Towards Distributed Pairwise Ranking using Implicit Feedback

Mohsan Jameel
University of Hildesheim
mohsan.jameel@ismll.de

Nicolas Schilling
University of Hildesheim
schilling@ismll.de

Lars Schmidt-Thieme
University of Hildesheim
schmidt-thieme@ismll.de

ABSTRACT

Learning with pairwise ranking methods for implicit feedback datasets has shown promising results as compared to pointwise ranking methods for recommendation tasks. However, there is limited effort in scaling the pairwise ranking methods in a large scale distributed setting. In this paper we address the scalability aspect of a pairwise ranking method using Factorization Machines in distributed settings. Our proposed method is based on a block partitioning of the model parameters so that each distributed worker runs stochastic gradient updates on an independent block. We developed a dynamic block creation and exchange strategy by utilizing the frequency of occurrence of a feature in the local training data of a worker. Empirical evidence on publicly available benchmark datasets indicates that the proposed method scales better than the static block based methods and outperforms competing state-of-the-art methods.

KEYWORDS

Recommender Systems, Distributed Pairwise Ranking, Implicit Feedback

ACM Reference Format:
Mohsan Jameel, Nicolas Schilling, and Lars Schmidt-Thieme. 2018. Towards Distributed Pairwise Ranking using Implicit Feedback. In *SIGIR '18: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, July 8–12, 2018, Ann Arbor, MI, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3209978.3210113

1 INTRODUCTION

Personalized recommender systems are pivotal in enhancing customer's online shopping experiences, due to their ability to make personalized recommendations. These recommender systems learn user behavior from past observations, which are captured either through explicit user feedback, i.e. ratings or implicitly from user interaction with the system, i.e. the user purchase history, movies watched etc. In real world systems, it is usually inexpensive to capture implicit feedback. The recommender systems generate a ranked list of targets i, for each personalized request represented through a given context c. The ranking function $\mathcal{R}(i|\mathbf{c}) \rightarrow \mathbb{N}^+$, defined in (1), generates an ordered ranking of the targets for each context $\mathbf{c} \in C$, where $\hat{y}(i|\mathbf{c}, \Theta)$ is a model with model parameters Θ.

SIGIR '18, July 8–12, 2018, Ann Arbor, MI, USA

ACM ISBN 978-1-4503-5657-2/18/07…$15.00
https://doi.org/10.1145/3209978.3210113

$$\mathcal{R}(i|\mathbf{c}) = |\{j \mid \hat{y}(j|\mathbf{c}, \Theta) \geq \hat{y}(i|\mathbf{c}, \Theta)\}| \quad (1)$$

The pairwise ranking algorithms [10] have shown to outperform pointwise ranking algorithms [5, 6] on the ranking task. The pairwise methods are well suited for implicit feedback as they learn a pairwise loss over a set of observed positive feedback and a set of unobserved feedback. However, scalability studies of the pairwise methods for implicit feedback are limited, and generally focused on shared memory systems [11].

In this paper we address the scalability of a pairwise ranking algorithm for a large scale dataset. Specifically, we investigate the viability of existing distributed Stochastic Gradient Optimization (SGO) algorithms [3, 5], which were originally designed for a pointwise loss and are mostly limited to a single relational Matrix Factorization (MF). These methods block partition the rating matrix and create static blocks of model parameters. However, the implicit feedback dataset only contains observed positive examples and unobserved examples are sampled. The static block partitioning of model parameters limits the sample space of unobserved examples, which can therefore introduce a bias in the gradient updates. We present a dynamic block partitioning and exchange strategy for model parameters by utilizing the information about the frequency of occurrence of features in each local data partition. We demonstrate the applicability of our algorithm by using a generic framework based on Factorization Machines (FM) [9] to solve a pairwise ranking problem including multiple entity relationships.

2 PERSONALIZED PAIRWISE RANKING FROM IMPLICIT FEEDBACK

2.1 Pairwise Ranking from Implicit Feedback

A context may consist of a single entity like a user, or multiple entities, i.e. a combination of a user and a resource, and can also have additional relations. Suppose there are D entities in a context, the context set is defined as $C = \prod_{l=1}^{D} \mathcal{X}^l$, where $\mathcal{X}^l$ is the set of entities l. A set of targets $\mathcal{T}$, for example movies watched or tags used for annotation and the historical observations are captured in a set $\mathcal{S} \subset C \times \mathcal{T}$. [1] Given training data $\mathcal{D}_s \leftarrow \{(\mathbf{c}, i^+, i^-) \mid (\mathbf{c}, i^+) \in \mathcal{S} \wedge (\mathbf{c}, i^-) \notin \mathcal{S}\}$, a pairwise ranking optimization problem is defined as,

$$\arg\min_{\Theta} \sum_{(c, i^+, i^-) \in \mathcal{D}_s} \mathcal{L}(i^+ >_c i^-, \Theta) + \lambda_\Theta \|\Theta\|_2^2 \quad (2)$$

where $i^+ >_c i^-$ defines a pairwise ranking, i.e. target i^+ is preferred over i^- for a given context $\mathbf{c}$, and $\lambda_\Theta \in \mathbb{R}^+$ is the regularization parameter for the L2 regularization of the model parameters. There are many choices for the pairwise loss function $\mathcal{L}(i^+ >_c i^-, \Theta)$, but for this paper we used the Bayesian personalized ranking (BPR)

[1] For example, in a tag recommender, the observation set $\mathcal{S}$ can be defined as $\mathcal{S} \subseteq \mathcal{X}^1 \times \mathcal{X}^2 \times \mathcal{T}$, where $\mathcal{X}^1$ is a set of all users, $\mathcal{X}^2$ is a set of all items and $\mathcal{T}$ is a set of all tags.

[10] loss in (3), which is designed using a Bayesian modeling of D_s and has shown strong empirical results [10, 12].

$$\mathcal{L}(i^+ >_c i^-, \Theta) = -\ln\left(\sigma(\hat{y}(\mathbf{c}, i^+, \Theta) - \hat{y}(\mathbf{c}, i^-, \Theta))\right) \quad (3)$$

BPR optimization can be applied to any recommender model. However, factorization models are the most popular among these models. Factorization Machines (FM) [9] present a generic approach towards solving recommender tasks and can mimic most of the factorization models through feature engineering, which is why we will focus on them. Let $\mathbf{x} \in \mathbb{R}^M$ be the feature vector, then the second order expansion of FMs is given as,

$$\hat{y}(\mathbf{x}) = w_0 + \langle \mathbf{w}, \mathbf{x} \rangle + \sum_{l=1}^{M} \sum_{j=l+1}^{M} \langle \mathbf{v}_l, \mathbf{v}_j \rangle x_l x_j \quad (4)$$

where $K \in \mathbb{N}^+$ is the hyper-parameter defining the dimensionality of factorization, $\Theta = \{w_o, \mathbf{w}, \mathbf{V}\}$ are the model parameters where $w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^M$, $\mathbf{V} \in \mathbb{R}^{M \times K}$, and $\langle \cdot, \cdot \rangle$ is the dot/scalar product of two vectors. The second term in (4) captures all two-way interactions between input and factorizes the interaction weight. The feature vector $\mathbf{x}$ for personalized ranking is represented through binary indicators as.

$$\mathbf{x} = (\overbrace{\underbrace{0, \ldots 0, 1, 0, \ldots 0}_{|\mathcal{X}^1|}, \ldots, \underbrace{0, \ldots 0, 1, 0, \ldots 0}_{|\mathcal{X}^D|}}^{C}, \underbrace{0, \ldots 0, 1, 0, \ldots 0}_{|\mathcal{T}|}) \quad (5)$$

3 DISTRIBUTED PAIRWISE RANKING (DPR) OPTIMIZATION

The pairwise ranking algorithm optimizes a ranking loss between pairs of observed and unobserved interactions. A simple way to learn in a distributed setting is by using parallel stochastic gradient descent [8]. The training data is randomly partitioned among workers. Each worker learns a separate model on each individual data partition and a master node averages these models after each pass over the data. The communication cost is high as each worker communicates a copy of the model parameters with a master worker. A better way is to exploit parallelism in the problem structure, such that every distributed worker updates an independent and disjoint set of model parameters. Many training instances in a sparse dataset are orthogonal, i.e. $\langle \mathbf{x}_i, \mathbf{x}_j \rangle = 0$, and can be scheduled in parallel without any overlapping updates to the model parameters. But finding all non-conflicting training pairs is computationally not feasible. We use simple heuristics based on the problem structure to overcome the computational complexity and apply it to a generic Factorization Model i.e Factorization Machines (FM). The main design consideration was building a dynamic block partitioning scheme, which is driven by the frequency of occurrences of model parameters on a given worker. Our approach differs from existing approaches [3, 13] in two aspects, a) they target a pointwise method, and b) create blocks through a static partitioning scheme.

3.1 Data Local Parallel

In distributed pointwise techniques, the observation set $\mathcal{S}$ is randomly sharded into P disjoint blocks i.e. $\mathcal{S}_{p=1,\ldots,P}$. This technique is not feasible for a pairwise loss optimization. Firstly, the unobserved instances are inferred from the observed set. Therefore the observed set for a given context must be available in the same data partition $\mathcal{S}_p$. Secondly, using a smart partitioning we can achieve partial parallelism for a subset of the model parameters. Therefore, we propose to partition the observation set on one of the context entities $\mathcal{X}^u$ with the highest number of features i.e. $u = \arg\max_{u' \in \{1,\ldots,D\}} |\mathcal{X}^{u'}|$. We used the famous 'bin packing' [7] algorithm to divide the observation dataset among workers. First, we calculated the number of observations for each context, then we assigned all the contexts with an element $a \in \mathcal{X}^u$ to a worker such that the total number of observations at each worker is approximately equal. This ensures that all the observations for a context is available to one worker only. The cost of partitioning using 'bin packing' is proportional to random sharding. Since the observation set at each worker is disjoint in $\mathcal{X}^u$, the model parameters corresponding to the entity $\mathcal{X}^u$ are not required to be exchanged among workers. Thus we achieved a partial parallelism in $\mathcal{X}^u$.

3.2 Model Local Parallel

The second level of parallelism is achieved by partitioning the model parameters. The data partitioning on entity $\mathcal{X}^u$ means we require $O(|\mathcal{X}^u|)$ less communication. The remaining entities of the context group are divided into P disjoint subsets $\mathcal{X}_i^v \cap \mathcal{X}_j^v = , \forall i, j \in \{1, \ldots, P\} | i \neq j$, where $\mathcal{X}_i^v \subset \mathcal{X}^v$ and $\mathcal{X}_j^v \subset \mathcal{X}^v$ and $v \in \{1, \ldots, D\} \setminus \{u\}$. Similarly the target entity is also divided into P subsets $\mathcal{T}_1, \mathcal{T}_2 \cdots, \mathcal{T}_P$. The partitioning of model parameters into subsets allows each individual worker to work on a disjoint set of training examples i.e. $\mathcal{X}_p^u \times \mathcal{X}_p^v \times \mathcal{T}_p \subset \mathcal{S}_p$. During the learning process the model parameters corresponding to entities $\mathcal{X}^v$ and $\mathcal{T}$ need to be exchanged between workers. It is important to create these partitions intelligently so that each worker receives those model parameters for which it has training examples.

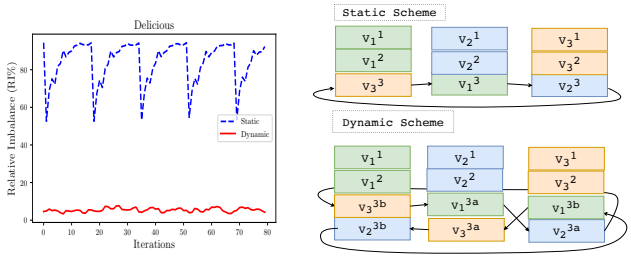

Figure 1: (a) Relative percentage of Imbalance (RI%) updates across 16 blocks. A lower value of RI% shows better work balance among workers. (b) The illustration of static and dynamic partitioning and exchange scheme with an example using the three entities and three workers.

3.2.1 Static Block Partitioning. The existing methods [3] create these subsets through the index division, i.e. let $b = |\mathcal{X}^v|$ then the indices of the subset $\mathcal{X}_p^v$ can be calculated as $p(b/P)$ to $(p+1)(b/P)$. These static blocks are than cyclically exchanged among workers and require P^{D-1} rounds to complete an epoch. There are two major short falls in this strategy. Firstly, although the model parameters are evenly divided, the number of updates per block are not. We tracked the number of updates per block U_p in each iteration and define the relative percentage of imbalance as $(U_{max} - U_{min})/U_{sum}$,

where U_{max}, U_{min}, and U_{sum} are maximum, minimum and sum across P blocks respectively. Fig(1) shows a high percentage of imbalance updates across blocks for the static scheme. Secondly, the static partitioning of the target entity restricts the sample space to a small subset, i.e. $|\mathcal{T}_p| \ll |\mathcal{T}|$.

3.2.2 Dynamic Block Partitioning. A more intuitive way is to utilize the information about how likely a parameter will be updated at the target worker. The frequency $\mathcal{F}$ of occurrence of a feature i in the training set S_p, is calculated as $\mathcal{F}_i(p) = \sum_{\mathbf{x} \in S_p} I(x_i \neq 0)$ (I is an indicator function that returns one if $x_i \neq 0$, and zero otherwise) provides a better guess about the likelihood of a parameter update. We propose to use the empirical distribution over this frequency to select the destination worker for a model parameter. Once the blocks are created they are communicated to the target workers. We employ this strategy every time a worker needs to communicate his current share of the model parameters with other workers. Fig (1) shows a comparison of the percentage of imbalance between the static and dynamic scheme. It can be seen that the percentage of imbalance is greatly reduced using the dynamic scheme. Each worker uniformly distributes the model parameters of a target entity $\mathcal{T}$ to the other workers, which helps in increasing the sample space of unobserved examples. The dynamic blocks of model parameters are created at every communication round as illustrated with an example in Fig (1). A complete overview of our Distributed Pairwise Ranking (DPR) algorithm is listed in Algorithm (1), where *Exchange* represents exchange strategies explained above.

Algorithm 1 Distributed Pairwise Ranking (DPR) Algorithm

```
w_0 ← 0; w ← (0, ..., 0); V ~ N(0, ρ) ▷ where Θ = {w_0, w, V}
▷ distributed workers P run in parallel
repeat
    for round ∈ {1, ..., D} do
        for (c, i+, i−) ∈ X_p^u × X_p^v × T_p × T_p ⊂ D_s^p do
            Δ ← (1 − σ(ŷ(c, i+, Θ) − ŷ(c, i−, Θ)))
            θ ← θ − ηΔ ∂/∂θ (ŷ(c, i+, Θ) − ŷ(c, i−, Θ))
        Exchange(Θ_p, F, round)
until stopping criterion is met
Gather all model parameters Θ from P workers
```

4 EXPERIMENTS

4.1 Datasets and Evaluation

In this section, we empirically evaluate the proposed algorithm DPR on five publicly available recommender systems benchmark datasets listed in Table (1). The Delicious dataset is an implicit feedback dataset, it contains tagging activities of users for resources. The movielens, netflix and Yahoo datasets are originally 5-star rating based datasets with entities being a user and a movie. They are preprocessed into implicit feedback dataset by retaining tuples with rating values ≥ 3. The Delicious dataset is preprocessed to 5 cores (i.e. each user, item and tag appears at least in 5 posts), the others are preprocessed to 10 cores. We did a hyperparameter search in grids, i.e $K = \{2^4, \cdots, 2^7\}$, $\lambda = \{10^{-5}, \cdots, 10^{-1}\}$, $\eta = \{10^{-5}, \cdots, 10^{-1}\}$ and $\alpha = \{10^{-2}, \cdots, 10\}$. A test set is created by removing a single post for each context from the training set. We have used average Area under the ROC curve (AUC) given in [10] for a comparison of the prediction quality of these methods[2]. DPR and PSGD [14] are implemented in C++ using the Message Passing Interface (MPI) [3].We also compared against the state-of-the-art pointwise ranking method WR-MF [5], which is based on a single relational Matrix Factorization and is implemented in Apache Spark [4]. To illustrate different partitioning schemes presented in Section 3, we denote DPR-dynamic for a dynamic block partitioning scheme and DPR-static for a static block partitioning scheme. A cluster of 10 nodes is used for the evaluation purposes, where each node has an Intel Xeon E5620 2.40GHz and 24 GB RAM, connected through a Gigabit interconnect.

Table 1: The statistics and parameter for each dataset

	ml10m	ml20m	netflix[1]	yahoo[2]	Delicious[4]
$\lvert\mathcal{X}^1\rvert$	7.1×10^4	1.3×10^5	4.8×10^5	1×10^6	5.3×10^5
$\lvert\mathcal{X}^2\rvert$	1.0×10^4	2.7×10^4	1.7×10^4	6.2×10^5	1.7×10^6
$\lvert\mathcal{X}^3\rvert$	-	-	-	-	2.4×10^6
$\lvert\mathcal{S}\rvert$	1×10^7	2×10^7	9.9×10^7	2.6×10^8	1.4×10^8

4.2 Comparison of Convergence

In the first set of experiments in Fig (2), the convergence speed of DPR, WR-MF, and PSGD were compared on the Yahoo, netflix and movielens datasets. The graph presents the relative difference to the maximum AUC achieved among the competing models vs time (seconds) in log-scale. On all the datasets, DPR-dynamic converges fastest to the best AUC values of 0.993 and 0.974 on the Yahoo and netflix datasets respectively, and clearly show superiority over DPR-static. WR-MF took longer to reach the same AUC value on Yahoo and movielens datasets, but it attains a lower AUC (0.968) on netflix. The DPR-static was slowest to converge to similar AUC values as DPR-dynamic and WR-MF. The DPR-dynamic achieved faster convergence because of the dynamic blocking scheme, which presents more diverse sample pairs as compare to DPR-static and speeds up learning. PSGD on the other hand did not converge to a better result on these datasets in a reasonable time.

The second set of experiments were conducted on the Delicious dataset, which is originally an implicit feedback dataset and contains three entities. WR-MF is not applicable in this settings as it is limited to the datasets containing relationships between two entities. We compared DPR-dynamic with DPR-static and PSGD. DPR-dynamic converges fastest to the best AUC value of 0.992, whereas DPR-static converges to an AUC of 0.990 and took considerably more time. PSGD again did not converge to a good value of AUC in a reasonable time. Overall, we observed that DPR-dynamic works better than DPR-static.

4.3 Scalability of DPR

In this section we present a scalability analysis of the DPR (DPR-dynamic) algorithm. Fig (3) presents the learning curves by varying the number of workers. The x-axis represents a log-scale of the CPU time (in seconds) expanded by multiplying with the number of workers. The y-axis represents relative difference to the maximum AUC. A linear scalability in the number of workers is achieved, if

[2]For a detailed comparison of BPR and WR-MF on different evaluation measures in a serial setting, we direct our readers to [10, 12].
[3]MPICH version 3.2, https://www.mpich.org/.
[4]Spark 2.2.0, https://spark.apache.org/docs/2.2.0/mllib-collaborative-filtering.html

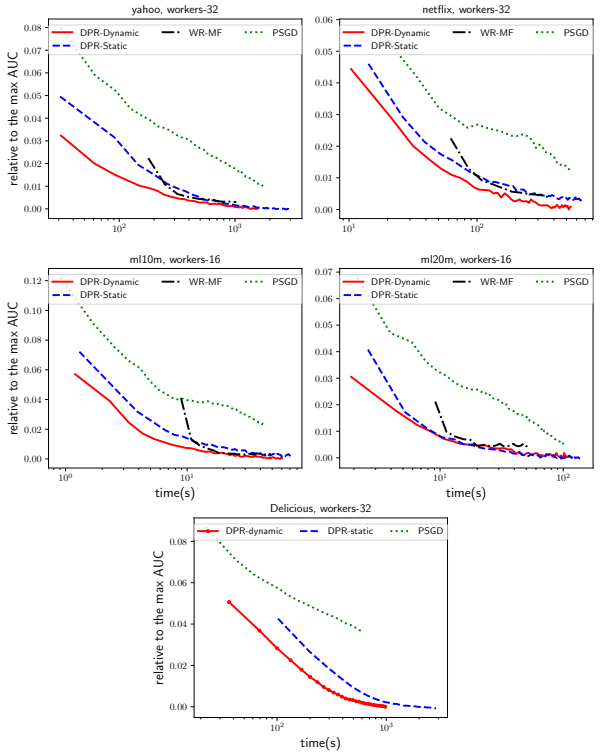

Figure 2: The convergence of different methods on recommender datasets. The y-axis is the relative difference to the maximum AUC value. The x-axis, represents time(sec) in log-scale.

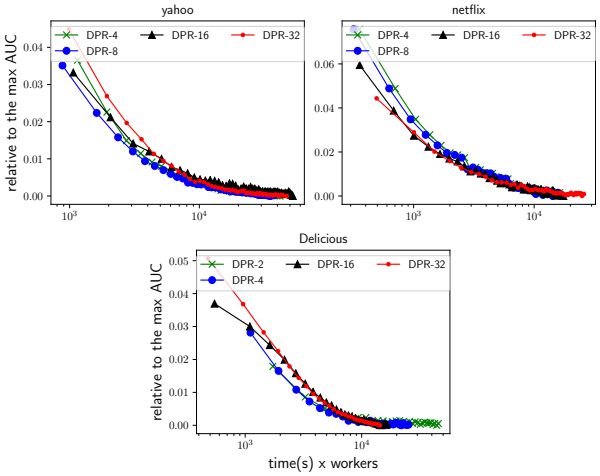

Figure 3: The scalability analysis of DPR (DPR-dynamic) on varying number of workers. The y-axis is the relative difference to the maximum AUC value. The x-axis, represents time(sec) expanded by multiplying by the number of workers in log-scale.

the learning curves recorded by running on a varying number of workers overlap. All the learning curves overlap, which shows the linear scaling of the learning curves across the number of workers.

5 CONCLUSION

In this paper we study the scalability of a pairwise ranking algorithm in distributed settings. We investigate the applicability of distributed SGD techniques for a pointwise method on a pairwise method. The static block partitioning scheme employed by pointwise methods was not useful for pairwise methods. We developed a dynamic block partitioning of model parameters and experimentally show that it works better than the static scheme. The experiments show that DPR outperforms WR-MR, which is a distributed algorithm that optimizes a pointwise loss.

As a future work, it would be interesting to investigate the dynamic partitioning scheme for the exchange of parameters for other model classes as well as in an asynchronous distributed algorithm. This would require overlapping exchange and update steps of the DPR algorithm.

REFERENCES

[1] Robert M. Bell and Yehuda Koren. 2007. Lessons from the Netflix prize challenge. *SIGKDD Explorations* 9, 2 (2007), 75–79.

[2] Gideon Dror, Noam Koenigstein, Yehuda Koren, and Markus Weimer. 2012. The Yahoo! Music Dataset and KDD-Cup '11. In *Proceedings of KDD Cup 2011 competition, San Diego, CA, USA, 2011.* 8–18.

[3] Rainer Gemulla, Erik Nijkamp, Peter J. Haas, and Yannis Sismanis. 2011. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011.* 69–77.

[4] Olaf Görlitz, Sergej Sizov, and Steffen Staab. 2008. PINTS: peer-to-peer infrastructure for tagging systems. In *Proceedings of the 7th international conference on Peer-to-peer systems, IPTPS'08, Tampa, FL, USA, February 25-26, 2008.* 19.

[5] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on.* Ieee, 263–272.

[6] Noam Koenigstein and Ulrich Paquet. 2013. Xbox movies recommendations: variational bayes matrix factorization with embedded feature selection. In *Seventh ACM Conference on Recommender Systems, RecSys '13, Hong Kong, China, October 12-16, 2013.* 129–136.

[7] John Levine and Frederick Ducatelle. 2004. Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society* 55, 7 (2004), 705–716.

[8] Mu Li, Li Zhou, Zichao Yang, Aaron Li, Fei Xia, David G Andersen, and Alexander Smola. 2013. Parameter server for distributed machine learning. In *Big Learning NIPS Workshop*, Vol. 1.

[9] Steffen Rendle. 2012. Factorization Machines with libFM. *ACM TIST* 3, 3 (2012), 57:1–57:22.

[10] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI '09, Montreal, Quebec, Canada.* 452–461.

[11] A. Murat Yagci, Tevfik Aytekin, and Fikret S. Gürgen. 2017. On Parallelizing SGD for Pairwise Learning to Rank in Collaborative Filtering Recommender Systems. In *Proceedings of the Eleventh ACM Conference on Recommender Systems, RecSys 2017, Como, Italy, August 27-31, 2017.* 37–41.

[12] Lu Yu, Ge Zhou, Chu-Xu Zhang, Junming Huang, Chuang Liu, and Zi-Ke Zhang. 2016. RankMBPR: Rank-Aware Mutual Bayesian Personalized Ranking for Item Recommendation. In *Web-Age Information Management - 17th International Conference, WAIM 2016, Nanchang, China, June 3-5, 2016, Proceedings, Part I.* 244–256.

[13] Erheng Zhong, Yue Shi, Nathan Liu, and Suju Rajan. 2016. Scaling Factorization Machines with Parameter Server. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (CIKM '16).* ACM, New York, NY, USA, 1583–1592. DOI:https://doi.org/10.1145/2983323.2983364

[14] Martin Zinkevich, Markus Weimer, Alexander J. Smola, and Lihong Li. 2010. Parallelized Stochastic Gradient Descent. In *24th Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada.* 2595–2603.