

Revisiting BPR: A Replicability Study of a Common Recommender System Baseline

Aleksandr Milogradskii
 alex.milogradsky@gmail.com
 National Research University Higher School of Economics, T-Bank Russia

Oleg Lashinin
 fotol764@gmail.com
 Moscow Institute of Physics and Technology, T-Bank Russia

Alexander P
 alexander.p.89@yandex.ru
 Independent Researcher Russia

Marina Ananyeva
 ananyeva.me@gmail.com
 National Research University Higher School of Economics, T-Bank Russia

Sergey Kolesnikov
 scitator@gmail.com
 T-Bank Russia

ABSTRACT

Bayesian Personalized Ranking (BPR), a collaborative filtering approach based on matrix factorization, frequently serves as a benchmark for recommender systems research. However, numerous studies often overlook the nuances of BPR implementation, claiming that it performs worse than newly proposed methods across various tasks. In this paper, we thoroughly examine the features of the BPR model, indicating their impact on its performance, and investigate open-source BPR implementations. Our analysis reveals inconsistencies between these implementations and the original BPR paper, leading to a significant decrease in performance of up to 50% for specific implementations. Furthermore, through extensive experiments on real-world datasets under modern evaluation settings, we demonstrate that with proper tuning of its hyperparameters, the BPR model can achieve performance levels close to state-of-the-art methods on the top-n recommendation tasks and even outperform them on specific datasets. Specifically, on the Million Song Dataset, the BPR model with hyperparameters tuning statistically significantly outperforms Mult-VAE by 10% in NDCG@100 with binary relevance function.

ACM Reference Format:

Aleksandr Milogradskii, Oleg Lashinin, Alexander P, Marina Ananyeva, and Sergey Kolesnikov. 2024. Revisiting BPR: A Replicability Study of a Common Recommender System Baseline. In *18th ACM Conference on Recommender Systems (RecSys '24), October 14–18, 2024, Bari, Italy*. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3640457.3688073>

1 INTRODUCTION

The issue of information overload, coupled with the rise of online services, has created a growing need for recommender systems [42]. These systems have attracted significant interest from both

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

RecSys '24, October 14–18, 2024, Bari, Italy

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM. ACM ISBN 979-8-4007-0505-2/24/10...\$15.00
<https://doi.org/10.1145/3640457.3688073>

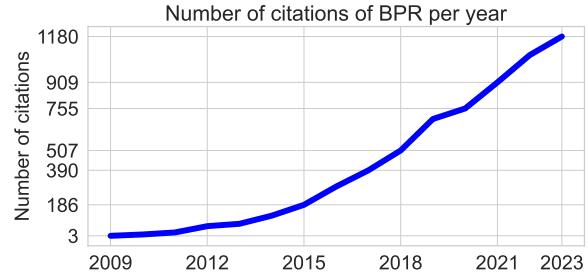


Figure 1: Number of citations of BPR per year according to Google Scholar, as of May 15, 2024.

academic [1, 28, 61, 64] and industrial researchers [12, 27, 37], who have studied various methods associated with personalization.

Traditionally, recommender systems have relied on either explicit [7] or implicit [26] feedback. Explicit actions, such as user ratings, are relatively limited. In contrast, implicit feedback, such as views, clicks, and purchases, is vast in number, making it easier to build recommender systems based on these logs [49]. However, these models are less confident [26].

The advancements in simple linear models and matrix factorization techniques have played a crucial role in the development of recommender systems, a subfield of machine learning. In recent years, however, deep learning (DL) methods [35] have become dominant in other areas of machine learning, such as natural language processing (NLP) [10], computer vision (CV) [59], and reinforcement learning (RL) [31]. Given these advances, one might expect similar improvements in recommender systems. Contrarily, several studies on recommender systems suggest that linear models are more effective than DL models for top-n recommendation [15, 36, 53]. These linear methods are easier and faster to train than many deep learning models. Additionally, many neural network models that claim superiority in specific tasks over simple linear models often face reproducibility issues, as outlined in [23, 47].

An excellent example of a linear model, proven successful in many recommendation systems tasks, is the Bayesian Personalised Ranking (BPR) model [49]. The authors introduced pairwise ranking loss in matrix factorization, arguing that it is better suited for item recommendation tasks than pointwise loss functions, such as

quadratic regression loss functions [26]. The idea of a pairwise target in BPR has attracted significant academic attention, evidenced by the increasing number of BPR paper citations, as highlighted in Figure 1. With 6624 citations as of May 15, 2024, the BPR paper is one of the most cited in the field of recommender systems.

Two notable factors affect the citation rate of the BPR paper and its popularity. *First*, BPR is a highly extendable model. Many researchers have built upon it with additional ideas, such as GBPR [45], VBPR [20], and CPLR [39]. In contrast, others have modified it to address issues like popularity bias [13] and item fairness [11]. *Second*, numerous studies utilize the BPR loss as an objective function for other model architectures. Examples of such modifications include LightGCN [21], GRU4Rec [25], FPMC [50], which incorporate different structures such as graphs, recurrent neural networks, and Markov chains, respectively.

The widespread popularity of BPR has led to numerous third-party implementations in well-known open-source frameworks such as Implicit [17], Cornac [52], LightFM [34], RecBole [66], and Elliot [4], despite the availability of the trustworthy implementation in MyMediaLite [18] library, provided by the authors of the BPR paper [49]¹. Unfortunately, the original implementation is implemented in C#, which hinders its adoption and integration with Python-based algorithms, which are dominant in the recommender systems field. In addition, it does not support GPU acceleration, making it slower than versions that benefit from it. These factors have contributed to the accumulation of the BPR third-party implementations.

However, recent reproducibility studies [23, 47] reveal that many third-party implementations for linear and deep learning models lack essential features, complicating comparisons with the original implementations. Thus, despite its widespread adoption, the BPR model might face similar challenges. Moreover, careful tuning of hyperparameters in well-established matrix factorization-based models has been shown to achieve performance near state-of-the-art (SOTA) methods [15, 51]. This aspect further emphasizes the importance of a thorough evaluation of BPR, as many recent papers report a subpar performance of this model in several evaluation settings.

Despite recent reviews of several popular models [15, 23, 47, 51], a comprehensive reproducibility study of the BPR model has yet to be conducted. The literature employing BPR as a baseline often omits a detailed description of its implementation, with key features such as sampling methods, learnable item biases, optimizer selection, and separate regularization factors missing from many open-source frameworks. These factors could impact BPR's performance, necessitating further investigation into their effects. Furthermore, the current evaluation standard for top-n recommendations relies on a global timeline method with ranking quality metrics [30], diverging from the approach outlined in the BPR paper [49]. These observations emphasize the demand for a reassessment of BPR to gain deeper insights into its performance and the factors shaping it.

To address this, we present an extensive set of experiments examining various features of the BPR model, open-source implementations, and datasets. *Our contributions are threefold.* **First**, we

thoroughly investigate open-source implementations of BPR to assess their consistency with the original implementation, revealing differences in the sets of features included. **Second**, we analyze the importance of each component of BPR on two real-world datasets, offering insights into the model's behavior and performance. **Third**, through careful hyperparameters tuning, we show that BPR can achieve better results than anticipated by existing implementations, even surpassing SOTA methods in some cases. Particularly, on the Million Song Dataset, the BPR model statistically significantly outperforms Mult-VAE by 10% in NDCG@100. The source code of the experiments can be found on our GitHub repository².

2 BACKGROUND

Top-n Recommendations. Collaborative filtering is a common approach for generating top-n recommendations based on implicit or explicit feedback data [55]. This method often relies on user-item interaction data and employs various learning methodologies, including rule-based approaches [38], heuristics [29], and neighborhood methods [41, 60]. The Netflix Prize competition [7] notably facilitated the development of matrix factorization techniques for recommender systems based on user behavior, leading to the introduction of new methods such as Implicit ALS (iALS) [26], SVD++ [62].

Since then, the task of top-n recommendation has seen widespread applications in implicit feedback scenarios. Novel methods continue to emerge, leveraging diverse methods such as linear methods [44, 54], variational autoencoders [37], graph convolutional networks [21], diffusion models [63]. Many studies associated with these approaches have demonstrated improvements over previous methods. Notably, the Bayesian Personalized Ranking (BPR) [49], often employed as a matrix factorization-based baseline, is frequently reported to be surpassed by these models [21, 37, 63].

Interestingly, recent studies have reported that simple linear models can outperform complex architectures. For instance, one study [15] evaluated several deep learning methods against well-tuned baselines, finding that many linear models, similar to BPR, serve as competitive baselines to deep learning models. Another recent study found that the linear model EASE outperformed numerous state-of-the-art methods across several datasets [3, 53]. Thus, linear methods continue demonstrating robust performance on tasks with implicit feedback despite the latest advancements in deep learning.

Reproducibility Studies. The study of Dacrema et al. [15] has led to a widespread discussion of reproducibility, creating a new research track that focuses on revisiting previously known methods. This area has gained considerable recognition, with numerous papers submitted each year. For example, the author of the popular sequential model GRU4Rec reviewed various open-source implementations, revealing that third-party implementations often lack certain features and contain bugs, impacting performance [23]. Another comprehensive study [47] revisited BERT4Rec [56], where authors found that various implementations yielded different quality metrics. Similarly, a study on iALS [26] reproducibility reported that minor changes to the original implementation greatly improved quality, emphasizing the importance of hyperparameters tuning [51]. However, to our knowledge, studies have yet to re-examine BPR despite being a highly cited model and often used as a baseline.

¹We find this implementation of BPR to be the most credible one because the original authors are involved in its development.

²<https://github.com/nemexur/revisit-bpr>

The development of evaluation protocols has also highlighted issues with data leakage [16, 30, 43]. Recent studies suggest that global temporal split is one of the most reliable ways of conducting offline experiments [43], and an evaluation protocol without it might lead to incorrect conclusions [30]. Consequently, assessing BPR in a global temporal split evaluation setup is necessary against the most advanced top-n recommendation methods.

3 BAYESIAN PERSONALIZED RANKING

3.1 Matrix Factorization

Matrix Factorization (MF) models operate by factoring the user-item rating matrix $R : U \times I$, where U is the set of all users and I is the set of all items. They associate each user and item with distinct sets of features, represented as real-valued vectors or latent factors. Each latent factor, denoted as p_u for users and q_i for items, comprises an f -dimensional vector \mathbb{R}^f . To estimate the interaction r_{ui} within the matrix R , MF computes an inner product $\hat{r}(u, i) = p_u q_i^T$ between the corresponding latent factors.

$$\underset{P, Q}{\operatorname{argmin}} \sum_{(u, i) \in R} L(\hat{r}(u, i), r_{ui}) \quad (1)$$

The model parameters of MF are the embedding matrices P and Q , which are learned by minimizing the objective function L (1).

Objective functions to learn MF model parameters can be categorized into three groups: pointwise, pairwise, and listwise [40].

3.1.1 Pointwise. The pointwise approach estimates the discrepancy between the predicted score $\hat{r}(u, i)$ and the actual score r_{ui} . It may be represented in the form of a squared loss objective:

$$\underset{P, Q}{\operatorname{argmin}} \sum_{(u, i) \in R} (r_{ui} - \hat{r}(u, i))^2 + \Omega(P, Q) \quad (2)$$

Here, the second term, Ω , penalizes the complexity of the model, often represented as an L2 regularization. The main limitation of pointwise approaches is the demand for a direct item relevance estimation instead of the item comparison for a user, which is a crucial aspect in recommendation tasks.

3.1.2 Pairwise. The pairwise objective function aims to optimize the relative ranking of items for a user, in contrast to predicting individual interactions in pointwise approaches, which addresses one of the main limitations of the previous approach. The main idea behind it is to correctly order pairs of items within the context of a user, achieved by comparing items in the user history $I^+(u) = \{i : (u, i) \in R\}$ with the remaining items $j \in I \setminus I^+(u)$. The objective function (4) then penalizes the model if the item j is ranked higher than the selected item from the user history using the function ϕ .

$$(u, i, j) \in D_R : \Leftrightarrow i \in I^+(u) \wedge j \in I \setminus I^+(u) \quad (3)$$

$$\underset{P, Q}{\operatorname{argmin}} \sum_{(u, i, j) \in D_R} \phi(\hat{r}(u, i) - \hat{r}(u, j)) + \Omega(P, Q) \quad (4)$$

3.1.3 Listwise. The listwise group of functions looks at the whole ranked list of items for each user $\mathbf{r}_u = [r_{u1}, \dots, r_{uI}]$ and optimizes it based on the ranking metric. One significant advantage of the listwise approach over other methods is its ability to distinguish the position of an item within the ranked list. It aids the model in understanding that items at the top of the list are more important

than those at the bottom, which is lacking in the pairwise approach. However, these objective functions (5) are more computationally extensive due to the direct optimization of the ranking quality of the entire recommendation list for a user $\hat{\mathbf{r}}_u = [\hat{r}(u, 1), \dots, \hat{r}(u, I)]$.

$$\underset{P, Q}{\operatorname{argmin}} \sum_{u \in U} L(\hat{\mathbf{r}}_u, \mathbf{r}_u) + \Omega(P, Q) \quad (5)$$

3.2 Bayesian Personalized Ranking Features

Bayesian Personalized Ranking (BPR) [49] is a popular version of the matrix factorization model with a pairwise objective function. What sets BPR apart from other pairwise models is its objective function. The goal of the function is to minimize the log-likelihood of the correct ordering in the form of:

$$\underset{P, Q}{\operatorname{argmin}} \sum_{(u, i, j) \in D_R} -\log \sigma(\hat{r}(u, i) - \hat{r}(u, j)) \quad (6)$$

During the training phase, it receives the tuple of a user u , a positive item i , and a negative item j and then computes the objective (6) using the function $\hat{r}(\cdot)$. While $\hat{r}(\cdot)$ frequently utilizes a Matrix Factorization model, it can be represented by any algorithm that calculates the score between a user and an item, such as KNN algorithms and Weighted MF utilized in [49]. Although the choice of the scoring function impacts the model's performance, we adhere to the matrix factorization version for our experiments. Additionally, five more features of BPR might influence its behavior:

Regularization. Authors of the original BPR paper [49] introduced three separate regularization parameters: user regularization, positive item regularization, and negative item regularization in the form of $\Omega(P, Q) = \sum_{(u, i, j) \in D_R} \lambda_u p_u^2 + \lambda_i q_i^2 + \lambda_j q_j^2$, where λ are regularization terms for specific embeddings.

Optimizer. The original implementation utilizes standard Stochastic Gradient Descent (SGD) for training. Hence, it is reasonable to expect potential performance improvements by integrating the latest optimization techniques, such as Momentum SGD [57], RMSProp [58], and Adam [33].

Negative Sampling. In [48], the authors of BPR observed that uniform sampling is not the best choice of negative sampling algorithm for the model. As the training progresses, it starts to yield *easy* negatives that result in minimal model parameter changes. To address this issue, they introduced the *adaptive sampling* algorithm for negatives, which respects the model's performance at each iteration. Although this sampling algorithm is not a part of the original paper [49], we still consider it a necessary addition to the algorithm that should be included in the list.

Item Bias. Many popular model architectures introduce item biases to the model as a prominent feature that frequently improves the performance [56, 65]. In the case of MF, the addition of item biases changes the inner product to $\hat{r}(u, i) = b_i + p_u q_i^T$. Furthermore, as we will show in Section 4.1.4, these biases exist in many popular open-source BPR implementations. Notably, the original implementation in MyMediaLite³ [18] also includes learnable item biases.

User Bias. User biases $\hat{r}(u, i) = b_u + p_u q_i^T$, like item biases, might also impact the model's performance. However, we have excluded them from the experiments because they are uncommon in open-source BPR implementations.

³<https://github.com/zenogantner/MyMediaLite>

Table 1: Statistics for the complete training datasets after preprocessing. The median per user/item is the middle of the distribution of the number of actions for users or items.

Dataset	Users	Items	Actions	Sparsity	Med. User/Item
Netflix ⁴	9949	4825	563577	0,9883	27/12
ML-20M	136677	20108	9,7M	0,9965	37/16
MSD	571355	41140	32,5M	0,9986	39/383
Yelp (time-split)	252616	92089	2,2M	0,9999	5/8
ML-20M (time-split)	124377	12936	8,9M	0,9944	38/57

⁴the subsample of the dataset similar to the original paper [49].

4 EXPERIMENTS

We conduct the experiments to address the following research questions:

- RQ1** How do the results achieved using open-source implementations compare with the originally reported results?
- RQ2** How do the BPR model’s features help improve performance?
- RQ3** How does the fine-tuned BPR compare to state-of-the-art models in the top-n recommendations?

4.1 Experimental Setup

4.1.1 Datasets. Similar to [51], we experiment using the benchmarks established in [37]. Furthermore, we include evaluation protocols with a global time-based split proposed in [24]. The experiments are conducted over four publicly available datasets: MovieLens-20M (**ML-20M**) [19], Million Song Dataset (**MSD**) [9], Netflix [7], and Yelp [6]. The characteristics of the datasets are summarized in Table 1.

4.1.2 Preprocessing and Evaluation Protocol. The evaluation methodology adopted in this study utilizes three methods, depending on the described research question.

RQ1. We employ the preprocessing and evaluation protocol from the BPR paper [49] on the Netflix dataset for this RQ. In this paper, the authors subsampled the dataset after preprocessing. We follow the same strategy and sample the dataset with a similar number of interactions: 563000 samples compared to 565000 samples in [49].

RQ2. We opt for ML-20M and MSD to conduct experiments for this RQ. We follow the same preprocessing and evaluation procedures from [37, 54]. For detailed steps and description, refer to [37] and their code⁵. However, we slightly modify it by adding held-out users to the training part, as the BPR model cannot handle cold users. The validation and testing parts remain unchanged.

RQ3. In addition to the datasets used in RQ2, we conduct experiments using a global temporal split on ML-20M and Yelp datasets, following [24, 30]. To preprocess these datasets, we first convert all numeric ratings or the presence of an interaction into implicit feedback. Afterward, we keep users and items with at least three interactions to ensure the dataset’s quality, following [22, 32, 50]. Regarding evaluation protocol, we set the testing time window to three years and the validation time window to one year for both datasets. Additionally, we filter out users absent from training and complete training parts in validation and testing on each dataset, respectively.

⁵https://github.com/dawenl/vae_cf

Table 2: BPR implementations. Source of the implementation: from the ACM Recommender Systems List of Frameworks (★), from the GitHub search (◇). Availability of features: available (✓), missing (✗). Original - MyMediaLite implementation.

	GitHub Stars ⁶	Commits ⁷	Features								
			Item Biases	Regularization			Optimizer	Negative Sampling			
				User λ_u	Positive λ_i	Negative $\lambda_{i'}$		Shared λ	SGD	Others ⁸	
Cornac*	820	116	✓	✗	✗	✗	✓	✓	✗	✓	✗
DaisyRec*	55	31	✗	✗	✗	✗	✓	✓	✓	✓	✗
Elliot*	265	1	✓	✓	✓	✓	✗	✓	✗	✓	✗
Rebole*	3,200	174	✗	✗	✗	✗	✓	✓	✓	✓	✗
ReChorus*	492	5	✗	✗	✗	✗	✓	✓	✓	✓	✗
RecPack*	6	70	✗	✓	✓	✗	✗	✗	✓	✓	✗
Implicit [◇]	3,400	27	✓	✗	✗	✗	✓	✓	✗	✗ ⁹	✗
LightFM [◇]	4,600	15	✓	✓	✓	✗	✗	✗	✓	✓	✗
Original	498	0	✓	✓	✓	✓	✗	✓	✓	✓	✗
Ours			✓	✓	✓	✓	✓	✓	✓	✓	✓

⁶As of May 15, 2024. ⁷From January 1, 2023, to May 15, 2024. ⁸Adam, Adagrad, Momentum SGD, RMSProp. ⁹Implicit utilizes popularity-based negative sampling.

4.1.3 Metrics. We employ two ranking-based metrics: *Recall@K* (R) and *NDCG@K* (N) with binary relevance function. Following [51], for each user, we remove all items from the predictions that the user has already interacted with in the training dataset.

4.1.4 BPR Implementations. To identify open-source implementations of BPR, we looked through the list of recommender frameworks by ACM Recommender Systems Conference¹⁰ and the list of popular recommender systems libraries on GitHub not present in that list. In Table 2, one can find frameworks with the BPR model and which features they implement. Notably, Microsoft Recommenders¹¹ is absent from this list because the authors of this repository use Cornac [52] to implement the BPR model.

Our selection criteria for experiments prioritize open-source implementations with good quality and support, assessed through the number of GitHub stars and consistent maintenance efforts. Specifically, we combine the Top-5 implementations based on GitHub stars with the Top-5 based on the number of commits. Additionally, we include MyMediaLite, regarded as the original implementation of BPR used for experiments in [49], and Elliot, as it closely follows the original BPR model from [49], as observed in Table 2. Based on these criteria, we select the following implementations: Cornac, Elliot, Implicit, LightFM, Rebole, and MyMediaLite (**Original**).

Additionally, it is evident from Table 2 that only Elliot [4] implements the complete set of features of the original model. Most other implementations use shared regularization factors rather than separate ones for users, positive and negative items. Moreover, all implementations, including the original, lack adaptive negative sampling from [48], which is a crucial feature, as we will demonstrate later in Section 5.2. Interestingly, many implementations introduce item biases as an additional feature, diverging from the original paper [49].

Moreover, as we will show in Section 5.1, Cornac performs best among third-party BPR implementations. However, it has a notable

¹⁰<https://github.com/ACMRecSys/recsys-evaluation-frameworks>

¹¹<https://github.com/recommenders-team/recommenders>

limitation: Cornac utilizes Cython, which only supports CPU execution, resulting in longer training time than GPU alternatives. Moreover, we could not add adaptive negative sampling to Cornac’s BPR model due to Cython constraints with sampling algorithms. Therefore, we decided to replicate BPR with PyTorch [46], which supports GPU, allowing us to incorporate all discussed features completely.

4.1.5 Baselines. In addition, we compare BPR implementations to baselines that have demonstrated state-of-the-art results in recent reproducibility papers on the selected datasets:

- **ItemPop.** A non-personalized model that ranks items based on their popularity. This baseline is used for reference.
- **EASE** [54]. An item-item collaborative filtering model with a closed-form solution. A state-of-the-art model on MSD [53].
- **Mult-VAE** [37]. An extension of variational autoencoders for collaborative filtering with implicit feedback using a multinomial likelihood objective. In addition, we include **Mult-DAE** as this version might outperform Mult-VAE in the most active users, which comprises the biggest part of any dataset.

4.1.6 Hyperparameters Search. Hyperparameters are optimized on a dedicated training/validation split created from the complete training set using the same process as the train/test split. Subsequently, the models are retrained on the complete training dataset with the best hyperparameters and evaluated on the testing dataset. The optimization procedure uses a popular framework for hyperparameters optimization Optuna [2] on all datasets using the sampler based on the Tree-structured Parzen Estimator (TPE) [8] algorithm. We search for the best hyperparameters using NDCG@100 on all datasets except Netflix, which utilizes the AUC metric.

Regarding model-specific hyperparameters, we adjust the L2-norm regularization parameter for the EASE model [54]. In the case of the Mult-VAE and Mult-DAE models, we vary the number of epochs and the learning rate while keeping the batch size consistent with the original paper [37]. The considered hyperparameter ranges and distributions are available on our GitHub repository.

4.1.7 Embedding Dimension. Matrix factorization models rely on the embedding dimension of user and item representations. It controls the capacity of the model. Models with high capacity are more likely to overfit the training set, whereas models with low capacity may underfit it. To explore the impact of embedding dimensions following [51], we conduct the experiments with varying dimensions: 32, 64, 128, 256, 512, and 1024 for ML-20M and MSD datasets, 16, 32, 64, and 128 for Netflix dataset, and 64, 128, 256, and 512 for ML-20M time-splitted and Yelp. We run the full hyperparameters search with fixed embedding dimension to properly asses the impact on the BPR algorithm. This approach allows us to identify the optimal embedding dimension for each dataset, which we then utilize in subsequent experiments on the dataset.

5 EXPERIMENTAL RESULTS

5.1 RQ1. Replicability

First, we analyze whether the selected open-source implementations can replicate the results of MyMediaLite BPR implementation (**Original**), a trustworthy version of BPR from the original authors, in a setup as close as possible to the original BPR paper. Figure 2

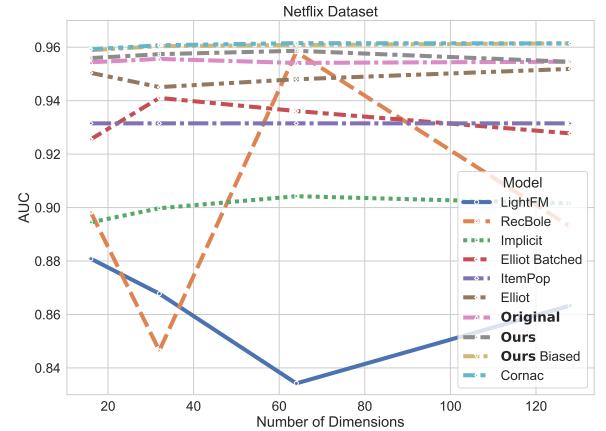


Figure 2: Area Under the ROC Curve (AUC) prediction quality for the Netflix dataset using various open-source BPR implementations. The ItemPop model is included to compare performance against a non-personalized baseline.

compares the implementations with the default configuration and hyperparameters search over epochs, regularization, and learning rate for each implementation. As shown in Figure 2, our implementation, Cornac’s, and Elliot’s implementations are close to the original results. Conversely, other open-source implementations exhibit inferior results compared to other models. Interestingly, Recbole and LightFM demonstrate unstable performance levels regarding the AUC metric in Figure 2. We assume this behavior is related to the adaptive optimization algorithms used in these implementations. Specifically, Adam [33] in RecBole and Adagrad [14] in LightFM. We observe a similar pattern in the experiments for Section 5.2, where we employ adaptive optimizers. We will provide an explanation of these findings in the subsequent sections.

Additionally, we check two implementations of our model: one incorporating item biases and one without them. Notably, on the Netflix dataset, we observe that item biases are only helpful at the start of training, but their utility diminishes over time. Eventually, non-biased models reach the same performance levels as biased models, being less than 1% worse than their counterparts.

Moreover, we assess two implementations of BPR in Elliot framework: BPRMF¹² and Batched BPRMF¹³. The former closely follows the BPR model from [49], while the latter is an optimized variant that diverges from the original BPR model. Specifically, Batched BPR is similar to LightFM but utilizes SGD instead of Adagrad optimizer. Therefore, we suppose these changes result in a 3% drop in performance compared to the BPRMF variant from Elliot.

5.2 RQ2. Influence of BPR features

In Table 2, we outline the availability of BPR features in each open-source BPR implementation. In order to demonstrate the effect of

¹²https://github.com/sisinfab/elliot/blob/v0.3.1/elliot/recommender/latent_factor_models/BPRMF/BPRMF_model.py

¹³https://github.com/sisinfab/elliot/blob/v0.3.1/elliot/recommender/latent_factor_models/BPRMF_batch/BPRMF_batch_model.py

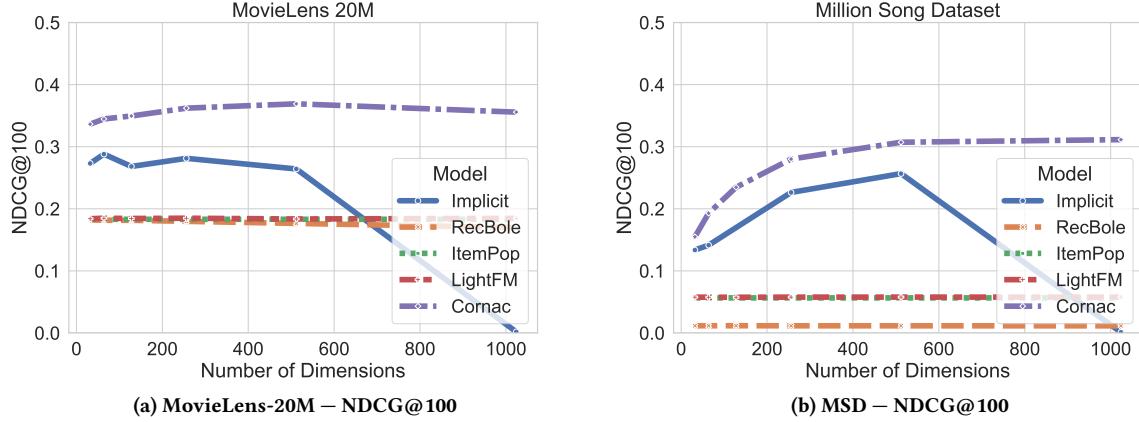


Figure 3: Performance in NDCG@100 relative to the number of embedding dimensions on the two datasets.

Table 3: The list of values for each subset of BPR features we use in the experiments. Bold denotes the best set of features according to our experiments in Section 5.2 for ML-20M.

Feature	Values
Item Biases	Enabled Disabled
Regularization	$\lambda_u p_u^2 + \lambda_i q_i^2 + \lambda_j q_j^2 (\lambda_u, \lambda_i, \lambda_j)$ $\lambda_u p_u^2 + \lambda_i q_i^2 + \lambda_j q_j^2 (\lambda_u, \lambda_i)$ $\lambda p_u^2 + \lambda q_i^2 + \lambda q_j^2 (\lambda)$ No Regularization
Optimizer	SGD Momentum SGD RMSprop Adam
Negative Sampling	Uniform Adaptive

these features on model’s performance, we conduct experiments comparing various subsets of the features using our implementation: (1) item biases, (2) regularization approaches, (3) optimizers, and (4) negative sampling. To speed up the hyperparameters search, we first fit our models with fewer epochs and then train the best model with the best hyperparameters using more epochs. Specifically, we opt for 70 epochs for the hyperparameters search phase, 1000 epochs for the best model on the ML-20M dataset, and 200 epochs for the MSD dataset. Such a two-step approach for hyperparameters search proves efficient for the BPR model, as it continues to train after thousands of epochs with minimal overfitting, mainly when uniform-based sampling produces good negative samples according to [48]. Additionally, the training protocol employs the Early Stopping criterion with patience equal to 13 epochs.

We conduct experiments following the grid of possible values for each feature outlined in Table 3. The most comprehensive method for experimenting with these features involves individually testing each combination of values. However, the cardinality of this approach is 64 experiments on each dataset, which results in over

5000 models. In order to lower the number of combinations, we split the features into three subsets: (1) item biases and regularization, (2) optimizers, and (3) negative sampling. Notably, *the first subset* holds the highest importance, as the optimal combination of values identified here serves as the base model for subsequent subsets. It lowers the number of experiments to 14 for each dataset (over 1000 models) but adds the sequential order to them.

Furthermore, we plan to evaluate the impact of the embedding dimension on the model in isolation. This modification significantly increases the number of trained models, particularly by a factor of six, thereby considerably prolonging the experiment’s duration. To streamline this, we obtain a rough estimate of the embedding dimension’s influence from open-source implementations. We conduct a full hyperparameters search for the Cornac, Implicit, LightFM, and RecBole libraries, using all available hyperparameters for the BPR model provided by these implementations. We excluded My-MediaLite and Elliot from the list due to their significantly longer training times than the other open-source implementations.

Figure 3 summarizes the results of this estimate. It is evident that Cornac achieves the best results with embedding dimensions of 512 and 1024 on both datasets. Surprisingly, there is a huge performance drop when Implicit employs embedding dimensions of size 1024. We assume this behavior is related to popularity-based negative sampling utilized in BPR implementation for Implicit, which might hinder the model’s performance, as highlighted in [48]. LightFM and Recbole implementations exhibit even lower performance than Implicit. We observed that both LightFM and Recbole adapt uniform negative sampling, but there is a discrepancy in their optimization algorithms. Cornac and Implicit implementations adapt regular Stochastic Gradient Descent (SGD), whereas LightFM and Recbole employ Adagrad [14] and Adam [33], respectively. We suppose that the choice of optimization algorithms led to these results, supporting our initial idea that the optimizer is a vital feature of the BPR model and the findings described in Section 5.1.

Therefore, we chose embedding dimensions of 512 and 1024 for the subsequent experiments. However, we will present results with the embedding dimension of 1024, as it consistently yielded better performance for our implementation on ML-20M and MSD datasets.

Table 4: Evaluation of the BPR model in terms of NDCG@K (N@K) and Recall@K (R@K) using various subsets of item biases and regularization features. All models in the table were trained with the SGD optimizer and uniform negative sampling. The best value is bolded, and the second-best is underlined.

Features											
		Regularization									
Item Biases	User λ_u	Positive λ_i	Negative λ_j	Shared λ	N@5	N@10	N@100	R@5	R@10	R@100	
	Positive λ_u	Negative λ_i	Shared λ_j								
	ML-20M										
	X	X	X	X	X	0.2537	0.2504	0.3510	0.1246	0.1948	0.5735
	X	X	X	X	✓	0.2656	0.2627	0.3665	0.1316	0.2048	0.5938
	X	✓	✓	X	X	0.2721	0.2695	0.3694	0.1353	0.2100	0.5940
	X	✓	✓	✓	X	0.2929	<u>0.2879</u>	<u>0.3883</u>	<u>0.1398</u>	<u>0.2171</u>	0.6150
	✓	X	X	X	X	0.2517	0.2469	0.3499	0.1185	0.1863	0.5771
	✓	X	X	X	✓	0.2419	0.2381	0.3441	0.1180	0.1855	0.5707
	✓	✓	✓	X	X	0.2565	0.2514	0.3530	0.1221	0.1906	0.5766
	✓	✓	✓	✓	X	0.2748	0.2686	<u>0.3723</u>	0.1291	0.2020	<u>0.5996</u>
MSD											
	X	X	X	X	X	0.1938	0.1906	0.2753	0.1007	0.1543	0.4145
	X	X	X	X	✓	<u>0.2183</u>	<u>0.2150</u>	<u>0.3115</u>	<u>0.1098</u>	<u>0.1707</u>	0.4695
	X	✓	✓	X	X	0.2085	0.2057	0.3018	0.1052	0.1638	0.4593
	X	✓	✓	✓	X	0.1844	0.1828	0.2796	0.0939	0.1472	0.4396
	✓	X	X	X	X	0.1859	0.1828	0.2706	0.0960	0.1477	0.4142
	✓	X	X	X	✓	0.2249	<u>0.2196</u>	<u>0.3132</u>	<u>0.1128</u>	<u>0.1734</u>	<u>0.4659</u>
	✓	✓	✓	X	X	0.2126	0.2092	0.3027	0.1066	0.1657	0.4559
	✓	✓	✓	✓	X	0.1796	0.1773	0.2748	0.0891	0.1411	0.4360

Item Biases and Regularization. The numerical results of various combinations of these features are present in Table 4. We can see that models without biases outperform those with item biases in ML-20M across all metrics. Regarding regularization variants, however, the situation is not that apparent. In this dataset, the model incorporating all regularization factors from the original paper [49] outperforms others, followed closely by the model using two separate regularization constants for user and item embeddings. This finding suggests that distinct regularization lambdas for users, positive and negative items, are crucial for specific datasets.

In contrast, the MSD dataset presents different results. Here, the model with item biases performs best overall. However, the effectiveness of item biases varies depending on the regularization factors used. Specifically, biased models surpass unbiased counterparts within two types of regularization approaches: shared and user/item regularization factors. Nevertheless, the difference between biased and unbiased models with shared regularization is statistically insignificant in NDCG@100, determined by a paired t-test with Bonferroni correction ($p < 0.05$) [5]. These findings indicate that the optimal regularization approach and the presence of item biases for BPR depend on the dataset, necessitating careful tuning.

Furthermore, it is notable that the performance of models without regularization is on par with regularized counterparts. In certain instances, these non-regularized models surpassed their regularized counterparts, particularly those utilizing a single regularization factor for user and item embeddings. This phenomenon might be attributed to the effective regularization inherent in the BPR model through the negative sampling algorithm alone.

Optimizers. Table 5 provides an overview of the performance exhibited by various optimizer algorithms and negative samplers.

Table 5: Evaluation of the BPR model in terms of NDCG@K (N@K) and Recall@K (R@K) using various subsets of optimizer and negative sampling features. For ML-20M, the models were trained without item biases and with three separate regularization factors. For MSD, the models were trained without item biases and shared regularization. The best value is bolded, and the second-best is underlined.

Features													
		Optimizer		Negative Sampling									
SGD	Momentum	SGD	RMSProp	Adam	Uniform	Adaptive	N@5	N@10	N@100	R@5	R@10	R@100	
	✓	X	X	X	✓	X	0.2929	0.2879	0.3883	0.1398	0.2171	<u>0.6150</u>	
	✓	X	X	X	X	✓	<u>0.3085</u>	<u>0.3020</u>	0.4012	0.1488	<u>0.2299</u>	0.6258	
	X	✓	X	X	✓	X	0.2669	0.2638	0.3655	0.1275	0.2017	0.5939	
ML-20M		X	X	✓	X	✓	X	0.1379	0.1393	0.2209	0.0641	0.1086	0.4030
		X	X	✓	✓	✓	X	0.2853	0.2811	0.3818	0.1384	0.2157	0.6090
		X	X	X	✓	✓	X	<u>0.3137</u>	<u>0.3042</u>	<u>0.3986</u>	0.1510	0.2281	0.6119
		MSD											
		✓	X	X	X	✓	X	<u>0.2249</u>	<u>0.2196</u>	<u>0.3132</u>	<u>0.1128</u>	<u>0.1734</u>	<u>0.4659</u>
		✓	X	X	X	X	✓	0.2475	<u>0.2395</u>	<u>0.3289</u>	<u>0.1253</u>	0.1881	<u>0.4730</u>
		X	✓	X	X	✓	X	0.1516	0.1518	0.2441	0.0771	0.1232	0.3966
		X	X	✓	X	✓	X	0.0318	0.0329	0.0627	0.0160	0.0274	0.1161
		X	X	X	✓	✓	X	0.2014	0.2000	0.2950	0.1052	0.1625	0.4484
		X	X	X	✓	X	✓	0.2118	0.2088	0.3029	0.1106	0.1687	0.4537

In this section, we focus on optimizers utilizing uniform-based negative sampling exclusively. Additionally, although the model with item biases demonstrates the best performance on MSD, we picked the model without them for MSD in this section due to the challenges negative sampling algorithms face with biased models, discussed later in this section.

The table shows that standard SGD demonstrates commendable performance across both datasets, surpassing other optimization algorithms across all metrics. On the other hand, RMSProp shows the poorest performance, particularly on the MSD dataset, where these models primarily minimized regularization constraints.

Consequently, standard SGD is the preferred choice for the BPR model. However, its crucial drawback must be considered: standard SGD typically requires more epochs than other algorithms. Fortunately, this issue might be mitigated using advanced negative sampling algorithms, which have the potential to significantly reduce the number of epochs required, as we will show later in this section.

Notably, algorithms such as Momentum SGD and Adam require rigorous hyperparameter tuning for optimal performance. We observed that default configurations of these algorithms often yield inferior results. We suppose this phenomenon is associated with uniform-based negative sampling, which tends to generate more simple negatives than strong ones. As highlighted in [48], this imbalance can result in small gradient magnitudes, slowing model training. This issue especially affects adaptive and momentum-based optimizers as they keep a vector of gradient momenta. We assume that these effects result in poor performance of Recbole and LightFM implementations discussed in Section 5.1.

Thus, if the momentum primarily comprises gradients with small magnitudes, uniform-based sampling yielding a strong negative can result in smaller weight updates for momentum-based optimizers than standard SGD. Therefore, this dynamic between adaptive

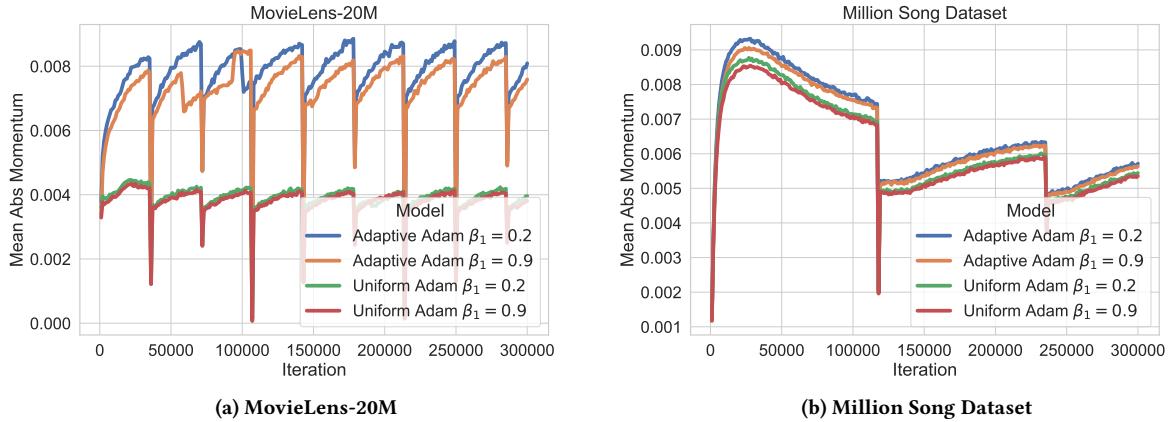


Figure 4: Mean absolute value of the first momentum in the Adam Optimizer with $\beta_1 = 0.2/0.9$ and uniform/adaptive negative sampling for two datasets over the first 300000 training iterations. The values are averaged over every 1000 iterations. Each drop on the graph indicates the beginning of an epoch.

optimizers and negative samplers may require more epochs to achieve satisfactory performance. Additionally, we hypothesize that it might lead to poorer performance in other algorithms that employ uniform-based sampling with an adaptive optimizer.

To address this issue, we reduced decay rates for moving averages in adaptive and momentum-based optimizers. This adjustment significantly improved performance, with decay factors in the range of 0.0 to 0.2, yielding better results. Figure 4 illustrates this behavior for the Adam optimizer. We see a slight difference (approximately 2% on both datasets) between Adam Optimizers with $\beta_1 = 0.2$ and $\beta_1 = 0.9$. It highlights that smaller momentum weights increase gradient magnitudes, improving the training process. However, even with these adjustments, the performance did not surpass that of regular SGD, which does not suffer from this issue.

Negative Sampling. To evaluate the effect of negative sampling, we conducted experiments using the two best-performing optimization algorithms from the previous section, namely SGD and Adam. Table 5 presents the results achieved on these optimization algorithms. It is evident from the table that adaptive sampling significantly improves the performance of all algorithms. Additionally, as outlined in [48], this negative sampling approach samples better negatives with higher gradient magnitudes, allowing models to perform better in fewer epochs. Notably, it reduced the number of epochs required by the SGD optimizer on the ML-20M dataset by 265 steps, resulting in only 85 epochs.

Additionally, Figure 4 illustrates the effect of adaptive negative sampling on the Adam optimizer’s first momentum. Obviously, momentum vectors tend to be smaller with uniform-based negative sampling than with adaptive sampling. Specifically, the difference is twice as much on the ML-20M dataset; for MSD, the difference is 5%. This might partially explain why the model with uniform-based sampling is just 4% worse than the adaptive-based sampling model.

Surprisingly, when we applied this negative sampling approach to biased models, it consistently resulted in poor performance across all datasets, regardless of other parameters. This phenomenon suggests that adaptive sampling may be ineffective when item biases are present, requiring further investigation in future works.

5.3 RQ3. Comparison with state-of-the-art models

We comprehensively evaluated various BPR implementations, comparing them against state-of-the-art baselines to assess their performance across various datasets and evaluation protocols. Table 6 summarizes the results for both user-based splits similar to [51] and global temporal splits discussed in [24, 30, 43]. Our implementation of BPR, utilizing both SGD and Adam optimizers, is included for comparison. For both optimizers in the global temporal split, we opt for the model with separated regularization factors ($\lambda_u, \lambda_i, \lambda_j$), disabled item biases, and adaptive negative sampling, which performed best on the ML-20M dataset according to Section 5.2.

In the user-based split evaluation, EASE consistently achieved the highest NDCG@K and Recall@K scores across both ML-20M and MSD datasets, demonstrating its robustness in recommending relevant items. Mult-VAE also performed well, particularly in Recall@100, surpassing EASE in some instances. Our SGD-based approach demonstrated competitive performance among the BPR implementations, particularly in the ML-20M dataset, outperforming other BPR implementations in most metrics. Our Adam-based BPR implementation also exhibited robust performance, demonstrating statistically insignificant differences from our SGD-based model across all metrics except Recall@100. Interestingly, our version with SGD optimizer outperformed Mult-VAE on the MSD dataset across all included metrics, while our Adam version occasionally performed worse than Mult-VAE.

The global temporal split evaluation is claimed to be preferred for the experiments [24, 30, 43]. Here, our BPR implementation with SGD optimization outperformed other methods in the ML-20M dataset, achieving the highest NDCG@K scores, although without statistical significance compared to EASE and Mult-VAE. Additionally, in the Yelp dataset, our BPR implementation using SGD optimization excelled, achieving the best results across all metrics with statistical significance. This finding indicates that our BPR implementations are adaptable and can maintain high performance across different datasets and evaluation protocols, highlighting their potential for various recommendation scenarios.

Table 6: Evaluation results of BPR implementations, baselines, and our BPR implementation in terms of NDCG@K (N@K) and Recall@K (R@K) using two evaluation protocols. For user-based evaluation protocol, the best models for open-source BPR implementations were selected from Figure 3. For our implementation in the user-based evaluation protocol, the best models were selected from Table 5. Values are formatted for convenience¹⁴.

User-based split similar to [51]													
	Models	ML-20M						MSD					
		N@5	N@10	N@100	R@5	R@10	R@100	N@5	N@10	N@100	R@5	R@10	R@100
Baselines	ItemPop	0.1298†	0.1275†	0.1906†	0.0580†	0.0912†	0.3298†	0.0360†	0.0349†	0.0582†	0.0176†	0.0271†	0.0986†
	EASE	0.3376†	0.3290†	0.4215†	0.1612†	0.2464†	<u>0.6371†</u>	0.3368†	0.3179†	0.3907†	0.1648†	0.2378†	0.5096†
	Mult-DAE	0.2949	0.2890†	0.3881†	0.1404†	0.2183†	0.6099†	0.2095†	0.1994†	0.2674†	0.1054†	0.1535†	0.3762†
	Mult-VAE	0.3071	0.3075	<u>0.4158†</u>	<u>0.1600†</u>	0.2460†	0.6520†	0.2221†	0.2156†	0.2973†	0.1144†	0.1716†	0.4307†
BPR Implementations	Cornac	0.2564†	0.2589†	0.3691†	0.1319†	0.2097†	0.6056†	0.2138†	0.2120†	0.3114†	0.1090†	0.1704†	0.4726
	Implicit	0.2071†	0.2052†	0.2880†	0.1093†	0.1673†	0.4745†	0.1749†	0.1737†	0.2567†	0.0929†	0.1425†	0.3898†
	LightFM	0.1295†	0.1262†	0.1843†	0.0544†	0.0869†	0.3087†	0.0362†	0.0349†	0.0575†	0.0175†	0.0268†	0.0964†
	RecBole	0.1242†	0.1213†	0.1785†	0.0513†	0.0820†	0.2973†	0.0004†	0.0009†	0.0117†	0.0002†	0.0008†	0.0328†
	Ours (SGD)	0.3085	0.3020	0.4012	0.1488	0.2299	0.6258	0.2475	0.2395	0.3289	0.1253	0.1881	0.4730
	Ours (Adam)	<u>0.3137</u>	0.3042	0.3986	0.1510	0.2281	0.6116†	0.2118†	0.2088†	0.3029†	0.1106†	0.1687†	0.4537†
Global temporal split recommended by [24, 30, 43]													
	Models	ML-20M (time-split)						Yelp					
		N@5	N@10	N@100	R@5	R@10	R@100	N@5	N@10	N@100	R@5	R@10	R@100
Baselines	ItemPop	0.0939†	0.0855†	0.0990†	0.0169†	0.0274†	0.1312†	0.0034†	0.0041†	0.0105†	0.0030†	0.0055†	0.0306†
	EASE	<u>0.1419</u>	<u>0.1332</u>	0.1698	0.0261	0.0450	0.2496	<u>0.0216†</u>	<u>0.0250†</u>	<u>0.0527†</u>	<u>0.0182†</u>	0.0307†	0.1387†
	Mult-DAE	0.1386	0.1308	0.1691	<u>0.0262</u>	0.0461	0.2455	0.0205†	0.0239†	0.0518†	0.0173†	0.0293†	0.1369†
	Mult-VAE	0.1343	0.1294	0.1722	0.0260	0.0473	0.2546	0.0184†	0.0214†	0.0468†	0.0161†	0.0269†	0.1268†
BPR Implementations	Cornac	0.1315	0.1263	0.1660	0.0256	0.0445	0.2416†	0.0208†	0.0247†	0.0566†	0.0176†	0.0310†	0.1565†
	Implicit	0.1140†	0.1063†	0.1315†	0.0214†	0.0370†	0.191†	0.0118†	0.0140†	0.0348†	0.0101†	0.0178†	0.1004†
	LightFM	0.0983†	0.0900†	0.1038†	0.0169†	0.0285†	0.1375†	0.0022†	0.0025†	0.0030†	0.0015†	0.0026†	0.0052†
	RecBole	0.0033†	0.0031†	0.0061†	0.0004†	0.0008†	0.0095†	0.0001†	0.0001†	0.0005†	0.0001†	0.0002†	0.0018†
	Ours (SGD)	0.1436	0.1357	0.1730	0.0277	<u>0.0471</u>	<u>0.2539</u>	0.0237	0.0276	0.0608	0.0203	0.0344	0.1658
	Ours (Adam)	0.1356	0.1277	0.1637†	0.0248	0.0431	0.2373†	0.0163†	0.0195†	0.0465†	0.0139†	0.0248†	0.1314†

¹⁴Within each column, the best value is bolded, the second-best is underlined, and † indicates a statistically significant difference ($p < 0.05$) from **Ours (SGD)** model, determined by a paired t-test with Bonferroni correction [5] for multiple comparisons.

Notably, among the open-source implementations considered, Cornac showed the best performance, aligning with the results obtained in Section 5.1. Other versions from Implicit, LightFM, and RecBole showed decreasing results in this order of performance, with LightFM and RecBole failing to learn anything meaningful.

6 CONCLUSIONS

In this work, we re-investigated matrix factorization with the BPR optimization criterion. We successfully replicated the BPR model on the Netflix dataset from the original BPR paper. Moreover, we analyzed open-source implementations of the BPR model, uncovering inconsistencies and deviations from the original paper that hindered their performance. Notably, implementation from the Cornac framework achieved the best performances among third-party implementations despite differing from the original model. Also, Elliot is the only open-source framework closely following the BPR paper.

Furthermore, our investigation into all model’s features revealed some intriguing insights. We found that the choice of regularization and negative sampling are pivotal for the model’s performance. Surprisingly, we also discovered that the SGD optimizer is a critical factor in achieving good results with the BPR model, as it is tightly linked to the negative sampling process. Also, our findings demonstrate that with proper tuning of these three features, our BPR implementation can achieve performance metrics close to state-of-the-art methods and even surpass some of them on several datasets. For instance, our model statistically significantly outperforms Cornac’s

results by 8.7% on ML-20M in NDCG@100 and Mult-VAE on MSD by 10%. Moreover, our implementation showcases exceptional results on datasets with global temporal split, surpassing even Mult-VAE and EASE. We hope these findings inspire further research into BPR-based extensions and other models that utilize the BPR objective, such as LightGCN, GRU4Rec, and Transformers4Rec.

ACKNOWLEDGMENTS

The contribution is an output of a research project implemented in the TBANK and the Laboratory for Models and Methods of Computational Pragmatics at the National Research University Higher School of Economics (HSE University).

REFERENCES

- [1] M Mehdi Afsar, Trafford Crump, and Behrouz Far. 2022. Reinforcement learning based recommender systems: A survey. *Comput. Surveys* 55, 7 (2022), 1–38.
- [2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-Generation Hyperparameter Optimization Framework. In *The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2623–2631.
- [3] Vito Walter Anelli, Alejandro Bellogín, Tommaso Di Noia, Dietmar Jannach, and Claudio Pomo. 2022. Top-n recommendation algorithms: A quest for the state-of-the-art. In *Proceedings of the 30th ACM Conference on User Modeling, Adaptation and Personalization*. 121–131.
- [4] Vito Walter Anelli, Alejandro Bellogín, Antonio Ferrara, Daniele Malitesta, Felice Antonio Merra, Claudio Pomo, Francesco Maria Donini, and Tommaso Di Noia. 2021. Elliot: A Comprehensive and Rigorous Framework for Reproducible Recommender Systems Evaluation. In *SIGIR ’21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11–15, 2021*, Fernando Diaz, Chirag Shah, Torsten

- Suel, Pablo Castells, Rosie Jones, and Tetsuya Sakai (Eds.). ACM, 2405–2414. <https://doi.org/10.1145/3404835.3463245>
- [5] Richard A Armstrong. 2014. When to use the Bonferroni correction. *Ophthalmic and Physiological Optics* 34, 5 (2014), 502–508.
- [6] Nabihi Asghar. 2016. Yelp Dataset Challenge: Review Rating Prediction. arXiv:1605.05362 [cs.CL]
- [7] James Bennett, Stan Lanning, et al. 2007. The Netflix Prize. In *Proceedings of KDD cup and workshop*, Vol. 2007. New York, 35.
- [8] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger (Eds.), Vol. 24. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfcd12577bc2619bc635690-Paper.pdf
- [9] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. 2011. The Million Song Dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*.
- [10] Erik Cambria and Bebo White. 2014. Jumping NLP curves: A review of natural language processing research. *IEEE Computational intelligence magazine* 9, 2 (2014), 48–57.
- [11] Lei Chen, Le Wu, Kun Zhang, Richang Hong, Defu Lian, Zhiqiang Zhang, Jun Zhou, and Meng Wang. 2023. Improving recommendation fairness via data augmentation. In *Proceedings of the ACM Web Conference 2023*. 1012–1020.
- [12] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.
- [13] Khalil Damak, Sami Khenissi, and Olfa Nasraoui. 2021. Debiased explainable pairwise ranking from implicit feedback. In *Proceedings of the 15th ACM Conference on Recommender Systems*. 321–331.
- [14] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research* 12, 61 (2011), 2121–2159. <http://jmlr.org/papers/v12/duchi11a.html>
- [15] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. 2019. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM conference on recommender systems*. 101–109.
- [16] Milena Filipovic, Blagoj Mitrevski, Diego Antognini, Emma Lejal Glaude, Boi Faltings, and Claudiu Musat. 2020. Modeling online behavior in recommender systems: The importance of temporal context. *arXiv preprint arXiv:2009.08978* (2020).
- [17] Ben Frederickson. 2016. Implicit: Fast Python Collaborative Filtering for Implicit Datasets. <https://github.com/benfrederic/implicit>
- [18] Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2011. MyMediaLite: A Free Recommender System Library. In *5th ACM International Conference on Recommender Systems (RecSys 2011)* (Chicago, USA).
- [19] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4, Article 19 (dec 2015), 19 pages. <https://doi.org/10.1145/2827872>
- [20] Ruining He and Julian McAuley. 2016. VBPR: visual bayesian personalized ranking from implicit feedback. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 30.
- [21] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 639–648.
- [22] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.
- [23] Balázs Hidasi and Ádám Tibor Czapp. 2023. The Effect of Third Party Implementations on Reproducibility. In *Proceedings of the 17th ACM Conference on Recommender Systems (RecSys '23)*. ACM. <https://doi.org/10.1145/3604915.3609487>
- [24] Balázs Hidasi and Ádám Tibor Czapp. 2023. Widespread Flaws in Offline Evaluation of Recommender Systems. In *Proceedings of the 17th ACM Conference on Recommender Systems (RecSys '23)*. ACM. <https://doi.org/10.1145/3604915.3608839>
- [25] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM international conference on information and knowledge management*. 843–852.
- [26] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE international conference on data mining*. Ieee, 263–272.
- [27] Kurt Jacobson, Vidhya Murali, Edward Newett, Brian Whitman, and Romain Yon. 2016. Music Personalization at Spotify. In *Proceedings of the 10th ACM Conference on Recommender Systems* (Boston, Massachusetts, USA) (RecSys '16). Association for Computing Machinery, New York, NY, USA, 373. <https://doi.org/10.1145/2959100.2959120>
- [28] Dietmar Jannach, Ahtsham Manzoor, Wanling Cai, and Li Chen. 2021. A survey on conversational recommender systems. *ACM Computing Surveys (CSUR)* 54, 5 (2021), 1–36.
- [29] Yitong Ji, Aixin Sun, Jie Zhang, and Chenliang Li. 2020. A re-visit of the popularity baseline in recommender systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1749–1752.
- [30] Yitong Ji, Aixin Sun, Jie Zhang, and Chenliang Li. 2023. A Critical Study on Data Leakage in Recommender System Offline Evaluation. *ACM Transactions on Information Systems* 41, 3 (Feb. 2023), 1–27. <https://doi.org/10.1145/3569930>
- [31] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4 (1996), 237–285.
- [32] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. arXiv:1808.09781 [cs.IR]
- [33] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]
- [34] Maciej Kula. 2015. Metadata Embeddings for User and Item Cold-start Recommendations. arXiv:1507.08439 [cs.IR]
- [35] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.
- [36] Ming Li, Sami Juliani, Mozhdeh Ariannezhad, and Maarten de Rijke. 2023. A next basket recommendation reality check. *ACM Transactions on Information Systems* 41, 4 (2023), 1–29.
- [37] Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara. 2018. Variational Autoencoders for Collaborative Filtering. arXiv:1802.05814 [stat.ML]
- [38] Weiyang Lin, Sergio A Alvarez, and Carolina Ruiz. 2000. Collaborative recommendation via adaptive association rule mining. *Data Mining and Knowledge Discovery* 6, 1 (2000), 83–105.
- [39] Hongzhi Liu, Zhonghai Wu, and Xing Zhang. 2018. CPLR: Collaborative pairwise learning to rank for personalized recommendation. *Knowledge-Based Systems* 148 (2018), 31–40.
- [40] Tie-Yan Liu. 2011. *Learning to Rank for Information Retrieval*. Springer Berlin, Heidelberg. 33 – 102 pages.
- [41] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. 2011. Content-based recommender systems: State of the art and trends. *Recommender systems handbook* (2011), 73–105.
- [42] Pattie Maes. 1995. Agents that reduce work and information overload. In *Readings in human-computer interaction*. Elsevier, 811–821.
- [43] Zaiqiao Meng, Richard McCreadie, Craig Macdonald, and Iadh Ounis. 2020. Exploring data splitting strategies for the evaluation of recommendation models. In *Proceedings of the 14th acm conference on recommender systems*. 681–686.
- [44] Xia Ning and George Karypis. 2011. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th international conference on data mining*. IEEE, 497–506.
- [45] Weike Pan and Li Chen. 2013. Gbpr: Group preference based bayesian personalized ranking for one-class collaborative filtering. In *Twenty-Third International Joint Conference on Artificial Intelligence*.
- [46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv:1912.01703 [cs.LG]
- [47] Aleksandr Petrov and Craig Macdonald. 2022. A Systematic Review and Replicability Study of BERT4Rec for Sequential Recommendation. arXiv:2207.07483 [cs.IR]
- [48] Steffen Rendle and Christoph Freudenthaler. 2014. Improving pairwise learning for item recommendation from implicit feedback. *WSDM 2014 - Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, 273–282. <https://doi.org/10.1145/2556195.2556248>
- [49] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proc. UAI 452–461*.
- [50] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*. 811–820.
- [51] Steffen Rendle, Walid Krichene, Li Zhang, and Yehuda Koren. 2021. Revisiting the Performance of iALS on Item Recommendation Benchmarks. arXiv:2110.14037 [cs.IR]
- [52] Aghiles Salah, Quoc-Tuan Truong, and Hady W Lauw. 2020. Cornac: A Comparative Framework for Multimodal Recommender Systems. *Journal of Machine Learning Research* 21, 95 (2020), 1–5.
- [53] Valeriy Shevchenko, Nikita Belousov, Alexey Vasilev, Vladimir Zholobov, Artyom Sosedka, Natalia Semenova, Anna Volodkevich, Andrey Savchenko, and Alexey Zaytsev. 2024. From Variability to Stability: Advancing RecSys Benchmarking Practices. arXiv:2402.09766 [cs.IR]
- [54] Harald Steck. 2019. Embarrassingly shallow autoencoders for sparse data. In *The World Wide Web Conference*. 3251–3257.
- [55] Xiaoyuan Su and Taghi M Khoshgoftaar. 2009. A survey of collaborative filtering techniques. *Advances in artificial intelligence 2009* (2009).

- [56] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 1441–1450.
- [57] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. 2013. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28* (Atlanta, GA, USA) (ICML '13). JMLR.org, III–1139–III–1147.
- [58] T. Tieleman and G. Hinton. 2012. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.
- [59] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. 2018. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience* 2018 (2018).
- [60] Jun Wang, Arjen P De Vries, and Marcel JT Reinders. 2006. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. 501–508.
- [61] Shoujin Wang, Longbing Cao, Yan Wang, Quan Z Sheng, Mehmet A Orgun, and Defu Lian. 2021. A survey on session-based recommender systems. *ACM Computing Surveys (CSUR)* 54, 7 (2021), 1–38.
- [62] Shijie Wang, Guiling Sun, and Yangyang Li. 2020. SVD++ Recommendation Algorithm Based on Backtracking. *Information* 11, 7 (2020). <https://doi.org/10.3390/info11070369>
- [63] Wenjie Wang, Yiyuan Xu, Fuli Feng, Xinyu Lin, Xiangnan He, and Tat-Seng Chua. 2023. Diffusion recommender model. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 832–841.
- [64] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. 2022. Graph neural networks in recommender systems: a survey. *Comput. Surveys* 55, 5 (2022), 1–37.
- [65] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. 2016. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the ninth ACM international conference on web search and data mining*. 153–162.
- [66] Lanling Xu, Zhen Tian, Gaowei Zhang, Junjie Zhang, Lei Wang, Bowen Zheng, Yifan Li, Jiakai Tang, Zeyu Zhang, Yupeng Hou, Xingyu Pan, Wayne Xin Zhao, Xu Chen, and Ji-Rong Wen. 2023. Towards a More User-Friendly and Easy-to-Use Benchmark Library for Recommender Systems. 2837–2847.