Incremental SVD-based Collaborative Filtering Enhanced with Diversity for Personalized Recommendation

Minh Quang Pham[1], Thi Thanh Sang Nguyen[2], Pham Minh Thu Do[3]

School of Computer Science and Engineering
International University, Vietnam National University, HCMC, Vietnam
[1]quangphamm1902@gmail.com, [2]nttsang@hcmiu.edu.vn,
[3]dpmthu@hcmiu.edu.vn

Adrianna Kozierkiewicz

Faculty of Computer Science and Management, Wroclaw University of Science and Technology, Wroclaw, Poland
adrianna.kozierkiewicz@pwr.edu.pl

Abstract. Along with the rapid rise of the internet, an e-commerce website brings enormous benefits for both customers and vendors. However, many choices are given at the same time makes customers have difficulty in choosing the most suitable products. A rising star solution for this is the recommender system which helps to narrow down the amount of suitable and relevant products for each customer. Matrix factorization is one of the most popular techniques used in recommender systems because of its effectiveness and simplicity. In this paper, we introduce a matrix factorization-based recommender system using Singular Value Decomposition (SVD) with some improvements in collaborative filtering and incremental learning. The SVD-based collaborative filtering methods can help generate personalized recommendations by combining user profiles. Moreover, the recommendation lists generated by the system are enhanced with diversity, which might attract more customer interests. Amazon's Electronic data set is used to evaluate our proposed framework of the SVD-based recommender system. The experimental results show that our framework is promising.

Keywords: Recommender System, Collaborative Filtering, Matrix Factorization, Singular Value Decomposition, Explicit Query Aspect Diversification

1 Introduction

An effective recommender system is not only the one that could give high accuracy rating-prediction for a user-item pair, but also give the least biased, most related, and reliable recommendations for each user.

In this paper, we propose a recommender system using collaborative filtering methods based on Singular Value Decomposition (SVD) [1]. To improve the limitation of SVD algorithm on producing online recommendations for new users, we build a more flexible version of SVD, called incremental SVD. This extended version helps the model to be able to fit new data without re-doing all the computation all over data again. Moreover, the popularity bias problem of the recommendation list is solved using a method called Explicit Query Aspect Diversification (xQuAD) [2]. For recommendation model evaluation, we use two metric types, one for measuring the prediction accuracy and one for measuring recommendation ranking accuracy. A number of experiments have been carried out to evaluate our proposal, compared with other collaborative filtering methods using K-Nearest Neighbors (KNN) [3] (Item-based and User-based) and Alternating Least Squares (ALS) [4].

The following sections will present related work (Section 2), methodology (Section 3), and experimental results (Section 4). Conclusions and future work are presented in Section 5.

2 Collaborative Filtering

Recommendation can be classified into two ways [5]: content-based filtering predicts items the user might like based on the item description, and collaborative filtering uses the knowledge about preferences of users for some items to make recommendations. In this paper, we will focus on collaborative filtering methods.

Collaborative filtering techniques which are most used [6, 7, 8] look at user previous ratings or transactions to make recommendations. Moreover, it uses the inter-dependent relationships between users and products for generating recommendations. Two popular methods of collaborative-filtering are neighborhood-based models and latent-factor models. While the neighborhood-based models only focus on users or products at a time, the latent-factor model tries to predict the ratings based on both product and user characteristics, called "latent factors". These latent factors are not always defined in the dataset but are inferred from the rating patterns.

2.1 Neighborhood Model [3]

The neighborhood model assumes users or items that are closed to each other would have similar ratings. The neighborhood models making predictions based on "k" nearest neighbors are called Item-based K-Nearest Neighbor (if neighbors are items) and User-based K-Nearest Neighbor (if neighbors are users) [9, 10, 11].

Item-based K-Nearest Neighbor. Item-based K-Nearest Neighbor (Item-based KNN) predicts the rating of a user *u* on item *i* by finding items similar to *i*. The similarity score between two items can be computed using Cosine Similarity as:

$$similarity(i,j) = \frac{r_i . r_j}{\|r_i\| \|r_j\|} \tag{1}$$

where, r_i and r_j are the rating vectors of the two items *i* and *j*, respectively, made by users.

Then, to predict rating on an item, the model will select *K* nearest neighbors for that item. Such prediction can be inferred from neighbors using the weighted mean:

$$\widehat{r_{ui}} = \frac{\sum_{j\in N_u^k(i)} similarity(i,j).r_{uj}}{\sum_{j\in N_u^k(i)} similarity(i,j)} \tag{2}$$

where $N_u^k(i)$ is the *k* nearest neighbors of item *i* that are rated by user *u*, r_{uj} is the rating of user *u* on item *j*, and $\widehat{r_{ui}}$ is the predicted rating of user *u* on item *i*.

User-based K-Nearest Neighbor. Similarly, the prediction can be made using a user-based model as:

$$\widehat{r_{ui}} = \frac{\sum_{v\in N_i^k(u)} similarity(u,v).r_{vi}}{\sum_{v\in_i^k(u)} similarity(u,v)} \tag{3}$$

where, $similarity(u,v)$ is the similarity between user *u* and *v*, and r_{vi} is the rating of user *v* on item *i*.

It is worth notice that in a real e-commerce system, the number of customers and items increases rapidly, there are many problems that need to be addressed, such as, cold-star [12], sparsity and scalability [13,14].

2.2 Matrix-Factorization Model

Matrix Factorization approaches [15] are latent factor models that base on previous user-item interactions to characterize users and items at the same time.

For matrix factorization models, each item i could be represented by a vector $q_i \epsilon \mathbb{R}^f$ and each user u could be represented by a vector $p_u \epsilon \mathbb{R}^f$ where *f* is the dimension of the joint latent factor space [15]. The vector q_i values measure how "close" the item is to those factors. Similarly, the vector p_u measures the level of interest of the user on items close to those corresponding factors. The approximated rating of user *u* on item *i* could then be inferred as:

$$\widehat{r_{ui}} = q_i^T p_u \tag{4}$$

Singular Value Decomposition (SVD). SVD is the most popular matrix factorization methods that can be applied in recommender systems to *solve sparsity and scalability problem* [16,17, 18], used as dimensionality reduction technique and a powerful mechanism [19]. SVD [1] factorizes a matrix *R* into three matrices M, Σ, U, where q_i makes up the columns of U^T, p_u makes up the rows of *M*, and the diagonal matrix that acts as a scaler on *M* or U^T.

For recommendation, because of the sparsity of the rating matrix, such matrix *R* is impossible to construct. So instead of directly finding *M* and *U*, we find all the vectors

q_i, p_u such that equation (4) is satisfied for all users *u* and items *i*. To find all the vectors q_i, p_u, we try to minimize the squared error on the set of known ratings:

$$min_{q^*,b^*} \sum_{(u,i \in \kappa)} (r_{ui} - q_i^T p_u)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2) \quad (5)$$

where, λ is the regularization parameter, κ is the set of (u, i) pairs for which r_{ui} is from the training set. This optimization is not convex, in other words, it may be impossible to find the pair of p_u and q_i that minimize the sum. Instead, we can use Stochastic Gradient Descent (SGD) or Alternating Least Squares (ALS) [4].

The SGD algorithm iterates over the ratings in the training set, for each case, it computes the error as:

$$e_{ui} = (r_{ui} - q_i^T . p_u) \quad (6)$$

Then, vectors p_u and q_i can be updated simultaneously as:

$$q_i = q_i + \alpha(e_{ui}.p_u - \lambda q_i); \quad (7)$$

$$p_u = p_u + \alpha(e_{ui}.q_i - \lambda p_u) \quad (8)$$

where, α is learning rate.

An alternative algorithm for SGD is to use Alternating Least Squares, which, in some cases, is more effective. ALS techniques tend to fix vectors q_i and treat them as constants, optimize p_u, then fix p_u and optimize q_i, and repeat until convergence [4]. One advantage of this approach is that the computation could be done in parallel because the system computes each p_u, q_i independently of each other.

We can refine our matrix factorization model by adding biases for both users and items. The predicted rating then is computed as:

$$\widehat{r_{ui}} = \mu + b_u + b_i + q_i^T . p_u \quad (9)$$

where b_u is the user bias and b_i is the item bias, and μ is the average rating. The observed rating is then computed by adding the biases and average rating, the system now tries to minimize the new loss function:

$$\min_{p*,q*,b*} \sum_{(u,i \in \kappa)} (r_{ui} - \mu - b_u - b_i - q_i^T . p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2 + b_u^2 + b_i^2) \quad (10)$$

Incremental Singular Vector Decomposition [20]. The workflow of a recommender system is divided into two different stages: the offline stage or training stage and the online stage. In the online stage, the system gives real-time recommendations to users. A real-time recommendation generated by the SVD model is high-quality and reliable, but the training stage of SVD is computationally expensive and time-consuming. For a $m \times n$ user-item matrix, the time complexity of SVD is almost $\Theta(m)^3$ [20]. The fold-in technique is used to project new users to the space of existed user-item matrix. It assumes that the projection of the additional user (or rating) can also provide a reliable and good approximation of the trained model. The Incremental SVD allows the model to incrementally learn the new ratings and makes more reliable and accurate recommendation for new users without the expensive computation.

2.3 Long-tail Phenomenon

There exists a phenomenon in recommender systems called long-tail phenomenon [2] which is the distance between physical and online stores. All of the items in the storage can be divided into three different parts: the short head, the long tail, and the distant tail. The short head part is extremely popular items which are often offered to customers by physical stores. The tail part consists of two different areas. The long tail is the most distribution of items which are potential items that should be included by the recommender system, even though most recommender systems often ignore them. The distant tail includes items with few ratings as known as cold-star items which may have a bad effect on a collaborative filtering recommender system. As the goal of the recommender system is to build personalized recommendations for each user, the regular problem caused by the biased popularity is that the recommendation lists are very identical amongst users since they include a lot of popular items. A recommender system that only produces very similar recommendations for every user should not be considered as a reliable system.

Explicit Query Aspect Diversification. A solution to the above described problem is the Explicit Query Aspect Diversification (xQuAD) method, which has been used in the context of information retrieval, such as web search engines. The goal of this algorithm [2] is to use the existing ranked recommendation list L to produce a new recommend list S (|S| < |L|) that manages to control the popularity bias while keeping the acceptable accuracy. The new list is built iteratively using the formula:

$$P(v|u) + \lambda P(v, S'|u) \tag{11}$$

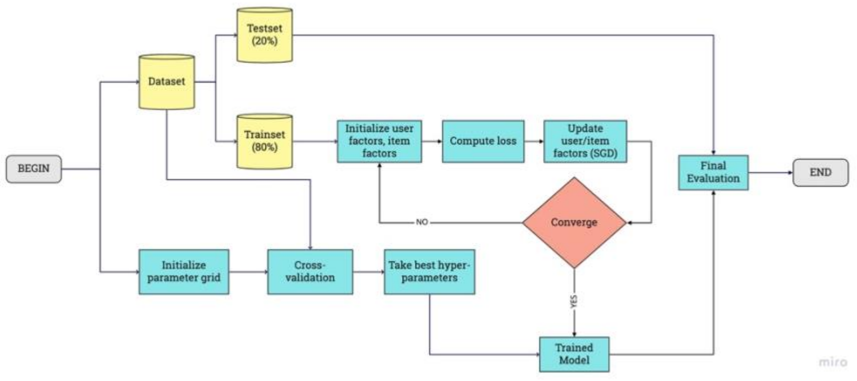
where $P(v|u)$ is the probability item *v* is interesting user *u* and $P(v, S'|u)$ denotes the probability of user *u* being interested in item *v* which is not currently in the recommend list *S*. The parameter λ is used to adjust the popularity bias. There are two different versions of xQuAD, one is the *Binary xQuAD* and another is *Smooth xQuAD* [2]. When selecting an item to be added to list *S*, the ranking score for each item in the recommendation list is computed as:

$$score = (1 - \lambda)P(v|u) + \lambda \sum_{c \in \{\Gamma, \Gamma'\}} P(c|u)P(v|c) \prod_{i \in S}(1 - P(i|c, S) \tag{12}$$

Briefly, for an item $v' \in c$, if the recommend list *S* does not cover (included in) area *c* (*c* represents either long-tail (Γ) or short-head (Γ ′) area), then the estimated user rating or preference will include the additional positive term. Thus, this increases the chance of an item to be selected, balancing the accuracy and the popularity bias.

3 Methodology

In this paper, the process of building the recommender system consists of two primary parts. Firstly, we will train and optimize the recommendation model for predicting the user ratings (Fig. 1). Then, we improve the relevance and reliability of the top-K recommendation generated by the recommender system in the first phase and use it in the final application to recommend products for users (Fig. 2).



Testset (20%)
Dataset
Trainset (80%)
BEGIN
Initialize user factors, item factors
Compute loss
Update user/item factors (SGD)
Final Evaluation
END
NO
Converge
YES
Initialize parameter grid
Cross-validation
Take best hyper-parameters
Trained Model
miro

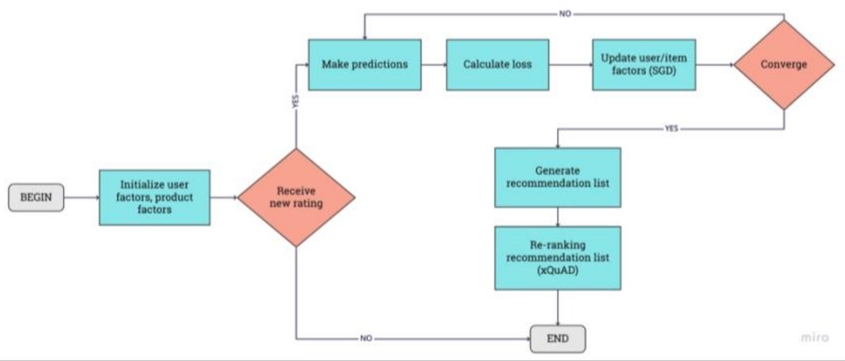
Fig. 1. The off-line stage of training and tuning the recommendation model.

To optimize the model for prediction accuracy, experiments are conducted with same training dataset and testing dataset on four different models which are Item-based, User-based K-Nearest Neighbors, SVD, and ALS. Random Search [21] is used to find the optimal hyperparameters. Two evaluation metrics Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) are used to evaluate prediction accuracy. The mathematical definition of RMSE and MAE are in Eq. (13) and (14).

$$MAE = \frac{1}{n}\sum_{u,i}|\hat{r}_{ui} - r_{ui}| \tag{13}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{u,i}(\hat{r}_{ui} - r_{ui})^2} \tag{14}$$

where, $\hat{r}_{ui}$ and r_{ui} are respectively the predicted and actual rating of user *u* on item *i* and *n* is the number of ratings.



NO
Make predictions
Calculate loss
Update user/item factors (SGD)
Converge
YES
YES
BEGIN
Initialize user factors, product factors
Receive new rating
Generate recommendation list
Re-ranking recommendation list (xQuAD)
NO
END
miro

Fig. 2. The online stage of producing real-time recommendations.

By experiments, the SVD model, which has the best performance is used for producing personalized recommendations. As mentioned before, an incremental version of SVD will also be proposed. The approach to implementing Incremental SVD is slightly different from the method in [20]. To predict the rating of a pair of user and product, either the user or the product is new (i.e., it isn't in the training set), the model first initializes its bias to 0. Then, its factor is randomly initialized from a normal distribution using the model's mean and standard deviation rating, which are set to 0 and 0.1 by default, respectively. The model then iteratively runs the Stochastic Gradient Descent for several epochs to update the new user or product biases and factors. After the partial fitting part, for each user, the model predicts the rating of that user and all products. The top-N highest rated products are used to build the recommendation list.

Then, to produce less biased recommendations, we follow the flexible approach to controlling the balance of the recommended items distribution in different areas, introduced in [2]. The framework is built using the xQuAD algorithm introduced in Section 2.3. The re-ranked recommendation list, whose size is smaller than the original list, is the final recommendation list.

4 Experimental Results

This section describes our experimental verification of the proposed framework for SVD-based recommender system. We first present the overview of the data set, the evaluation metrics for different aspects of the framework. Finally, we display the result of the experiment and propose our discussion on that.

4.1 Experimental Platform

In this section, four models are evaluated using Amazon's Electronic dataset [22]. The dataset has 7,824,482 ratings of 4,201,696 distinct users and 476,002 distinct products. After taking a deep look into the dataset, we observe that most users have rated less than 10 times (the 99th percentile of number of ratings per user is 12 ratings). Since we need to evaluate the top-N recommendation, we need an appropriate number of ratings for each user, therefore, users with less than 50 ratings are removed. The size of the dataset is reduced to 125,871 ratings of 1,540 users and 48,190 items. Without the pre-processing, the dataset can adversely affect the quality of the model. The training set size is 80% of the dataset and the rest are for the testing set.

4.2 Evaluation Metrics and Results

For this experiment, we evaluate our framework on two different types of metrics. The prediction accuracy metric will measure how good the model is in making prediction. Besides, an additional experiment is done to demonstrate the quality of the incremental approach of the SVD-based model. Later, using the ranking accuracy metric, we measure the effectiveness of the xQuAD algorithm on enhancing the top-N recommendation quality as well as its affection on the ranking accuracy of the system.

Prediction Accuracy. As mentioned in Section 3, the two metrics used for evaluating prediction accuracy are Root Mean Squared Error and Mean Absolute Error. Using the cross validation, the results in Fig. 3 is the average values of 5-fold cross validation. The high prediction accuracy of the SVD model stands out with the corresponding RMSE and MAE of approximate 0.998 and 0.735, respectively. The results of the two neighborhood models seem identical since their predictions are produced similarly. The ALS model performs worst amongst all.

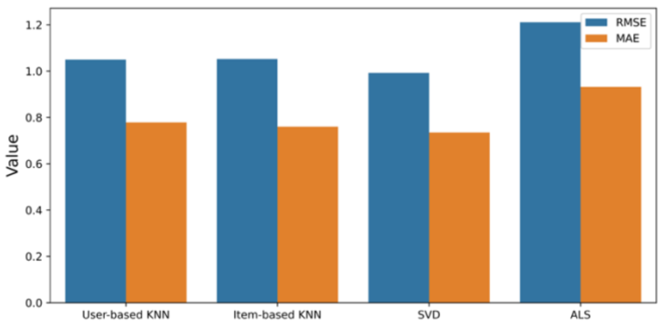


Fig. 3. The results of Root Mean Square Error and Mean Average Error.

Besides, to observe the performance of the incremental SVD approach, we divide the training set into two subsets with the ratio of x and $1-x$, respectively. Since we do not want the model to learn on too small data nor incrementally learn too big data, we choose to do experiments for x from 0.2 to 0.5. First, we train the model on the larger subset. Later, follow the incremental approach, we fold-in the other sub-set (called the fold-in set) into the model. Finally, we test the model on the testing set and observe how the prediction accuracy varies compared to the original model (trained on the full training set).

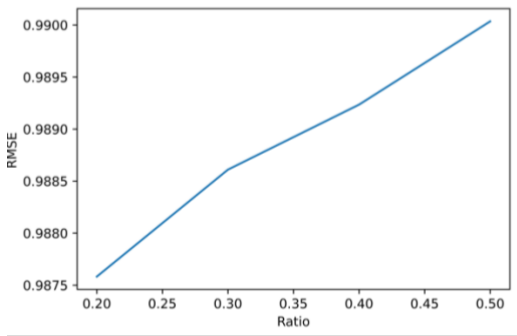


Fig. 4. Prediction Accuracy over Different Fold-in Set Ratio.

Fig. 4 plots the prediction quality result with the incremental model on different fold-in data ratio. It shows that even with a high ratio of fold-in set, the model is still be able to obtain good prediction. With the ratio of fold-in set and train-set of 0.5 (meaning that 50% of the original training data is used for fitting and the rest is used for folded-in), the RMSE accuracy is very close to the original model (0.990 compared to 0.998, only 0.8% drop). This suggests that the incremental technique offers a good quality model even when the fold-in set size is large.

Recommendation Ranking Accuracy. To evaluate the recommendation ranking accuracy, we use two different types of ranking metrics. One is the decision support metrics, in which we compute Precision@M, Recall@M, and F1@M Another one measures how much the trade-off is between balancing popularity bias and ranking accuracy, in which we compute the recommendation popularity, its coverage of long-tail items, and Normalized Discounted Cumulative Gain [23].

Precision@M, Recall@M, F1@M. In this paper, a product is considered relevant to a user if it is rated higher than 3.0 and is recommended when the system predicts its rating to be high than 3.0. By that, Precision@M is computed as the proportion of the first *M* recommended items that are relevant and Recall@M is the proportion of relevant items that are recommended in the first *M* recommended items. The F1@M is then computed as:

$$F1@M = 2 * \left(\frac{Precision@M * Recall@M}{Precision@M + Recall@M}\right) \tag{15}$$

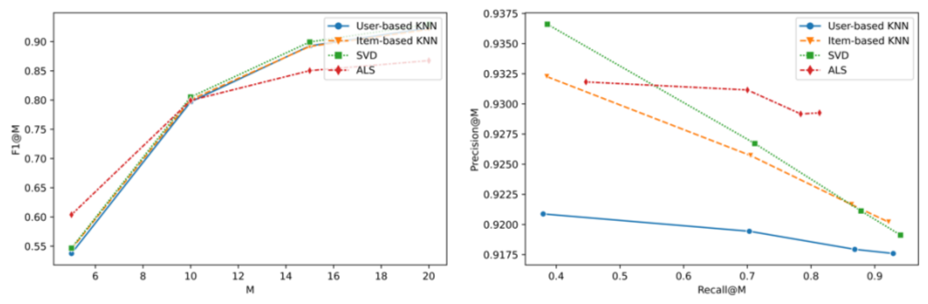


Fig. 5. The results of Precision@M, Recall@M, F1@M

The result illustrates in Fig. 5 is measured by using each model to predict the rating of all users on the items they rated in the test set. The result is the average of Precision@M, Recall@M, and F1@M for M from 5 to 20.

The accuracy for all models seems to be very high. The trade-off between the precision and recall is high for Item-based KNN and SVD models, and low for User-based KNN and ALS models. Amongst all, the most well perform model is the SVD model, however, the trade-off is high when M increases. As expected, the Recall@M increases when M increases since we are increasing the numerator of the Recall@M while keeping the dominator fixed. While the Recall@M tends to increase, the Precision@M decreases, once again, we can see the trade-off.

To solve the popularity bias problem, we apply metrics in [2]: Average Recommendation Popularity (ARP), Average Percentage of Long-tail Items (APLT), and Average Coverage of Long-tail Items (ACLT), to adjust popular items in the recommendation list and make it more personalized. Besides, Normalized Discounted Cumulative Gain (NDCG) [23], a popular ranking metric, is computed for a user's recommendation list.

To build the top-N recommendation, the recommender system uses the trained model to predict each rating in the testing set. In this experiment, the top-50 highest rated products for each user are considered as their recommendation. From the top-50 recommendation list, we apply the xQuAD algorithm to get the final ranked top-10 recommendation list.

Fig. 6 is measured using two different versions of xQuAD algorithm on re-ranking the recommendation list. The *reg* in the x-axis label is the regularization parameter of the ranking scoring function, it adjusts how strongly the biased popularity is controlled. As the regularization gets bigger, more long-tail items are included in the recommendation list (measured by APLT and ACLT), and the average popularity of the recommendation decreases (measured by ARP). However, as the popularity bias is getting controlled more strongly, the ranking accuracy of the recommendation seems to be lower (Fig. 7).

Choosing the appropriate regularization parameter is crucial when using the xQuAD algorithm for recommender system. For the Amazon's Electronic dataset used in this paper, the optimal value of λ we choose is 0.4. With λ of 0.4, we prevent the algorithm from affecting adversely to the accuracy of the recommender system while keeping the power of the algorithm on controlling the popularity bias.

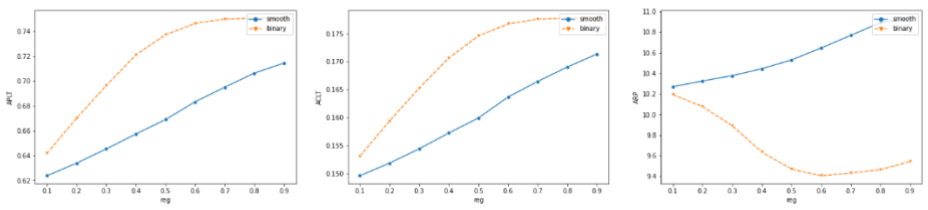


Fig. 6. Results of APLT, ACLT, ARP.

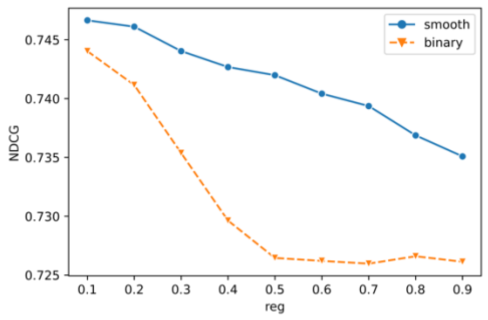


Fig. 7. Popularity Bias and Ranking Accuracy Trade-off.

5 Conclusions and Future Work

In this paper, our proposed method can overcome some limitations of usual recommender methods. First, we experimented on many algorithms and choose the one that fits best, which was the SVD-based recommendation model. After that, to deal with the expensive computation of the SVD algorithm in the online stage, we upgrade the SVD by applying the Incremental SVD method. Then, to produce high-quality recommendations, we balance the popularity bias of personalized recommendations while keeping an acceptable ranking accuracy (measured by NDCG) using the xQuAD algorithm. Finally, we apply different metrics to evaluate the effectiveness of the recommendation model on different aspects. As shown in the experiments, the proposed recommendation model can gain accuracy above 90% and include more popular items in the recommendation lists. This makes the recommender system achieve more effective performance.

In future, we plan to build an automatic pipeline for combining all of the methods and algorithms proposed in this paper. Such pipeline can be used with flexibility to fasten the process of build a recommender system. Besides, we also plan to use implicit data along with the existing explicit data to build a more effective model. Also, we will try to optimize the implementation of the Incremental SVD for a faster, less expensive fold-in technique.

References

1. Alter O, Brown PO, Botstein D.: Singular value decomposition for genome-wide expression data processing and modeling. Proc Natl Acad Sci U S A, 97, 10101-6 (2000).
2. Abdollahpouri, H., Burke, R., Mobasher, B.: Managing Popularity Bias in Recommender System with Personalized Re-ranking, (2019).
3. Koren, Y.: Factorization Meets the Neighborhood: A Multifaceted Collaborative Filtering Model. In: 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, USA (2008).
4. Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan: Large-scale Parallel Collaborative Fltering for the Netflix Prize, Proc. 4th Int'l Conf. Algorithmic Aspects in Information and Management, LNCS 5034, Springer, 337–348 (2008).
5. Lops, P., De Gemmis, M., Semeraro: Recommender System Handbook, Springer, Boston, (2011).
6. Naumov, Maxim, et al.: Deep learning recommendation model for personalization and recommendation systems (2019). arXiv preprint arXiv:1906.00091
7. Xiong, Ruibin, et al.: Deep hybrid collaborative filtering for web service recommendation, Expert systems with Applications, 110, 191-205 (2018).
8. Jiang, Liaoliang, et al.: A trust-based collaborative filtering algorithm for E-commerce recommendation system, Journal of Ambient Intelligence and Humanized Computing, 10(8), 3023-3034 (2019).
9. J.L. Herlocker, J.A. Konstan, A. Borchers, J. Riedl: An algorithmic framework for performing collaborative filtering, SIGIR Forum, 51, 227–234 (2017).

10. B. Sarwar, G. Karypis, J. Konstan, J. Riedl: Item-based collaborative filtering recommendation algorithms, in: Proc. 10th Int. Conf. World Wide Web, ACM, New York, NY, USA, pp. 285–295 (2001).
11. Jannach, D., Lerche, L., Kamehkhosh, I., & Jugovac, M.: What recommenders recommend: An analysis of recommendation biases and possible countermeasures. User Modeling and User-Adapted Interaction (2015).
12. S. Cleger-Tamayo, J.M. Fernández-Luna, J.F. Huete: Top-n news recommendations in digital newspapers, Knowl. Based Syst. 27, 180–189 (2012).
13. Claypool, Mark, et al.: Combing content-based and collaborative filters in an online newspaper (1999).
14. Sarwar, Badrul, et al.: Analysis of recommendation algorithms for e-commerce, Proceedings of the 2nd ACM conference on Electronic commerce (2000).
15. Bell, R.M., Koren, Y., and Volinsky, C.: Matrix Factorization Techniques for Recommender System (2009).
16. Barathy, R., and P. Chitra: Applying Matrix Factorization In Collaborative Filtering Recommender Systems, 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), IEEE (2020).
17. Nilashi, Mehrbakhsh, Othman Ibrahim, and Karamollah Bagherifard: A recommender system based on collaborative filtering using ontology and dimensionality reduction techniques, Expert Systems with Applications, 92, 507-520 (2018).
18. Teodorescu, Oana Maria, Paul Stefan Popescu, and Marian Cristian Mihaescu: Taking e-assessment quizzes-a case study with an SVD based recommender system, International Conference on Intelligent Data Engineering and Automated Learning, Springer, Cham (2018).
19. Zarzour, Hafed, et al: A new collaborative filtering recommendation algorithm based on dimensionality reduction and clustering techniques, 2018 9th International Conference on Information and Communication Systems (ICICS), IEEE (2018).
20. Sarwar, B., Karypis, G., Konstan, J., Reidl, J.: Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender System. In: 5th International Conference on Computer and Information Science (2002).
21. Bergstra, J., Bengio, Y.: Random Search for Hyper-Parameter Optimization. Journal of Machine Learning Research, 13, 281–305 (2012).
22. Ruining He and Julian McAuley. Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering. In Proceedings of the 25th International Conference on World Wide Web (WWW ’16). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 507–517 (2016).
23. Wang, Y., Wang, L., Li, Y., He, D., Liu, T., Chen, W.: A Theoretical Analysis of NDCG Type Ranking Measures. Journal of Machine Learning Research, 30 (2013).