Assessing the Performance of Recommender Systems with MovieTweetings and MovieLens Datasets

Ruipeng Li and Luiz Fernando Capretz

***Abstract*—Nowadays the Internet plays a significant role in our day-to-day life. An abundance of information is uploaded every second. The excess of information creates barriers to Internet users' ability to focus on their own interests. Therefore, numerous recommendation frameworks have been implemented to predict and provide suggestions to help users find their preferred items. However, it is difficult to find the most appropriate recommendation strategy(s), one that works best for the above issue. In this paper, we present a benchmarking experiment that is made by different recommendation algorithms on the MovieTweetings latest dataset and the MovieLens 1M dataset. The assessment focuses on four distinct categories of recommendation evaluation metrics in the Apache Mahout library. To make sure that we can control the benchmarking procedure efficiently and correctly, we also used the RiVaL toolkit as an evaluation tool. Our study shows that it is difficult to say which recommender algorithm provides the best recommendation and unfiltered datasets should be avoided in similar evaluations.**

***Index Terms*—Recommender systems, benchmarking recommender systems, recommendation frameworks, MovieTweetings and MovieLens datasets.**

I. Introduction

It is apparent that the volume of Internet users has been increasing sharply in past decades. There is a tremendous amount of different information be generated on the Internet every second. Internet users require using recommender systems to help them filter through various dataset sections such as hotels, music, films, travel destinations and so on. The main purpose of the recommender system is to discover the most correct, useful and surprising items from a set of selections that best match the interests of an end user [1]. With appropriate recommender algorithms, it is possible to demonstrate a personalized ranking model and promote customer loyalty.

Many website owners are starting to focus on analyzing the performance of recommender systems [2]. However, it was difficult to benchmark existing recommender strategies and algorithms. There are three major constraints: The first issue is related to the dataset selection; there exist different barriers including data sparsity, rating density and other biases of datasets. The second problem of evaluation is that there are numerous of different metrics that aim to evaluate various aspects of a recommender algorithm, it is hard to analyze the correct result without an explicit evaluation goal. Finally, many researchers realized that the measurement of accuracy is not enough to tell the performance of a recommender system, other relevant aspects also need to be measured in the evaluation process [3].

The purpose of this experiment is to figure out what is the best algorithm(s) that best matches users' interests about movies nowadays. In this paper, we will show an offline evaluation of predictions that are made by different collaborative filtering algorithms on the MovieTweetings latest dataset (on August 7th, 2017) and MovieLens dataset series (100k and 1M). The evaluation focuses on calculating four distinct categories of recommendation evaluation metrics (accuracy, decision support metrics, rank-aware top n and recommendation quality) in the Apache Mahout framework. We used the RiVaL toolkit as the evaluation tool to ensure that we can control the benchmarking procedure effectively and correctly. The rest of the report is organized as follows: First, we will provide a brief introduction of collaborative filtering recommender system in Section II. In Section III, we talk about evaluation components in detail. We elaborate on limitations and benefits of using existing datasets and the reason of dataset selection. We also discuss the evaluation toolkit and cross-validation in detail, as well as five evaluation metric categories and their advantages and disadvantages. Section IV demonstrates the evaluation setup. Section V discusses the evaluation result, as well as analyzation and comparison with different dataset. In section VI, we explain three problems that we encountered during the experiment. The last section talks about future works and conclusions.

II. Procedure for Paper Submission

The collaborative filtering recommender system would be one of the most successful and prevalent recommendation techniques in the world. The key idea of this approach is to assume that if there exist two users and they are similar in mind, they probably will like same items. The collaborative filtering technique then collects direct or indirect feedback, where direct feedback means item ratings and indirect feedback refers to the user clicking a link or searching a keyword [4].

The collaborative filtering recommender system can be divided into two broad categories: the memory-based and model-based methods [5]. The memory-based technique recommends items for a user by analyzing the ratings from other neighbor users. The most famous algorithm in the memory-based method is called nearest-neighbor collaborative filtering, and there are two main approaches: user-based and item-based, where the user-based approach calculates the prediction based on a group of nearest-neighbor users. Similarly, the item-based collaborative filtering method filters items from a subset of

Manuscript received July 7, 2019; revised November 12, 2019.

Ruipeng Li and Luiz Fernando Capretz are with Western University in Canada (e-mail: zack.liruipend@gmail.com, lcapretz@uwo.ca). Dr. L. F. Capretz is on leave and is currently a visiting professor of computer science at New York University in Abu Dhabi/UAE.

doi: 10.18178/ijimt.2019.10.6.865

nearest-neighbor items. Because the number of items would not be likely to grow sharply, the item-based collaborative filtering method can be calculated offline [6]. Moreover, the model-based technique generates prediction by constructing a probabilistic model; it collects a particular user's rating information and analyzes it in statistical models. One of the most popular algorithms of the model-based recommendation system is matrix factorization [7]. The singular value decomposition (SVD) technique is one of the matrix factorization techniques; it reduces the number of dataset features by lowering its space dimensions from N to K, where K is less than N [8].

A. Notation and Description

We used consistent mathematical notations for referencing various aspects of the recommender system model in this experiment. It is common to have a set U of users and a set I of items. The lowercase u and u' represent a specific user and a group of users in a user set, and the lowercase i and i' account for a particular item and a group of items in an item set. Ru is the set of items rated/liked or brought by the user u, and Au is the group of users who has rated/liked or brought the item i. Li is a collection of items remaining in the itemset. Nu is a set of nearest-neighbor users, and Ni is a set of nearest-neighbor items selected by the recommendation algorithm. Finally, Tn represents a list of items generated by the recommender algorithm. Table I shows the relationship of all notations.

TABLE I: COLLABORATIVE FILTERING NOTATION AND EXPLANATION

Notation	Explanation
U	Userset
I	Itemset
u, us	Users in the userset. us(u)$\in$ U
i, is	Items in the Itemset. is(i)$\in$ I
Ru	Items that the user u liked/rated. Ru $\subseteq$ I
Rui	User 's rating of item i
Au	Users who also likes item i. Au $\subseteq$ U
Li	Items left in the itemset. Lu $\subseteq$ I
Nu/Ni	Selected neighborhood user-set/itemset for the user u. 1.Nu $\subseteq$ Au 2.Ni $\cap$ Li = $\emptyset$
Tn	The top-n recommendation made by the algorithm. 1. Tn $\subseteq$ Li 2. Tn $\cap$ Ru = $\emptyset$

B. Final Stage Collaborative Filtering Algorithms

The user-based recommender provides predictions by finding the most similar set of users. The typical prediction process of this approach [5] used two standards to select Nu. The first rule is selected users must be in the Au group, and the second rule is selected users must share similar characteristics with the user u. If there is no such user in the Au group, the algorithm will predict the rating for a set of users in u recursively by finding the Nu of those selected users then adding those predicted ratings into the prediction process for the active user u. The algorithm that we used in the user-based recommendation is called generic user-based recommender. This algorithm calculates the similarity between u and i using different similarity metrics.

Sometimes [9] the information of an active user u is not available or irrelevant because the user u changed his/her short-term interest. Therefore, item-based algorithms have been developed. the item-based recommender generates predictions by detecting the most similar group of items. If items in the Li are highly similar to Ru, then those items in the Li will become part of Ni. To determine Tn, the algorithm first calculates the similarity between the item i and other items in the Li. The algorithm also checks the value of Ru and then computes a weight rating to each item in the Tn. Finally, the item-based recommender creates a similarity index for each item. The advantages of this algorithm are that the recommender may reduce the cost of calculation and maintain the recommendation quality. The opposite part is that the algorithm must calculate the similarity for every item in the itemset; thus, the calculation might take longer.

We selected GenericItemBased and GenericUserBased with five similarity classes (Pearson's Correlation, Uncentered Cosine Similarity, Spearman Rank, Euclidean Distance and Tanimoto Coefficient) in the Apache Mahout library [10].

Model-based recommender system commonly uses matrix factorization to generate predictions [2]. Matrix factorization methods characterize both items and users as vectors by deducting rating patterns; it provides recommendations based on the high parallelism between item and user factors. SVD recommenders are distinct by factorization types. In our experiment, we used three types of factorization: FunkSVD [11], RatingSGD, and SVD++ [7]. The FunkSVD is an incremental SVD algorithm that uses gradient descent to shorten the processing time. The algorithm iterates overall ratings in the training set and predicts the Ru, and then calculates all prediction errors. Rating SGD includes user and item biases. It can be training data faster. It is an optimized version of gradient descent. On the other hand, the SVD++ method was implemented based on SGD matrix factorization. The SVD++ method combines the latent factor model and neighborhood-based model. The algorithm not only focuses on how users rate, but it also considers "who has rated what." As a result, the accuracy has been increased.

C. Identified Issues

Memory-based methods have three identified issues: sparsity, scalability, and the cold-start problem [1]. The sparsity problem talks about shortages in a dataset. Typically, a user cannot provide too many ratings and can only lay over part of the item. As a result, the recommender algorithm may output a fewer number of items. One possible way to solve the high sparsity issue is to use model-based recommendation methods. Moreover, if the number of users and items increases sharply, then the time and resources that are required to run those algorithms also rises gradually. This is referred to as the scalability issue. One optimized solution for the scalability issue is to implement distributed processing methods that reduce the processing time for mapping.

The cold start problem is the most prevalent issue for memory-based methods; it appears when the recommender system is not able to collect sufficient information to provide trustable recommendations. Bobadilla *et al*. [12] declared three types of cold start problems: new community, new item, and new user. The new community code start means it is

impossible to start the recommendation due to lack of information. The new item has less impact of the system. It refers to new items that just added into system that do not have enough ratings. The new item cold start problem can be solved by creating a group of active users to rate each new item in the database mandatorily. Lastly, the new user issue is the one of the most critical problems [13]. A new user does not provide any ratings. Thus, it is impossible to obtain any personalized recommendations. A common solution to the new user cold start problem is to let new users submit additional information during registration.

A dataset is a collection of user ratings, and it contains several components. There are a lot of identified issues, and they involve in different areas. Some of them are caused by limitations of recommendation methods. Some of the biases are related to the data itself; these includes synonymy, gray sheep, and shilling attacks [14]. Synonymy refers to an item having multiple names. Most of the recommenders cannot distinguish the difference between names and count them as different items. Thus, the recommendation quality is decreased. Gray Sheep pertains to a group of users who cannot take advantage of the recommender system because they have inconsistent points of view. Shilling attacks have a bearing on the reliability of data. Any user who can access the Internet can create a rating for a product. In some cases, product stakeholders generate an adequate number of positive recommendations for their goods and give negative ratings to their business adversaries. In other words, these people provide counterfeit data to the database. Data synonymy and shilling attacks have a significant impact on the reliability and quality of recommendation, while gray sheep users are undesired.

III. Revaluation Design and Related Works

Recommender systems are designed with different strategies. However, they bring two problems: the first one is unapparent evaluation goals and tasks. Many studies evaluated incorrect or insufficient data because different strategies have different purposes and missions. To avoid this issue, the evaluation goal and tasks must be identified clearly at the beginning. In our experiment, we defined and validated evaluation objectives in the initial and planning stages. Moreover, the resulting comparison is that various libraries become an issue because the values of the metrics are usually different even when they use the same method. A minor difference in the method of developing algorithm and data organization may cause a notable change in results. Our solution is to use the RiVaL toolkit [14], a tool that can perform a cross-platform evaluation. Therefore, we can compare results generated by different frameworks effectively.

A. Goal and Tasks

The evaluation goal can differ because recommender systems can be designed in different ways and with different algorithms. Even though we already have a good algorithm, we should improve its performance. First, it is essential to understand what should be optimized. This is the core of evaluation, as it is difficult to evaluate a recommendation system without a clear goal. Many researchers point out that the measurement of the difference between the prediction and actual ratings is not adequate. Harper and Konstan [15] claims that a set of different evaluation metrics has been introduced to analyze the performance of recommender systems. This includes the usefulness of the algorithm, the correctness of top-ranked items, and the analyzation of the lack of coverage. Our evaluation goal is to identify which recommender system is better for movie website owners. We focused on accuracy, decision support metrics, rank-aware metrics, and quality.

Herlocker *et al*. [8] argued that the first step of the evaluation process is to understand user tasks that the system must achieve. In our point of view, finding some good items and finding credible recommenders are two important user tasks that best fit our evaluation goals. The term "find some good items" determines how good the recommender system is. Many recommender algorithms try to achieve accuracy by predicting how a specific user would like the list of items and rank them. The above process is indispensable, but it is not enough to identify the best recommender system(s). However, in some situations, customers want to overlook all the existing recommendations, so they spend a lot of time researching the recommendations and filtering the recommendations they think are useless [8].

Therefore, it is essential to premeditate coverage in the evaluation process. Furthermore, "finding credible recommenders" is another valuable task that must be complete. Many customers, in particular users who want to find useful and interesting music and movies, are not likely to accept a recommender automatically. Some users tend to change their account settings to investigate how the recommendation set changes. If the recommender system cannot provide items that are already known by and are guaranteed to satisfy users (i.e., very famous songs or movies), those users will withdraw from the service and seek other valuable recommender systems. The above situation is unacceptable for website owners. Therefore, it is desirable to analyze decision support metrics and rank-aware metrics such as Mean Average Precision (MAP), Precision@N, and nDCG@N.

B. Cross-Validation

The fundamental process of evaluation involves using existing data to simulate user behavior and check if a recommender method can predict the behavior successfully. Cross-validation is one such desired simulation method [15]. In a single fold cross-validation simulation, the selected datasets have been divided into two sections, a training dataset and a test dataset, which have different functions. On the one hand, the recommender algorithm interacts with the training data set, and the algorithm treats the training data as input and uses this data set to generate a list of recommendations and predictions. On the other hand, the test data has been concealed, and the recommender algorithm cannot touch it. It is noteworthy that the materials in the test data will never change. Then we compare the recommendations and predictions from the training data with the test data and calculate the divergence between them. In the last step, we use these calculations as inputs to generate various metrics. In the above process, we treat the training data as preferences or past behaviors that already exist in the

database. Similarly, we look at the test dataset as ratings that users are likely to give after they receive recommendations made by the recommender system. That is the simulated user behavior.

A potential issue with single fold cross-simulation is the recommender algorithm might evaluate a group of the easiest or hardest users. Thus, the accuracy might decrease. To avoid this issue, we used five-fold cross-validation, which separates data into multiple splits [9]. Each split has a training dataset and test dataset. For each partition, we perform a single fold cross-validation. Lastly, we average the value of the metrics. This way, the possibility of picking the hardest or easiest users will become extremely small.has rated what." As a result, the accuracy has been increased.

C. Datasets

Datasets would be one of the most important parts of the evaluation. We found several existing datasets and discovered that many of them are outdated and no longer available, including CAMRa (2012), Netflix (2007), and EachMovie (2004). Furthermore, we researched about Jester, which is a set of rated online jokes [17], as well as the Book-Crossing Dataset [18]. The above two datasets are in divergent domains and not related with evaluation goals.

MovieTweetings datasets. The original dataset was integrated from the IMDb website and has a rating scale from one to ten. It has many advantages. First, the dataset always includes the most recently added users and movies; second, the dataset uses similar formats and data structure. It is possible to perform a reproducible evaluation with other datasets. Lastly, it reduced the synonymy issue and lowered the possibility of shining attacks because of the way they collect data. One drawback is the sparsity problem; it is difficult for recommender systems to generate predictions for those people with few ratings. Another disadvantage is the age distribution of Twitter user's. According to Statistics, the age distribution of Twitter users in the United States was more than 60% between 18 to 44 years of age in 2016. Therefore, the MovieTweetings dataset may not contain too much data uploaded by the 45+ age group.

MovieLens Datasets. There are several versions of MovieLens datasets [9]. We observed two datasets that similar with the MovieTweetings latest dataset. The first one is MovieLens Latest Datasets, which contains twenty-six million ratings. The second one is MovieLens 1M datasets. In addition, we also found MovieLens 100k, which is the most popular dataset that has already been analyzed in many types of research and studies. We finally selected MovieLens 1M and 100k datasets. The first reason is the limitation of time, and the second reason is that MovieTweetings and MovieLens 1M datasets have close ratings. Therefore, it is convincing to make comparisons. Moreover, the evaluation result of the MovieLens 100k dataset is publicly available. We can compare them with the result generated in our experiment.

Table II demonstrates basic numerical information about selected datasets. The MovieTweetings dataset has the largest volume of users and movies but the least density, and vice versa for MovieLens 100k. In addition, MovieLens 1M datasets have the most ratings, and its density is ten times higher than the MovieTweetings dataset. There is no users and movies had been filtered in the MovieTweetings latest dataset [16], while the MovieLens dataset series only included users with more than twenty ratings. Although we selected different datasets, the density problem remains in the latest dataset.

TABLE II: DATASET INFORMATION

Dataset	Users	Ratings	Movies	Density	Latest rating
MT	50298	632225	28721	0.044%	Aug 07,2017
ML 1M	6000	1 M	4000	0.537%	Feb,2003
ML 100K	943	100000	1682	6.305%	Apr,1998

D. Evaluation Tool

The RiVaL toolkit is a cross-framework and open-source evaluation project implemented in Java. It allows experimenters to control the entire evaluation process with a transparent evaluation setting [14]. It divides the benchmarking process into four stages: data splitting, item recommendation, candidate item generation, and performance measurement. In our experiment, the item recommendation was performed by Apache Mahout and other phases were performed in RiVaL. The Apache Mahout framework uses the training and test file generated by cross-validation to create recommendation list and use it as input in candidate item generation stage. Therefore, in our case, stage one, three, and four were executed in the RiVaL toolkit.

IV. EVALUATION SETUP

Evaluation for MovieLens datasets (1M and 100K) are conducted on a Machenike laptop, which has a seventh-generation Intel Core i7-6700HQ processor @ 2.6GHz, 16GB DDR4, 2400MHz RAM, and NVIDIA GeForce GTX 1060 graphics card with 6GB video memory under the Windows operating system. Evaluation for MovieTweetings latest dataset is conducted on a desktop, which has a seventh-generation Intel Core i7-6700 processor @ 3.6GHz, 16GB DDR4, 2400MHz RAM, and NVIDIA GeForce GTX 1060 graphics card with 3GB video memory under the Windows operating system.

The actual process can be divided into seven steps. We apply different settings in our evaluation algorithm at the first phase. For instance, we set the cutoff to be 10, 20, and 50, and the neighborhood size is set to be 50, 100, and 200 in user-based and item-based recommender algorithms while in SVD recommenders, the value of the cutoff and neighborhood size are constant at 20 and 100. The variable of SVD recommenders is the number of interaction levels which is set to be 10, 20, and 50, respectively.

Second, a selected dataset (MovieTweetings and MovieLens) is downloaded and stored on the local drive. In the third phase, we perform the five-fold cross-validation. In the fourth and fifth steps, recommended items are generated by algorithms in the Apache Mahout, and candidate movies are generated. In the sixth stage, we use data generated in the fourth step as input to create five different evaluation results. In the final step, we store the evaluated result on Google drive for later data analyzation. We have also implemented a timer

to calculate the execution time from step four to six because this is the actual evaluation processing time.

V. Results and Discussions

In general, the total execution time for the MovieTweetings latest dataset is 1379155 seconds (about 383.09 hours). The evaluation time for MovieLens 1M is 1539738 seconds (about 427.7 hours). Furthermore, the total running time for MovieLens 100k is 30927 seconds (approximately 8.590 hours). According to the results, we observed that as the size of the dataset increased numerically, the algorithm execution time will increase exponentially. Moreover, the Item-based recommender has the highest running time, while the overall user-based recommender has the lowest time among all three datasets.

We discovered that several parameters can influence the performance of memory-based recommender algorithms. The first variable is the value of cutoff. Table III shows mean results and its relationship with the number of cutoff in each dataset. We also discovered that a change in the cutoff would only affect decision-support and rank-aware metrics; when the value of the cutoff rises, the amount of P@N and nDCG@N also increase. Another interesting point related to the cutoff is that it has a significant effect on the item-based recommender algorithm.

We noticed that the performance of nDCG@N and P@N increased sharply when the number of cutoffs increases. Furthermore, from the results, we can determine that values of MAE, RMSE, and MAP did not change when we set the cutoff from 10 to 50 and let the neighborhood size remain at 100. Therefore, the value of the cutoff cannot affect the value of accuracy metrics when the neighborhood size is constant. The second finding is that the user-based recommendation method has the highest MAE and RMSE across all three results. This means that the user-based recommendation strategy has the best overall accuracy. The third observation is that the values of nDCG in all three datasets are very low (less than 0.1), which means that users will not gain much from those algorithms and similarities.

The last observation is that the neighborhood size has a significant effect on the user-based recommender algorithm while it cannot influence the item-based recommender. For the user-based recommender algorithm, the rising trends in the neighborhood size will decrease the overall accuracy, the value of decision support metrics, and the rank-aware top n metrics, while the amount of coverage will increase. For example, if we increase the neighbor size from 50 to 200 and keep the cutoff at 10, whenever we use these kinds of similarity strategies, the algorithm will reduce the MAE and the RMSE by approximately 2%, and the value of P@10, nDCG@10, as well the MAP will drop about 50 to 60%. Table IV indicated above result. Moreover, the execution time and the neighbor size have a proportional relationship: when the size of the neighbor increases, the algorithm requires more computing time and vice versa. However, it is interesting to note that, in the item-based recommender method evaluation, changes in the neighborhood size did not affect the performance. The above result shows that the neighborhood size does not have a connection with item-based recommenders.

TABLE III: Comparison of Cutoff from Datasets

	Cutoff	nDCG @N	P@N	MAE	RMSE	MAP
ML 1M	10	0.0080983	0.0123392	0.8167670	1.0479166	0.0197379
	20	0.0102622	0.0134760	0.8167670	1.0479166	0.0197379
	50	0.0218086	0.0176419	0.8167670	1.0479166	0.0197379
ML 100K	10	0.0097958	0.0139963	0.8217578	1.0395967	0.0287574
	20	0.0179058	0.0201968	0.8217578	1.0395967	0.0287574
	50	0.0350645	0.0232610	0.8217578	1.0395967	0.0287574
MT	10	0.0104423	0.0050132	1.2400169	1.6397329	0.0287574
	20	0.0159250	0.0050281	1.2400169	1.6397329	0.0287574
	50	0.0235189	0.0056467	1.2400169	1.6397329	0.0287574

TABLE IV: Comparison of Neighborhood Size from Datasets

	Nsize	nDCG @N	P@N	MAE	RMSE	MAP
ML 1M	50	0.0184158	0.0270659	0.8373394	1.0742361	0.0205713
	100	0.0129493	0.0205141	0.8276088	1.0799900	0.0202672
	200	0.0118568	0.0181899	0.8178517	1.0427814	0.0228097
ML 100K	50	0.0220666	0.0308544	0.8500545	1.0817932	0.0312157
	100	0.0191629	0.0278134	0.8219783	1.0438510	0.0374045
	200	0.0108040	0.0154825	0.7998577	1.0125592	0.0382584
MT	50	0.0186235	0.0100869	1.2938143	1.7378427	0.0172397
	100	0.0130644	0.0065263	1.2811670	1.7227290	0.0130026
	200	0.0079058	0.0041793	1.2714237	1.7123390	0.0093901

For model-based recommender algorithms, we found that the performance of each metrics is unstable, which means that sometimes the increase in iteration improves the recommendation results and sometimes it does not. Therefore, it was difficult to say how the number of iterations can affect the final result. However, it can be concluded that the SVD++ algorithm produced the best results on nDCG@N and P@N in all three results, which means that the SVD++ provided the most useful recommendations at the top of the list and did not waste the users' time. Another interesting finding is the performance of the coverage. We can find that the MovieLens 100k dataset has the highest coverage in all three recommender algorithms, while the MovieTweetings dataset has the opposite result. As can be seen, the data sparsity issue discussed above would still be a challenge for a dataset with a large user population.

According to the above results and graphs, we can conclude that the best recommendation strategy for the Apache Mahout library is based on SVD recommenders. The reason is that the SVD recommender algorithm provides the most valuable and appropriate predictions at the top of the list and is the most helpful to users in making good decisions.

VI. Conclusion

In this paper, we presented an experiment for

benchmarking best recommender algorithm(s) that can predict users' interest in movies. The purpose of the evaluation has been clearly defined. Popular recommender algorithms and their advantages and drawbacks have been explained. All evaluation components and the process have been discussed. Our study shows that it is difficult to say which recommender algorithm provides the best recommendation. However, we can claim that the SVD++ provided more valuable and appropriate relative to the preferences on top of the list and will help users to make good decisions, while user-centered recommender systems can predict with better overall accuracy. There is no evidence to say which algorithm has the best quality. In addition, the MovieTweetings dataset can achieve better overall accuracy than MovieLens datasets while it did not perform well in other evaluation categories. The experiment also showed that unmodified datasets would cause lack of memory space and suspend the RiVal toolkit. Therefore, unfiltered datasets should be avoided in similar evaluations.

Due to the time limitation, we did not have a chance to evaluate the performance of MovieTweetings dataset on the LensKit library. Therefore, we want to perform that in the future so that we can retrieve more valuable information to compare and contrast. In addition, we need to improve the evaluation algorithm used in the RiVal toolkit. One possible improvement could be reorganized and optimize the algorithm used in the evaluation so that we can prevent or minimize problems we talked in section six. Another improvement may focus on developing more evaluation metrics such as serendipity and popularity, as well as adding more libraries for comparison.

CONFLICT OF INTEREST

The authors declare that there are is no conflict of interest.

AUTHOR CONTRIBUTIONS

Ruipeng Li processed the experimental data, performed the analysis, and drafted the manuscript. Luiz Fernando Capretz conceived the study and was in charge of overall direction and planning as a project advisor.

REFERENCES

[1] Y. Koren, "Factorization meets the neighborhood: A multifaceted collaborative filtering model," in *Proc. the 14th ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining (KDD '08).* New York, NY, USA, 2008, pp. 426-434.

[2] L. Baltrunas, T. Makcinskas, and F. Ricci, "Group recommendations with rank aggregation and collaborative filtering," in *Proc. the 4th ACM Conf. on Recommender Systems (RecSys '10),* New York, NY, USA, 2010, pp. 119-126.

[3] M. Ge, C. Delgado-Battenfeld, and D. Jannach, "Beyond accuracy: Evaluating recommender systems by coverage and serendipity," in *Proc. the 4th ACM Conf. on Recommender Systems (RecSys '10),* New York, NY, USA, 2010, pp. 257-260.

[4] S. G. Walunj and K. Sadafale, "An online recommendation system for e-commerce based on apache mahout framework," in *Proc. the 2013 Annual Conf. on Computers and People Research (SIGMIS-CPR '13).* ACM, New York, NY, USA, 2014, pp. 153-158.

[5] J. Zhang and P. Pu, "A recursive prediction algorithm for collaborative filtering recommender systems," in *Proc. the 2007 ACM Conf. on Recommender Systems (RecSys '07).* New York, NY, USA, 2007, pp. 57-64.

[6] D. Bokde, S. Girase, and D. Mukhopadhyay, "Matrix factorization model in collaborative filtering algorithms: A survey," *Procedia Computer Science,* vol. 49, pp. 136-146, 2013.

[7] N. Koenigstein and Y. Koren. "Towards scalable and accurate item-oriented recommendations," in *Proc. the 7th ACM Conf. on Recommender Systems (RecSys '13).* New York, NY, USA, 2013, pp. 419-422.

[8] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Trans. Inf. Syst. 22*, vol. 1, pp. 5-53, 2004.

[9] S. Dooms, T. De Pessemier, and L. Martens, "MovieTweetings: A movie rating dataset collected from twitter," in *Proc. Workshop on Crowdsourcing and Human Computation for Recommender Systems, CrowdRec,* 2013.

[10] P. Kantor, F. Ricci, L. Rokach, and B. Shapira, *Recommender Systems Handbook*, 1st ed., New York: Springer-Verlag, 2010.

[11] S. Funk. (December 2006). *On the Stochastic Gradient Descent Algorithm.* [Online]. Available: http://sifter.org/~simon/journal/20061211.html

[12] J. Bobadilla, F. Ortega, A. Hernando, and A. GutiéRrez, "Recommender systems survey," *Know.-Based Syst.,* vol. 46, pp. 109-132, 2013.

[13] T. Nathanson, E. Bitton, and K. Goldberg, "Eigentaste 5.0: Constant-time adaptability in a recommender system using item clustering," in *Proc. the 2007 ACM Conf. on Recommender Systems (RecSys '07).* New York, NY, USA, vol. 8, 2007, pp. 149-152.

[14] A. Said and A. Bellogín, "Rival: A toolkit to foster reproducibility in recommender system evaluation," in *Proc. the 8th ACM Conf. on Recommender Systems (RecSys '14),* New York, NY, USA, 2014, pp. 371-372.

[15] F. M. Harper and J. A. Konstan, "The MovieLens datasets: History and context," *ACM Trans. Interact. Intell. Syst.,* vol. 5, no. 4, vol. 19. 2015.

[16] C. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen, "Improving recommendation lists through topic diversification," in *Proc. the 14th International Conf. on World Wide Web (WWW '05).* New York, NY, USA, 2005, pp. 22-32.

[17] S. Dooms, A. Bellogín, T. D. Pessemier, and L. Martens, "A framework for dataset benchmarking and its application to a new movie rating dataset," *ACM Trans. Intell. Syst. Technol.,* vol. 7, no. 3, pp. 28-29, vol. 41, 2016.

[18] A. Rashid, G. Karypis, and J. Riedl, "Learning preferences of new users in recommender systems: An information theoretic approach," *SIGKDD Explor. Newsl.,* pp. 90-100. 2008.

Ruipeng Li received his bachelor's degree in computer science degree from Dalhousie University, Halifax, Nova Scotia, Canada, in 2016 and obtained his master of engineering in Western University, London, Ontario, Canada in the year of 2018. Ruipeng Li's research interest are big data and recommender systems. Ruipeng Li is currently a software programmer in Toronto.

Luiz Fernando Capretz has vast experience in the software engineering field as practitioner, manager and educator. Before joining the University of Western Ontario (Canada), he worked at both technical and managerial levels, taught and did research on the engineering of software in Brazil, Argentina, England, Japan and the United Arab Emirates since 1981. He is currently a professor of software engineering and assistant dean (IT and e-Learning), and former director of the software engineering program at Western. He was the director of informatics and coordinator of the computer science program in two universities in Brazil. He has published over 200 academic papers on software engineering in leading international journals and conference proceedings, and co-authored two books: *Object-Oriented Software: Design an Maintenance* published by World Scientific, and *Software Product Lines* published by VDM-Verlag. His current research interests are software engineering, human aspects of software engineering, software analytics, and software engineering education. Dr. Capretz received his Ph.D. from the University of Newcastle upon Tyne (U.K.), M.Sc. from the National Institute for Space Research (INPE-Brazil), and B.Sc. from UNICAMP (Brazil). He is a senior member of IEEE, a distinguished member of the ACM, a MBTI certified practitioner, and a certified professional engineer in Canada (P.Eng.). Dr. Capretz is currently spending his sabbatical leave as visiting professor of computer science at New York University in Abu Dhabi, United Arab Emirates. He can be reached at lcapretz@uwo.ca, and further information about him can be found at http://www.eng.uwo.ca/people/lcapretz/