average	3140.683	273.447	11.485

We can see from Table 3 that there are millions (MSLR-WEB10K) or billions (MSLR-WEB30K) of training pairs, indicating a huge amount of arithmetic operations for BLM-Rank. Tables 4 and 5 give the running time of BLM-Rank with GPU and CPU implementations on both data, respectively. It can be seen from these tables that the GPU implementation is ten-to-eleven times faster than the CPU implementation for training. This result demonstrates the efficiency of GPU implementation of the BLM-Rank method.

5.3.2 Speedup Test in Testing Phase

In this subsection, we test how much we can improve by using GPU implementation of the BLM-Rank in the testing phase. We simulate the search engine scenario for this testing. In search engine, there are two varying factors that will affect its response time to a particular query. One is the number of relevant documents to be scored and ranked, and the other is the number of simultaneous queries started by users at the same time. Let t and u represent these two numbers respectively. We assume that the number of relevant documents t varies and takes values in $\{10000, 100000, 500000, 1000000, 1500000, 2000000\}$, and the number of simultaneous queries u takes values in $\{1, 10, 50, 100\}$. As a sequel, we consider the combinatorial cases of these two numbers and then make a comparison on running time between GPU implementation and CPU implementation. Table 6 shows the running time of the BLM-Rank for these cases. Clearly, we can see from this table that the speedup ratio increases from 1 to 4 or so as t increases from 10,000 to 2,000,000. This observation means that the GPU implementation of the BLM-Rank indeed improves compared to its CPU counterpart, and this improvement becomes large as the number of testing documents increases.

Due to the capacity limitation of GPU memory in our settings, the numbers both of training samples and testing samples are not large enough compared to real application. Therefore we obtain merely ten-to-eleven times and one-to-four times speedup ratios in the training and testing phase respectively. However, we claim that these ratios will be enlarged as the numbers of training and testing samples become larger.

6. Concluding Remarks

In this paper, we have proposed BLM-Rank, a Bayesian linear method to solve learning to rank problem. This method incorporates a linear function into a Bayesian framework to model the probability of each preference pair of training samples, and then employs Bayesian parameter estimation to fit this linear function. In the testing phase, this linear function is used to compute the relevance scores of test samples for ranking. Moreover, a GPU implementation of this method has also been given and studied.

We have conducted experiments on benchmark data sets from LETOR to test the performance of the proposed BLM-Rank method. Experimental results have demonstrated the BLM-Rank method indeed outperforms the state-of-the-art learning to rank methods in retrieval performance, including RankSVM-Struct, RankBoost, AdaRank-NDCG, AdaRank-MAP and ListNet. In addition, the results have shown the GPU implementation of the BLM-Rank method is ten-to-eleven times faster than its CPU implementation in

the training phase, and one-to-four times faster in the testing phase.

References

[1] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: bringing order to the web," 1999.
[2] J.M. Kleinberg, "Authoritative sources in a hyperlinked environment," Journal of the ACM (JACM), vol.46, no.5, pp.604–632, 1999.
[3] M.K.-P. Ng, X. Li, and Y. Ye, "Multirank: co-ranking for objects and relations in multi-relational data," Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, pp.1217–1225, ACM, 2011.
[4] X. Li, M.K. Ng, and Y. Ye, "Har: Hub, authority and relevance scores in multi-relational data for query search," SDM, pp.141–152, 2012.
[5] J.M. Ponte and W.B. Croft, "A language modeling approach to information retrieval," Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, pp.275–281, ACM, 1998.
[6] S.E. Robertson and K.S. Jones, "Relevance weighting of search terms," Journal of the American Society for Information science, vol.27, no.3, pp.129–146, 1976.
[7] S.E. Robertson and S. Walker, "Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval," Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval, pp.232–241, Springer-Verlag New York, 1994.
[8] D. Cossock and T. Zhang, "Subset ranking using regression," in Learning theory, pp.605–619, Springer, 2006.
[9] R. Herbrich, T. Graepel, and K. Obermayer, "Large margin rank boundaries for ordinal regression," Advances in Neural Information Processing Systems, pp.115–132, 1999.
[10] Y. Freund, R. Iyer, R.E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," Journal of Machine Learning Research, vol.4, pp.933–969, 2003.
[11] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," Proceedings of the 22nd international conference on Machine learning, pp.89–96, ACM, 2005.
[12] J. Xu and H. Li, "Adarank: a boosting algorithm for information retrieval," Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, pp.391–398, ACM, 2007.
[13] Z. Cao, T. Qin, T.Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," Proceedings of the 24th international conference on Machine learning, pp.129–136, ACM, 2007.
[14] K. Crammer, Y. Singer, et al., "Pranking with ranking," NIPS, pp.641–647, 2001.
[15] R. Caruana, S. Baluja, T. Mitchell, et al., "Using the future to "sort out" the present: Rankprop and multitask learning for medical risk evaluation," Advances in Neural Information Processing Systems, pp.959–965, 1996.
[16] C. Burges, "From ranknet to lambdarank to lambdamart: An overview," Learning, vol.11, pp.23–581, 2010.
[17] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma, "Frank: a ranking method with fidelity loss," Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, pp.383–390, ACM, 2007.
[18] I. Matveeva, C. Burges, T. Burkard, A. Laucius, and L. Wong, "High accuracy retrieval with multiple nested ranker," Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, pp.437–444, ACM, 2006.
[19] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "BPR: Bayesian personalized ranking from implicit feedback," Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, pp.452–461, AUAI Press, 2009.
[20] S. Rendle and L. Schmidt-Thieme, "Pairwise interaction tensor factorization for personalized tag recommendation," Proceedings of the third ACM international conference on Web search and data mining, pp.81–90, ACM, 2010.
[21] W. Pan and L. Chen, "GBPR: Group preference based bayesian personalized ranking for one-class collaborative filtering," Proceedings of the Twenty-Third international joint conference on Artificial Intelligence, pp.2691–2697, AAAI Press, 2013.
[22] A. Krohn-Grimberghe, L. Drumond, C. Freudenthaler, and L. Schmidt-Thieme, "Multi-relational matrix factorization using bayesian personalized ranking for social network data," Proceedings of the fifth ACM international conference on Web search and data mining, pp.173–182, ACM, 2012.
[23] L. Lerche and D. Jannach, "Using graded implicit feedback for bayesian personalized ranking," Proceedings of the 8th ACM Conference on Recommender systems, pp.353–356, ACM, 2014.
[24] T.Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li, "Letor: Benchmark dataset for research on learning to rank for information retrieval," Proceedings of SIGIR 2007 workshop on learning to rank for information retrieval, pp.3–10, 2007.
[25] T. Qin, T.-Y. Liu, J. Xu, and H. Li, "Letor: A benchmark collection for research on learning to rank for information retrieval," Information Retrieval, vol.13, no.4, pp.346–374, 2010.
[26] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of ir techniques," ACM Transactions on Information Systems (TOIS), vol.20, no.4, pp.422–446, 2002.



Huifeng Guo received the B.S. and M.S. degrees in Computer Science from Lanzhou University and Harbin Institute of Technology in China in 2012 and 2014, respectively. Since Sep, 2014, he has been at the School of Computer Science and Technology, Harbin Institute of Technology shenzhen Graduate School, P.R. China, where he is currently a first-year PhD student. His research interests are in the areas of data mining and clustering algorithm.



Dianhui Chu received his Ph.D. Degree from Department of Computer Science and Technology at Harbin Institute of Technology, He is now a professor in the Harbin Institute of Technology at Weihai. His research are within the domain of service computing, service engineering, and software architecture.



Yunming Ye received the Ph.D. degree in Computer Science from Shanghai Jiao Tong University. He is now a professor in the Shenzhen Graduate School, Harbin Institute of Technology. His research interests include data mining, text mining, and ensemble learning algorithms.



Xutao Li received the B.S. and M.S. degrees in Computer Science from Lanzhou University and Harbin Institute of Technology in China in 2007 and 2009, respectively, and the PhD degree in the Department of Computer Science, Harbin Institute of Technology. He is an associate professor in the Shenzhen Graduate School, Harbin Institute of Technology. His research interests include data mining, clustering algorithms, and machine learning.



Xixian Fan received M.S. degrees in Computer Science from Harbin Institute of Technology in China in 2013. She is with Baidu Inc