A Distributed Pairwise Learning

On Distributing Bayesian Personalized Ranking from Implicit Feedback

Modou Gueye

LID
Université Cheikh Anta Diop
Dakar
Sénégal
modou2.gueye@ucad.edu.sn

ABSTRACT. Pairwise learning is a popular technique for collaborative ranking with implicit, positive only feedback. Bayesian Personalized Ranking (BPR) was recently proposed for this task and its ranking is among the bests. Because its learning is based on stochastic gradient descent (SGD) with uniformly drawn pairs, it converges slowly especially in the case of a very large pool of items.
We propose an approach to distribute its computation in order to face its scalability issue.

RÉSUMÉ. Le classement par pairs d'objets est une approche populaire d'apprentissage pour la recommandation d'objets à un individu. On se base sur l'hypothèse que ce dernier s'intéresse plus à un objet qu'il pris qu'un autre qu'il n'a pas considéré. De cette hypothèse, un classement des objets selon les intérêts qu'il porterait sur eux peut-être appris.
Nous proposons dans ce papier, une nouvelle approche permettant de paralléliser l'apprentissage du classement et donc de réduire considérablement le temps de calcul.

KEYWORDS : Distribution, Bayesian pairwise learning, Matrix factorization

MOTS-CLÉS : Distribution, Classement par pair, Factorisation de matrice

1. Introduction

Collaborative ranking with implicit, positive only feedback (so called one-class collaborative filtering) aims to make personalized ranking by providing a user with a ranked list of items [4]. In this kind of application, the collected data from user actions/behaviors are in an *one-class* form like what they purchased, clicked on or listened. Such data are referred as "implicit feedback" of users [2]. Contrary to the explicit ones in rating prediction where users rate items, and therefore we directly know the preference relationship between users and some items, we have here to infer user preferences from implicit feedbacks. That say to say, we have to only consider the presence or not of some users' actions (e.g., purchases, clicks, or even search events) in order to rank items for a given user when making recommendations to it.

For more formalization, let us consider an online shop and its users' history of purchases $S \subseteq U \times I$ with U the set of all its users and I the one of items to sell. The task of the recommender system is here to provide the user u with a personalized total ranking $\succ_u \subset I^2$ of all items, where $\succ_u$ has to meet the properties of a total order [9].

Collaborative ranking has been steadily receiving more attention, mostly due to the "one-class" characteristics of collected data in various services (e.g., "bought" in Amazon, "like" in Yahoo!Music, and "clic" in Google Advertisement). Bayesian Personalized Ranking (BPR) was recently proposed for this task. It is a matrix factorization technique which is able to learn individual ranking from implicit data. BPR is also admitted as one of the best current RS for item recommendation [9, 5, 8, 6, 3]. It takes pairs of items as basic units and maximize the likelihood of pairwise preferences over observed items and unobserved items. However BPR uses stochastic gradient descent and converges slowly especially if the pool of items is very large.

In this paper, we present a new approach to face the scalability of BPR by distributing its computation. Our proposal can be adapted to both shared-memory configuration or fully-distributed one. In the sequel, we first present the underlying ideas of BPR and its generic algorithm, then we detail our method to parallelize it. Finally we show that our proposal reduces almost proportionally the execution time according to the degree of distribution.

2. Bayesian Personalized Ranking

The key idea of BPR is to use partial order of items to train a recommendation model, contrary to previous works which just considered single user-item examples [2, 4]. BPR introduces the interpretation of positive-only data as partial ordering of items. When we observe that a user u has selected an item i (e.g., user u purchases item i in an online shop), we assume that this user prefers this item than the others without observed feedbacks. Thus from this assumption, one can infer partial order of items for the user.
Figure 1 shows an example of inference. On the left side, we have a matrix of observations collected from user actions from which user specific pairwise preferences $i \succ_u j$ between pairs of items can be inferred. On the right side, we present pairwise preferences deduced from user u_1. The symbol plus (+) indicates that he prefers an item than another, while minus (-) says the contrary. For items that he has both seen, we cannot infer any preference.

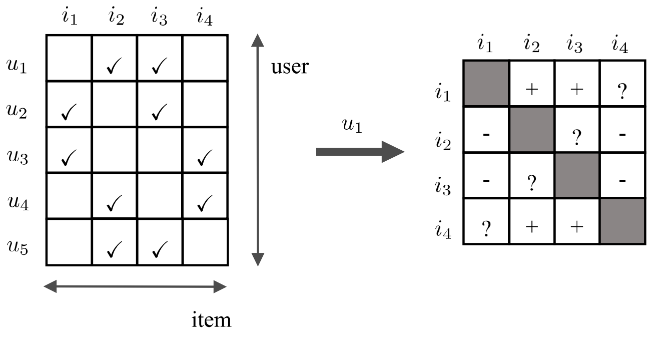


Figure 1 – Preferences retrieved from positive user-item occurrences

Let be $I_u^+ := \{i \in I : (u,i) \in S\}$ the set of implicitly-preferred items of user u. We can extract a pairwise preference dataset $\mathcal{P} : U \times I \times I$ by uniformly drawing for each user couples of an implicitly-preferred item and another one without observed feedback as follows

$$\mathcal{P} := \left\{(u,i,j) | i \in I_u^+ \wedge j \in I \backslash I_u^+\right\}$$

Each triplet $(u,i,j) \in \mathcal{P}$ implies that user u prefers item i than j. Due to the very large number of possible triplets, $\mathcal{P}$ is usually extracted by sampling techniques.

As BPR uses matrix factorization, it represents each user u (resp. each item i) by a vector p_u (resp. q_i) of latent factors. Thus, for each triplet $(u,i,j) \in \mathcal{P}$ we have the following order relation between the interests of u in i and j:

$$p_u \cdot q_i^T > p_u \cdot q_j^T, \ \ (u,i,j) \in \mathcal{P} \tag{1}$$

Hence, the main goal of BPR's optimization criterion ($BPR\text{-}OPT$) is to find an arbitrary model class to maximize the following posterior probability over all triplets in $\mathcal{P}$:

$$BPR\text{-}OPT = - \sum_{(u,i,j) \in \mathcal{P}} \ln \sigma(f_{uij}) + \lambda_\Theta \|\Theta\|^2 \tag{2}$$

For simplification, we posed f_{uij} as $p_u \cdot q_i^T - p_u \cdot q_j^T$. Θ represents the parameter vector of the arbitrary model class and λ_Θ the model specific regularization parameters. σ is the logistic sigmoid. The latter is used to approximate to non-differentiable Heaviside loss function [9]. Stochastic gradient descent (SGD) is used to learn the optimization criterion. In each step the gradient over the training data in $\mathcal{P}$ is computed and then the model parameters are updated with a learning rate α:

$$\Theta \leftarrow \Theta + \alpha \frac{\partial \, BPR\text{-}OPT}{\partial \Theta} \tag{3}$$

Algorithm 1 presents the learning of the optimization criterion with SGD.

Although BPR is among the best ranking technique, it converges slowly due to its sequential appraoch and pairs sampling, especially if the number of items is large [8]. Because that BPR relies on sampling pairs of items, its computation time grows relatively to the size of the pool of items to carry out.

In many large applications, we have to handle matrices with millions of both users and

```
Algorithm 1: Learning BPR
    Data: $\mathcal{P}$, $\Theta$
    Result: $\Theta$
1   Initialize $\Theta$;
2   repeat
3       Draw $(u, i, j)$ from $\mathcal{P}$;
4       $\Theta \leftarrow \Theta + \alpha\Big((1 - \sigma(f_{uij})) \cdot \frac{\partial f_{uij}}{\partial \Theta} + \lambda_\Theta \Theta\Big)$;
5   until convergence;
6   return $\Theta$
```

items, and so many entries [1]. At such scales, distributed algorithms for matrix factorization are essential to achieve reasonable performance as discussed in [1]. This make BPR not suited for web-scale applications. We propose below a way to do it by generalizing the Distributed SGD (DSGD) of Gemulla et al. [1]. In our knowledge, there is not currently any proposition on this topic in the literature. Of course, one may think to used DSGD-liked approaches as in [1, 7, 10]. Thus it can partition the matrix of observations as illustrated in the left side of Figure 1 into independent blocks as in Figure 2. Independent blocks constitute a stratum (in gray color). Therefore parallel learning may be done on each block, stratum-by-stratum. Although this idea seems fine and was well applied to rating prediction, that is not the case for preference ranking. Indeed here we do not consider couples of user-item (i.e., the user and an item that he rated) but triplets (u, i, j) where the first item is more preferred by the user than the second. Thus using DSGD limits each computation node to take the items i and j from only its current block. Therefore the user-specific rankings that one will make may have partial, and block-limited order. Indeed, in the pairwise preference dataset $\mathcal{P}$, any triplet u, i, j where i and j are in different blocks can not be considered.

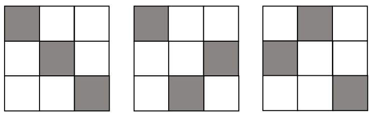


Figure 2 – Interchangeable blocks for a 3-by-3 gridded matrix

We propose a novel pair-blocks strategy which always keeps the notion of interchangeability of Gemulla et al. in [1]. They formulated it as two blocks which do not share any column nor row are interchangeable (i.e., independent). Thus two SGD instances can separately process them at the same time without any worrying. They define a stratum as a list of interchangeable blocks. The strata are processed in turn.

In the next section, we formalize and detail our proposal based on "pair-blocks" interchangeability. We show how we avoid partial, and block-limited order while ensuring distributed computing.

1. http://2016.recsyschallenge.com/

3. Distributed Bayesian Personalized Ranking

As we said above, block-based parallel gradient descent as introduced in [1, 7] is an original approach for distributing matrix factorization. Their well-minded concept of “interchangeability” underlies their contribution. We can define it as follows

Definition 1. *Blocks interchangeability*
Let be U_1 and U_2 to subsets of U, similarly I_1 and I_2 two subsets of I. Let be $\mathcal{B}_1 := U_1 \times I_1$ and $\mathcal{B}_2 := U_2 \times I_2$ two data blocks. They are interchangeable iff $U_1 \cap U_2 = \emptyset$ and $I_1 \cap I_2 = \emptyset$ (i.e., they do not share any row nor column).

From this definition, one can run operations completely in parallel on these blocks. Hence we introduce our notion of interchangeable pair-blocks, but for convenience, we define first our consideration of pair-block.

Definition 2. *Pair-block*
A pair-block φ is a couple of not interchangeable blocks $\mathcal{B}_1$ and $\mathcal{B}_2$ such as $U_1 = U_2$.

Definition 3. *Pair-blocks interchangeability*
Two pair-blocks are interchangeable if each block of the one is interchangeable with each block of the others.

Figure 3 shows two interchangeable pair-blocks represented with different colors. The matrix of observations can be expressed as unions of strata. Each stratum contains a group of interchangeable pair-blocks. In Figure 3, we list the sequence of strata that one have to set up when targeting two processors. As one can remark in this figure, in the two last

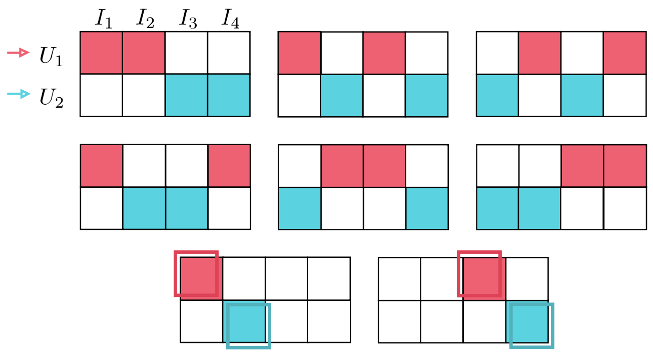


Figure 3 – Interchangeable pair-blocks-based strata

strata, the couple of blocks in pair-blocks has the same block. This allows us the possibility to consider any triplet $(u, i, j) \in \mathcal{P}$ wherever the position of i and j in the matrix of observation. As well as the number of block-columns is the double of the number of processors since each processor must have its own input. Therefore the sequence of strata must be carefully chosen in order to avoid re-using a pair-block in two different strata. Let be n the number of processors (e.g., two processors in Figure 3), each stratum must have its own n interchangeable pair-blocks to be processed in a distributed manner. From this point, it is easy to compute the number of strata to made since it becomes a combination problem. Indeed for n processors, the number N_s of strata to consider is the one of all 2-combinations of the $2n$ block-columns $C^2_{2n} + 2n$ in order to cover all the triplets in $\mathcal{P}$

given by $N_s = C_{2n}^2 + 2n = n(2n+1)$.
One consequence of the use of pair-blocks is that we are able to join all any two blocks of the same block-row. Contrary to DSGD, we can infer preference ranking for each user over all the items. Therefore our ranking is not partial or block-limited while we are able to process each of our pair-blocks-based stratum in parallel. The processors of computation are synchronized when starting learning on a stratum.
We called our approach of distributing bayesian personnalized ranking by DBPR. Algorithm 2 details its functioning. Lines 7 to 10 are the distributed part. In Line 2 the strata

Algorithm 2: Learning DBPR

```
   Data: $\mathcal{P}$, $\Theta$, $n$
   Result: $\Theta$
1  Initialize $\Theta$;
2  Generate strata $\mathcal{S}$;
3  // To balance workloads across the computing resources
4  Balance pair-blocks' data;
5  repeat
6      foreach $s \in \mathcal{S}$ do // We take the strata in turn
7          for $\varphi \in s$ do in parallel // Processing of pair-blocks $\varphi$
8              Draw $(u, i, j)$ from $\mathcal{P}_\varphi$;
9              $\Theta \leftarrow \Theta + \alpha\Big(\big(1 - \sigma(f_{uij})\big) \cdot \frac{\partial f_{uij}}{\partial \Theta} + \lambda_\Theta \Theta\Big)$;
10         end
11     end
12 until convergence;
13 return $\Theta$
```

are generated and their pair-blocks' data balanced in Line 4.

4. Experimentation

We demonstrate in this section the efficiency of our proposal. Due to the limited paper size and the closeness of DBPR and BPR ranking qualities (see Section 7.2), we compare here their learning times. We led a set of experiments with two publicly available datasets.

4.1. Datasets

Due to the lack of implicit feedback datasets, researchers usually rely to transforming rating datasets [9, 5]. Thus we evaluate our algorithm using two different rating datasets: MovieLens@1M and MovieLens@10M [2]. As we want to solve an implicit feedback task, we first take only the ratings with a value ≥ 4 (the range of ratings is from 1 to 5), then we generated user-item pairs by removing the rating scores. Thus we obtain implicit, positive only feedback datasets. Table 1 shows the final characteristics of the datasets.

2. http://www.grouplens.org/node/73

Table 1 – Characteristics of the datasets

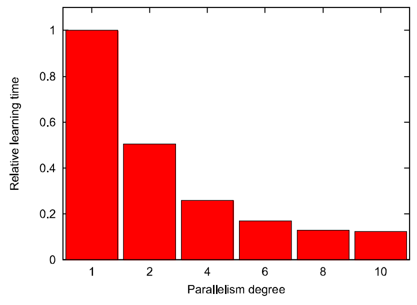
Dataset	$\lvert U\rvert$	$\lvert I\rvert$	$\lvert S\rvert$
MovieLens@1M	6,036	3,483	450,771
MovieLens@10M	56,071	10,119	4,010,795

4.2. Setup

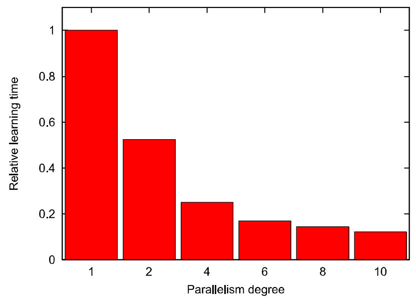
We implemented DBPR and BPR in C/C++ and used shared-memory processing. We generated all strata by backtracking. Then to balance the amount of data in the pair-blocks, we used a round-robin-based approach which permutes both users and items. Two indexes allow us to find the final position of a user or an item.
Our evaluation consisted to run DBPR with increasing degree of parallelism, and compares its computation time to the one of BPR [3]. Of course, we included in the final processing time of DBPR the one spent to generate strata and balance data between the blocks. We ran our experiments on a linux computer (Intel/Xeon with 24 cores at 2.93 GHz, and 64 GB of memory).

4.3. Learning time vs Parallelism degree

On each dataset, we launched one instance of BPR, and successively instances of DBPR with increasing degrees of parallelization. To ensure considering the same number of triplets per iteration for both BPR and DBPR, with compute the number of triplets per both iteration and pair-blocks as follows $N_\varphi = \frac{N}{n \times N_s}$, where N represent the number of triplets per iteration for BPR. With this consideration, we ensure that all our executions do the same amount of calculation. For each dataset, we drew $10 \times |S|$ triplets at each learning iteration. The number of factors per user and item is fixed to 10 and the total number of iteration to 200.
Figure 4 points out the contribution of DBPR on learning time relatively to the one of BPR. The latter equals 494 and 6,053 seconds for respectively the learning time on MovieLens@1M and MovieLens@10M. We can observe that the learning time decreases almost



(a) MovieLens@1M



(b) MovieLens@10M

Figure 4 – Relative learning time vs Parallelism degree

proportionally to the degree of parallelization thanks to the independence of pair-blocks in each stratum.

3. One can consider that an execution of BPR corresponds to the one of DBPR without parallelization

5. Conclusion

DBPR is a new proposal to improve the learning time of BPR-liked models. In our experimentation, we demonstrated its efficiency as it is able to nearly decrease the computation time proportionally to the degree of parallelization. Time reduction allows to learn BPR models from very large datasets by adapting our proposal to distributed framework like MapReduce.

Following the statement of the law of large numbers and the central limit theorem, one can expect a better ranking precision by increasing the size of the dataset $\mathcal{P}$ while ensuring moderated learning time with DBPR.

6. References

[1] Rainer Gemulla, Erik Nijkamp, Peter J. Haas, and Yannis Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 69–77, New York, NY, USA, 2011. ACM.

[2] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society.

[3] Lukas Lerche and Dietmar Jannach. Using graded implicit feedback for bayesian personalized ranking. In *Proceedings of the 8th ACM Conference on Recommender Systems*, RecSys '14, pages 353–356, New York, NY, USA, 2014. ACM.

[4] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N. Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 502–511, Washington, DC, USA, 2008. IEEE Computer Society.

[5] Weike Pan and Li Chen. Gbpr: Group preference based bayesian personalized ranking for one-class collaborative filtering. In Francesca Rossi, editor, *IJCAI*. IJCAI/AAAI, 2013.

[6] Shuang Qiu, Jian Cheng, Ting Yuan, Cong Leng, and Hanqing Lu. Item group based pairwise preference learning for personalized ranking. In *Proceedings of the 37th International ACM SIGIR Conference on Research &#38; Development in Information Retrieval*, SIGIR '14, pages 1219–1222, New York, NY, USA, 2014. ACM.

[7] Benjamin Recht and Christopher Recht. Parallel stochastic gradient algorithms for large-scale matrix completion. *Mathematical Programming Computation*, 5(2):201–226, 2013.

[8] Steffen Rendle and Christoph Freudenthaler. Improving pairwise learning for item recommendation from implicit feedback. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 273–282, New York, NY, USA, 2014. ACM.

[9] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 452–461, Arlington, Virginia, United States, 2009. AUAI Press.

[10] Yong Zhuang, Wei-Sheng Chin, Yu-Chin Juan, and Chih-Jen Lin. A fast parallel sgd for matrix factorization in shared memory systems. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 249–256, New York, NY, USA, 2013. ACM.

7. Annexes

7.1. Biographie

M. Gueye hold a PhD degree from Telecom ParisTech, a leading French engineering school specialized in computer science, under the supervision of Pr Talel Abdessalem (Telecom ParisTech) and Dr Hubert Naacke (University Pierre & Marie Curie, France). His thesis' subject was about designing scalable and accurate recommender systems.

M. Gueye is currently an Assistant Professor at University Cheikh Anta Diop (Sénégal). His research interests are in large scale data management and mining, recommender systems and web information extraction.

7.2. performance of DBPR in terms of ranking

Due to the limited size of the paper, we report here the performance of DBPR in terms of quality measures commonly employed in the recommendation field. For the performance evaluation, we used the Precision, Recall, F1 and NDCG measures which are references in this field.

Tables 2 and 3 show the ranking qualities of BPR and some instances of DBPR with increasing parallelism degree (2, 4 and 8 degree).

In almost all the measures, we see that the ranking quality of DBPR is close enough to the one of BPR. The slight lost of quality when the parallelism degree increases can be related to the pair-blocks-based learning of DBPR. Indeed, each processor unit is constrained to sample triplets into its current pair-blocks. Although this ensures independant processing, but we can not sample so much different triplets as BPR allows. We target to face this drawback in our future work. Indeed with the decreasing of computation time thanks to the distributed approach of DBPR, we can increase the number of triples to use in each iteration in order to expect better ranking in recommendations.

Table 2 – top@5 comparison of DBPR and BPR on MovieLens@1M

Algorithm	Recall	Precision	F1	NDCG
BPR	0.1057	0.3997	0.1671	0.0782
DBPR-2	0.1032	0.39	0.1632	0.0765
DBPR-4	0.0997	0.3886	0.1586	0.0732
DBPR-8	0.0972	0.3865	0.1553	0.0718

Table 3 – top@10 comparison of DBPR and BPR on MovieLens@1M

Algorithm	Recall	Precision	F1	NDCG
BPR	0.1766	0.3573	0.2363	0.1095
DBPR-2	0.1743	0.3502	0.2327	0.1061
DBPR-4	0.1726	0.3513	0.2314	0.1023
DBPR-8	0.1688	0.3481	0.2273	0.0994