`          `

by
Ruipeng Li
rli445@uwo.ca

Student No.: 250941655
Professor: Luiz Fernando Capretz

Department of Electrical and Computer Engineering
The University of Western Ontario
London, Ontario, Canada
October, 2017

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 five different 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.

Categories and Subject Descriptors

General Terms
Algorithms, Performance, Experimentation

Keywords
Recommender Systems; Evaluation; Benchmarking;Recommendation Frameworks; Experiments

TABLE OF CONTENT

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. It is common that people in the real world demand recommendations from others through various data transportation methodologies such as media, reference papers and so on (Walunj & Sadafale, 2013). Just like real-world scenarios, Internet users require using recommender systems to help Internet users filter through various dataset sections such as hotels, music, films, travel destinations and so on.

There are three major approaches to generating recommendations: collaborative filtering, content-based filtering, and the hybrid recommender. The collaborative filtering recommender system predicts suggestions from user's preferences and compares the similarity with a group of end users. The content-based filtering strategy selects similar items that users rated in the past. The hybrid recommender system is the combination of different filtering methodologies. In addition, different approaches have their own favorable and unfavorable factors, which will be discussed briefly in section 2.1.3.

Recommender systems have been used widely in the last twenty years, especially in e-commerce based websites (Schafer, Konstan, & Riedl, 2001). With appropriate recommender algorithms, it is possible to demonstrate a personalized ranking model and promote customer loyalty. For example, the Facebook Company uses the improved collaborative filtering recommender system to generate personalized online store recommendation lists with sub-second processing time (Kabiljo, & Ilic, 2015).

It is clear that 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 a particular end user (Neves, 2015). Recommender systems are more important than before, and integral to the website's accessibility and usability (Ozok & Norcio, 2010). Many website owners are starting to focus on analyzing the performance of recommender systems. However, it was difficult to benchmark existing recommender strategies and algorithms. Herlocker, Konstan, Terveen, and Riedl (2004) pointed out that 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 particular recommender algorithm, it is hard to analyze the correct result without an explicit evaluation goal. Last but not least, many researchers realized that the measurement of accuracy is not enough to tell the performance of a particular recommender system, other relevant aspects also need to be measured in the evaluation process.

There are three primary ways to evaluate recommender algorithms: the offline evaluation, the online evaluation and user-centered evaluation (Ricci, Rokach, & Shapira, 2010). The offline evaluation is focuses on calculating a set of problems by using some specific metrics. However, many researchers realized that offline evaluation is not enough to determine the best recommender algorithm(s). Therefore, the online evaluation and the user-centered evaluation methods have been introduced. The online evaluation tries to measure the user experience, and the user-centered evaluation engages in analyzing the performance of interactive interfaces. Due to time limitation, we only discuss the offline evaluation in this paper.

1.1 Objectives

The purpose of this experiment is to figure out what is the best algorithm(s) that best matches users' interests about movies nowadays. To achieve this goal, it is necessary to identify the characteristics of each recommender approach and determine typical recommender methods in each recommender system.

The dataset selection is also an important aspect. First, it is recommended to use the most updated data and classic dataset in order to compare and contrast information. Moreover, the selected dataset must be available to use at all time. Lastly, it is clear that fake ratings and imaginary users will lower the accuracy of the experiment, so the dataset must act as a natural dataset that is collected by real users with credible ratings.

The final step is to evaluate selected datasets using correct evaluation metrics on a trustable evaluation tool. For evaluation metrics, it is essential to use different metrics other than accuracy to address other problems in the process of evaluation. In this way, we can solve the issues that are mentioned above.

The evaluation tool should create reproducible results and support different evaluation libraries. The RiVaL toolkit is such tool, and it has been selected for evaluation. Therefore, the result of the experiment and its conclusions can be compared with other similar evaluations on the MovieTweetings dataset and MovieLens dataset using the RiVaL toolkit.

1.2 Overview of the Report

In this paper, we will show an evaluation of predictions that are made by different collaborative filtering algorithms on the MovieTweetings latest dataset (on August 7th, 2017) and the MovieLens 1M dataset. The evaluation focused on five different categories of recommendation evaluation metrics (accuracy, decision support metrics, rank-aware top n, recommendation quality and novelty) 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 different categories of recommender system approaches in section 2. In Section 3, 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 4 demonstrates the evaluation setup. Section 5 discusses the evaluation result, as well as analyzation and comparison with different dataset. In section 6, we explain three problems that we encountered during the experiment. The last section talks about future works and conclusions.

II. Recommender system classification

Recommender Systems collect explicit and/or implicit information from its users' preference for a set of items (Bobadilla, Ortega, Hernando, & Gutiérrez , 2013). The term expressly information refers to information such as users' ratings and likes while the implicit information refers to users' behaviors such as searched travel destinations, songs that were played, game that were played and so on.

Recommender systems may also use clients' personal information such as gender, nationality, and languages or prefered activities(posts, users followed or places traveled) on social media to provide predictions and recommendations. It is also necessary for system designers to balance all factors such as surprising, accurate recommendations and recommendation quality.

In this section, we will describe three common recommender system methodologies in detail. We will explain popular algorithms as well as the advantages and disadvantages of each recommender method.

2.1 Collaborative Filtering

The collaborative filtering recommender system would be one of the most successful and prevalent recommendation techniques in the world (Herlocker et al., 1999). 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 (Yang, Guo, Liu, & Steck, 2014).

The collaborative filtering recommender system can be divided into two broad categories: the memory-based and model-based methods (Zhang & Pu, 2007). The memory-based technique recommends items for a particular 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. It is not easy to weigh similar user groups due to the dynamic nature of users. Similarly, the item-based collaborative filtering method filters items from a subset of 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 (Walunj & Sadafale, 2013). 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 such as the cluster model, Bayesian network models, and so on. The model-based recommendation system has faster prediction speed because it only queries the model instead of the whole dataset. One of the most popular algorithms of the model-based recommendation system is matrix factorization. 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 (Koren, Bell, & Volinsky, 2009). Industry level recommender systems often mix both the memory-based and model-based approach to improve performance.

2.1.1 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 one demonstrates the relationship of all notations.

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 ⊆ U
Li	Items left in the itemset. Lu ⊆ I
Nu/Ni	Selected neighbor user-set/itemset for the user u. 1.Nu ⊆ Au 2.Ni ∩ Li = ∅
Tn	The top-n recommendation made by the algorithm. 1. Tn ⊆ Li 2. Tn ∩ Ru = ∅

Table 1. Collaborative Filtering notation and explanation

2.1.2 Collaborative filtering Algorithms

We used three recommender methods: Generic user-based recommender, Generic item-based recommender, and SVD recommender. In this section, we will explain in detail the above algorithms that were used in the evaluation process.

2.1.2.1 User-based and Item-based recommender

The user-based recommender provides predictions by finding the most similar set of users. Zhang and Pu (2009) explained that the typical prediction process of this approach 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. The formula used by user-based recommenders to predict the item for a specific user is:

$$\text{predict(u,i)}=\overline{R}\text{u}+\frac{\Sigma n\subset Nu(u)*S(u,n)*(Rni-\overline{R}u)}{\Sigma n\subset Nu(u)*S(u,n)}$$

Equation 1. User-based prediction

Both item-based and user-based recommenders use a similar selected S to create neighbor users or items, where Rui is the rating of the user u for an item i, and Ru calculates the average recommendation value for a user u. Other symbols can be found in section 2.1.1.

Koenigstein and Koren (2013) claimed that sometimes 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. As mentioned before, 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. Last but not least, 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. The algorithm that we used in the item-based recommendation is called GenericItemBasedRecommender. The equation of the item-based recommender is:

$$\text{predict(u, i)} = \frac{\Sigma Ni(u)S(u,i).Rui}{\Sigma Ni(u)S(u,i)}$$

Equation 2. Item-based recommender

Similarity
We used five different similarity metrics in the evaluation process. The term S(u,i) can represent pearson's correlation, spearman correlation, euclidean distance, tanimoto coefficient or uncentered cosine similarity. In this section, we will provide formulas and explanation for each similarity metrics that we used in the evaluation.

The first similarity metric is the Pearson correlations that are introduced by Resnick, Iacovou, Suchak, Bergstrom, and Riedl (1994). The Pearson correlation calculates the similarity between two users, u1 and u2. The Pearson correlation similarity can be defined as:

$$\text{S(u1,u2)} = \frac{\sum_{n=1}^{N}(Ru1-\overline{R}u1)(Ru2i-\overline{R}u2)}{\sqrt{\sum_{n=1}^{N}(Ru1-\overline{R}u1)^2 \sum_{n=1}^{N}(Ru2-\overline{R}u2)^2}}$$

Equation 3. The Pearson correlation

Cosine similarity assuming two users, u0 and u1, are two vectors. The cosine similarity calculates the cosine value of the angle between u1 and u2. The term "uncentered" means that there is no intersection between two vectors. The formula of uncentered cosine similarity is:

$$S(u1,u2)= \frac{Ru1*Ru2}{||Ru1||Rui2||}$$

Equation 4. The uncentered cosine similarity

Tanimoto coefficient calculates the similarity by looking at the intersection or overlap part.Tanimoto coefficient treats each rating as a "binary" value. Tanimoto coefficient does not care about the value of actual ratings; it only cares whether the preference exists or not. The value of Tanimoto coefficient is in [0,1].

The Spearman rank similarity and the Pearson correlation are similar. The Spearman rank sorts each rating and gave them a ranking, and then it compares the relative ranking of rating values.

The Euclidean distance algorithm is the simplest method for similarity calculation. The Euclidean distance cannot be negative. The value would be zero if two items are identical and the measured value of Euclidean distance can be very high if two items are not similar.

2.1.2.3 SVD recommenders

Model-based recommender system commonly uses matrix factorization to generate predictions (Bokde, Girase, & Mukhopadhyay, 2015). 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. The SVD recommenders are popular matrix factorization methods. SVD recommenders provide good scalability with predictive accuracy; moreover, SVD recommenders are more flexible in various real-life situations. SVD recommenders are distinct by factorization types. In our experiment, we used three types of factorization: FunkSVD, RatingSVD, and SVD++.

Funk (2006) claimed that the FunkSVD is an incremental SVD algorithm that uses gradient descent to shorten the processing time.The algorithm iterates over all ratings in the training set and predicts the Ru, and then calculates all prediction errors. The prediction formula can be defined as:

$$\text{Ru}= \mu + bu + bi + qT_i p\text{u}$$

Equation 5 FunkSVD prediction

The term bu and bi represent bias of user and item, respectively, pu means factors. If the user u is not given, then the bu, bi, and qi should be zero.

SGD stands for stochastic gradient descent; it is an optimized version of gradient descent. Rating SGD includes user and item biases. It can training data faster. On the other hand, the SVD++ method was implemented based on SGD matrix factorization. Koren (2008) explained that the SVD++ method combines the latent factor model and neighbourhood-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.

2.1.3 Identified issues

Memory-based methods have three identified issues: sparsity, scalability, and the cold-start problem (Neves, 2015). The sparsity problem talks about shortages in a data set. 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. (2013) 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 (Rashid, 2008). 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.

Bokde et al. (2015) claimed that the model-based recommender system might be expensive to build and that such model and the loss of information in SVD are two significant drawbacks. Gantner (2011) pointed out that at the industry level, the combination of collaborative filtering methods solved the issue of sparsity. Nevertheless, such combination is extremely complex and difficult to implement.

2.2 Content-based filtering recommender system

Many website owners take advantage of the content-based filtering recommendation system to show personalized information to their users from the large database (Ricci et al., 2011). Content-based filtering recommender algorithms utilize inputs from a customer's past activities to generate a list of recommended items. Heuristic methods or classification algorithms, such as rule induction, decision trees, association rules, linear classifiers, and probabilistic methods, have been employed in the content-based filtering techniques (Bobadilla et al., 2013). In addition, the rising trends of the use of social media have led content-based filtering recommender systems to become more important than before (Neves, 2015). Website users present different types of contents regarding their interests and hobbies to find new friends and build social relationships.

One identified contribution of the content-based filtering recommendation method is the reduced cold start problem. Neves (2015) summarized two unfavored aspects of the content-filtering recommender system. The first issue addressed was the limitation of content analysis. The difficulty of retrieving and analyzing automated content from different file formats may reduce the recommendation performance and increase the cost of computation. Overspecialization is another significant barrier. Overspecialization means that the recommender algorithm only recommends items that are similar to items that the user rated or liked. That is, the recommender algorithm cannot recommend "surprising items." An overspecialized recommender system has high accuracy but may be useless. For example, if there is a supermarket recommender that only recommends bread and eggs, it is a futile recommender because customers will buy bread and eggs without the need of a recommender system. Therefore, the ability to provide surprising item is important and must be considered in the evaluation.

2.3 Hybrid filtering recommender systems

Hybrid filtering methods have been developed to reduce the weakness of content-based and collaborative filtering strategies. As mentioned before, the content-based method increased the prediction quality and lowered the cold start and sparsity problems discussed in section 2.1.3, while collaborative filtering methods overcame the limitations of content-based filtering. A hybrid recommender system is a combination of multiple recommendation techniques that aims to create some synergy between the techniques.

There are different methods to integrate content-based and collaborative filtering models. Robin (2007) identified seven different hybridization strategies. The first one is weighted, which combines different recommendation scores numerically. The second method is called switching, which lets the recommender system select from a set of recommendation components and employs the selected method. The third fashion demonstrates all recommendation from various recommender methods together, named mix hybridization. Moreover, feature combination method combines all features from various knowledge sources and applies those features into one single algorithm, while the feature augmentation method uses one recommendation algorithm to calculate features from knowledge sources and save the output as the input of next technique. The meta-level method is similar to the feature augmentation strategy; it constructs a model from a single recommendation algorithm and serves the model as the input of next algorithm. The last method constructs a priority scale for each recommender, where the lower ranked recommenders can refine higher ranked recommenders.

Robin (2007) verified that most Hybrid recommendation systems improved recommendation performance successfully and reduced the issue of the cold start problem. However, hybrid recommendation techniques are significantly different. Some techniques that discussed above may not perform properly for some particular tasks. Since the hybrid filtering recommendation system uses both content-based filtering and collaborative filtering methods, it is necessary to find the balance between different recommendation strategies.

III. Evaluation design and related works

The focus on evaluation and benchmarking has been increased in past decades (Cremonesi, Said, Tikk, & Zhou, 2016). It is important to understand and identify all required evaluation components to perform the evaluation correctly and effectively. There are three significant evaluation elements: tasks, datasets, and metrics. Those units must be defined clearly during the evaluation designing phase. In this section, we identified two user tasks that need to be achieved. We provided solutions for three identified problems related to three evaluation elements. We clarified our offline evaluation design process in detail, and this includes cross-validation, the process of dataset selection, a brief discussion about the Apache Mahout framework and the RiVaL toolkit, as well as different evaluation metrics and their impacts.

3.1 Solutions to evaluation and benchmarking problems

Several barriers related to evaluation components have been determined. Herlocker et al. (2014) pointed out the disadvantages of ambiguity evaluation goals and tasks. Su and Khoshgoftaar (2009) classified a set of problems related to dataset selection. Said and Bellogín (2014) introduced an evaluation tool to avoid obtaining incomparable results.

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 significant change in results. Our solution is to use the RiVaL toolkit, a tool that can perform a cross-platform evaluation. Therefore, we can compare results generated by different frameworks effectively.

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 (i.e., the sparsity and scalability problem). Some of the biases are related to the data itself; these includes synonymy, gray sheep, and shilling attacks. 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. It is clear that data synonymy and shilling attacks have a significant impact on the reliability and quality of recommendation, while gray sheep users are undesired. However, this problem can be solved by hybrid filtering methods (Campos, Fernández-Luna, Huete, & Rueda-Morales, 2009).

Lastly, the format of the dataset may be a potential issue for evaluation. Data are collected from different sources and organized in different ways. Therefore, datasets may need some modification before use. In our design, we have analyzed all the above problems. Our solution is to use the MovieLens series and MovieTweetings datasets. This way, all the data are in nature and in the same format. Also, most of the problems mentioned above can be minimized. More details can be found in section 3.4.1.

3.2 Goal and tasks

The evaluation goal can differ because recommender systems can be designed in different ways and with different algorithms. Therefore, we need to identify which strategy is better in a particular situation. Even though we already have a good algorithm, we should improve its performance. First of all, 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. Konstan (2017) 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 and the surprising level of recommendations. Other researchers argued that the most important aspect of a recommender system is user satisfaction. As a result, a large number of online evaluations such as online A/B tests, user surveys, and log analysis have been performed to understand user behavior. 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, quality, and novelty, which will be discussed in section 3.2.3.

Herlocker et al. (2014) argued that the first step of the evaluation process is to understand user tasks that the system must achieve. It is necessary to identify the most significant tasks when we evaluate recommender algorithms in ways that can benefit users. 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, Herlocker et al.(2014) stated that in some situations (e.g., legal cases), 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. 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. Herlocker et al. (2014) explained that 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.

3.3 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, and it is also known as hidden data evaluation (Ekstrand & Konstan, 2017). 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, it is necessary to use K-fold cross-validation, which separates data into multiple splits (Ekstrand & Konstan, 2017). 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. In our evaluation design, we set K equals five. This way, the possibility of picking the hardest or easiest users will become extremely small.

With K-fold cross-validation, the statistical validity of the evaluation is enhanced. It also provides efficient and trustable results and does not require user participation. The only drawback is we need to be aware of the computation power when we evaluate using large datasets and complex algorithms.

3.4 Evaluation components

The fundamental components of evaluation are datasets, recommendation algorithms, and evaluation tool. In this section, we introduced a set of existing datasets and their limitations, as well as RiVaL toolkit and the Apache Mahout library. We clarified advantages of MovieTweetings and Movielens datasets, and metrics we used in the experiment.

3.4.1 Datasets

In this section, we talked about some characteristics of MovieTweetings dataset, as well as drawbacks of other classic datasets. We discussed datasets that we evaluated in the experiment at the end.

MovieTweetings

MovieTweetings is a new dataset collection of movie ratings that has been updated since 2013. The original dataset was integrated from the IMDb website and has a rating scale from one to ten. The data is collected from the IMDb iOS mobile application and Twitter API every day. The Movietweetings dataset uses similar formats and data structures as other popular and commonly used datasets such as MovieLens series.

The IMDb website allows anyone to rate a movie with a valid account, thus the database has a large volume of different users and movies. According to Dooms, Pessemier, and Martens (2014), the number of ratings per user differs from 1 to 305 because there is no modification of the data. There are two dataset versions: latest data and snapshots. The latest data contains all up-to-date ratings. Moreover, just like MovieLens, the Snapshots version has fixed portions of the dataset in different sizes to stimulate and ensure experimental reproducibility.

MovieTweetings dataset 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 with the Movietweetings dataset 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 (2016), 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.

Classic datasets
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 (Nathanson, Bitton, & Goldberg, 2007), as well as the Book-Crossing Dataset (Ziegler, Mcnee, Konstan, & Lausen, 2005).The above two datasets are in divergent domains; it is hard to compare them with the most recent MovieTweetings dataset.

The Movielens dataset series (Harper & Konstan, 2016) contributed a lot in in the field of recommender systems evaluation. Cremonesi et al. (2016) claimed that the data collected from movies, such as MovieLens, is important in recommender systems research, so we decided to use the MovieLens dataset in order to contrast and compare. There are several versions of MovieLens datasets. We observed two datasets that similar with the MovieTweetings latest dataset.The first one is MovieLens Latest Datasets, which contains twenty-six million ratings and was updated on August 2017. The second one is MovieLens 1M datasets, which contains one million ratings and was last updated data in February 2003. 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 2 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. Dooms, Bellogín, and Pessemier (2016) explained that no users and movies had been filtered in the MovieTweetings dataset, 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. The only thing left which defines the formula to calculate density can be expressed as follows:

$$rating\ density\ =\ 100 \times \frac{Available\ Ratings}{All\ Users * All\ Movies}$$

Equation 6. Rating density

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

Table 2 Datasets information

3.4.2 RiVaL Toolkit

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. Said and Bellogín (2014) states that the benchmarking process can be divided into four stages: data splitting, item recommendation, candidate item generation, and performance measurement.

The selected data has been divided into training and test datasets in the data-splitting stage, and the recommendation list was generated in the item-recommendation phase. One thing that needs to be noted is the RiVaL toolkit was not designed as a framework; it does not have any resource library. Therefore, it cannot perform candidate item generation. Nevertheless, the third stage can be performed by Mahout, MyMediaLite, and LensKit, three integrated open source recommendation frameworks. Lastly, the program will measure the performance of the produced scores.

Due to time limitation, we only used Apache Mahout in the candidate item generation stage. As mentioned before, We selected GenericItemBased and GenericUserBased (also know as item-based Knn and user-based Knn) with five similarity classes (Pearson's Correlation, Uncentered Cosine Similarity, Spearman Rank, Euclidean Distance and Tanimoto Coefficient). We also developed settings for Matrix Factorization with three different factorization methods(FunkSVD, SVD++ factorizer, and Rating SGD) in Apache Mahout framework.

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.

Neves (2015) pointed out that the RiVaL toolkit must work with good datasets where all users have a possible recommendation. Therefore, if the dataset has a sparsity problem, the RiVaL might pause to throw exception errors.

3.4.3 Apache Mahout

Apache Mahout is an open-source and industry-level library. It has high scalability and supports large dataset processing in a distributed fashion. The Mahout library has a large number of well-developed recommendation methods and evaluation metrics. Also, the Mahout developer community has many active users and interesting topics, and it provides helpful support documentation.

3.4.4 Evaluation Metrics

In order to observe the best recommender system(s), several different aspects need to be evaluated. We identified the following five categories: accuracy prediction metrics, decision support metrics, rank-aware metrics, recommendation quality, and surprising level. We defined seven metrics that we employed in the experiment, as well as metrics that we decided not to use in the evaluation.

Accuracy prediction metrics

Accuracy prediction metrics are often considered as an intuitive approach to determine how recommender systems work (Ekstrand & Konstan, 2017). Those metrics calculate errors between predicted ratings and real ratings. In the experiment, we applied RMSE and MAE as accuracy prediction metrics.

Root mean square error (RMSE) is a traditional method to evaluate the accuracy. It measures the error in the rating predictions, and it places more emphasis on significant errors. RMSE employs the square to ensure that the measurement has the same unit as the primitive ratings. Let Pi be ratings predicted by the recommender algorithm, and the formula of RMSE can be defined as:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Pi - Rui)^{2}}$$

Equation 7. RMSE

MAE stands for mean absolute error. It calculates the divergence of average absolute value between prediction and actual marks. MAE is simple and easy to understand. We use MAE to calculate large-scale datasets like MovieTweetings. The equation can be expressed as:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |Pi - Rui|$$

Equation 8. MAE

Decision support metrics

Decision support metrics measure the ability of how a recommender system can help users make good decisions. Precision and Recall are two commonly used metrics; they also are classified as information retrieval metrics (Ekstrand & Konstan, 2017). In this case, we view recommender systems as information retrieval tasks that we want to find all items that are predicted to be "good."

Precision@N appraises if the recommendation algorithm can return the most useful result that does not waste user's time. Precision is a true positive rate (Gunawardana and Shani, 2009). Let Ir donates the number of items that are relevant, with Is being a number of selected items. Precision can be defined as:

$$\text{Precision} = \frac{Ir \cap Is}{Is}$$

Equation 9. Precision@N

Moreover, Recall@N determines the percentage of relevant items are selected; it checks if the algorithm was not missing any useful stuff. We selected Precision@N in our experiment because the above two metrics are similar. Thus, we only need to pick one.

Rank-aware top n

Users are going to pay more attention to the top and ignore the rest of the list. That is why we need to focus on rank-aware evaluation metrics; we want to know how good the recommender system is at modeling relative preference (Ekstrand & Konstan, 2017). We want to discover the overall accuracy at the top of the recommendation lists. In addition, we want to weigh items at the top of the recommendation list more heavily. Therefore, MAP and normalized Discounted Cumulative Gain (nDCG) have been selected, where MAP is the average of all precision values and nDCG measures the normalized gain by best possible gain since different users have different ratings and different gains. The range of nDCG@N is between 0 and 1, where one means perfect (Baltrunas et al., 2010). The formula of nDCG can be defined as

$$nDCG_n^u = \frac{DCG_n^u}{IDCG_n^u}$$

Equation 10. nDCG@rank n

Recommendation quality

Coverage is a metric to evaluate recommendation quality. According to Ge et al. (2010), there are two types of coverage. The first one is prediction coverage, which measures the percentage of products for which a recommender can make a prediction. The second type is called "catalogue coverage," which calculates the percentage of effective recommendations. We used the first definition and constructed weighted prediction coverage. Let Ip be the itemset that can make predictions and let r(x) be the usefulness, so the formula of weighted prediction coverage can be defined as:

$$\text{weighted prediction coverage}=\frac{\Sigma i \varepsilon Ip\ r(i)}{\Sigma j \varepsilon I\ r(j)}$$

Equation 11.weighted prediction coverage

Surprising level

The level of unexpectedness can be evaluated by serendipity and novelty. Serendipity has many definitions. Many researchers state that serendipity is a measurement of how surprising a recommender list is. We abandoned this metric because, if a user does not have too many ratings, then the serendipity will become quite low. Therefore, serendipity cannot work properly with selected datasets. Novelty is another surprise level measurement metric. Higher novelty values represent that less popular items are being recommended, and thus less well-known items are likely being surfaced for users. We performed popularity-based item novelty in our experiment.

4. 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 neighbor 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 neighbor 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 (more detail can be found in section 3.3). 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 (more details can be found in section 3.4.4.). 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.

5. Result and discussion

In general, the total execution time for the MovieTweetings latest dataset is 1379155 seconds (about 383.09 hours), and Figures 1, 2, and 3 demonstrate the performance of this dataset under different settings. The evaluation time for MovieLens 1M is 1539738 seconds (about 427.7 hours); the result can be found in Figures 4, 5, and 6. Furthermore, the total running time for MovieLens 100k is 30927 seconds (approximately 8.590 hours), and the performance can be checked in Figures 7, 8, and 9. According to the results, we observed that as the size of the dataset increased numerically, the algorithm execution time will increase exponentially. Moreover, the user-based recommender has the highest running time, while the overall user-based recommenders have 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. We 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 were not changing when we set the cutoff from 10 to 50 and let neighbor size remain at 100. Therefore, it is clear that the value of the cutoff cannot affect the value of accuracy metrics when neighbor 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 size of the neighbor 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 neighbor 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 are using 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 as MAP will drop about 50 to 60%. 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 size of the neighbor did not affect the performance. The above result shows that the size of the neighbor does not have a connection with item-based recommenders.

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 doesn't. 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.

Figures 10-13, 14-17, and 18-21 show the average of each metrics for user-based, item-based, and SVD recommenders in each dataset respectively. The first fascinating finding is that the MovieTweetings dataset had the highest overall accuracy but the lowest value among other metrics. 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 provided the most valuable and appropriate predictions at the top of the list and the most help to users in making good decisions.

6. **Lessons learned**

The first lesson that we learned is that in some cases, the unmodified MovieTweetings latest dataset may not be the most desirable dataset for evaluation. The reason for that is sometimes the RiVaL toolKit throws the "No Such User Exception" error when we employ the MovieTweetings dataset, especially for SVD recommenders. There are two causes for the problem: the first reason is that RivaL was designed and tested with good datasets that do not have sparsity problems; the second reason is the Movie
Tweetings latest dataset is unfiltered, so some users only have a few ratings included in the dataset. When the cross-validation technique divides the dataset into separate datasets, those users that appeared in the training dataset may not have ratings in the testing dataset. Thus, the RiVaL toolKit cannot simulate predictions, and if the algorithm detected that either user is known to be nonexistent in test datasets, the error would occur. The solution here is either modifying the latest dataset or using MovieTweetings snapshots version as an alternative.

The second thing that we learned is the current RiVaL ToolKit may not work well with larger datasets, such as MovieLens 20M. We encountered the "OutOfMemoryError: Java heap space" error at the beginning of the evaluation.The cause of the problem is that the dataset is too large, and the space in the Java Virtual Machine (JVM) is not enough to run the program. Typically, a Java memory area in the JVM can be divided into two areas: the heap space and permgen, and each zone has a limited size. If processed data exceeds the default memory space and the algorithm tries to input more data into the JVM memory, the JVM requires more heap space in order to run correctly. The JVM will suspend the evaluation algorithm and throw out the memory error. There are many ways to solve this issue. The simplest solution is to add more space into JVM because the size of the heap space can be customized by specifying the JVM parameter called "Xmxn". Xmxn means the maximum memory space in bytes. It is important to know that the value of Xmxn must be larger than two megabytes. However, the above solution cannot solve the problem permanently. The program still may stop when the computer reaches its maximum available RAM. A more desirable solution would be modifying the existing code to reduce computation costs. In the experiment, we used both methods to solve the above error. We set Xmxn to 10g and changed some global variables to local and static variables.

We also faced a garbage collection issue called "GC overhead limit exceeded"; the above issue also related to memory space, which means that the JVM uses all its memory to execute the garbage collection so it cannot run the program. This only happened during the MovieTweetings latest dataset evaluation. A potential cause for the problem is the big number of users and movies. With the five fold cross-validation, the algorithm creates too many temporary objects in the JVM (some objects may be counted more than five times). Therefore, we found two solutions: The first one is turning off the warning by typing -XX:-UseGCOverheadLimit, and the second one is using the mark-and-sweep garbage collector, which is a concurrent low pause collector. From the above issue, we understood the importance of using the filtered dataset in the recommender system evaluation. Therefore, we highly recommend using well-filtered datasets on RiVaL toolKit.

7. Conclusion and Future Work

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 have the ability to 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.

7. References

[1] Zhang, J., & Pu, P. (2007). A recursive prediction algorithm for collaborative filtering recommender systems. Proceedings of the 2007 ACM conference on Recommender systems - RecSys 07. doi:10.1145/1297231.1297241

[2] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994). GroupLens:An open architecture for collaborative filtering of netnews. Proceedings of the 1994 ACM conference on Computer supported cooperative work - CSCW 94. doi:10.1145/192844.192905

[3] Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems, 22(1), 5-53. doi:10.1145/963770.963772

[4] Said, A., and Bellogín, A. (2014). Comparative recommender system evaluation. Proceedings of the 8th ACM Conference on Recommender systems - RecSys 14. doi:10.1145/2645710.2645746

[5] Said, A., & Bellogín, A. (2014). RiVaL–A Toolkit to Foster Reproducibility in Recommender System Evaluation. Proceedings of the 8th ACM Conference on Recommender systems - RecSys 14. doi:10.1145/2645710.2645712

[6] Dayan, A., Katz, G., Biasdi, N., Rokach, L., Shapira, B., Aydin, A., . . . Fishel, R. (2011). Recommenders benchmark framework. Proceedings of the fifth ACM conference on Recommender systems - RecSys 11. doi:10.1145/2043932.2044003

[7] Gantner, Z., Rendle, S., Freudenthaler, C., & Schmidt-Thieme, L. (2011). MyMediaLite: A Free Recommender System Library. Proceedings of the fifth ACM conference on Recommender systems - RecSys 11. doi:10.1145/2043932.2043989

[8] Cremonesi, P., Said, A., Tikk, D., & Zhou, M. X. (2016). Introduction to the Special Issue on Recommender System Benchmarking. ACM Transactions on Intelligent Systems and Technology, 7(3), 1-4. doi:10.1145/2870627

[9] Doerfel, S., Jäschke, R., & Stumme, G. (2016). The Role of Cores in Recommender Benchmarking for Social Bookmarking Systems. ACM Transactions on Intelligent Systems and Technology, 7(3), 1-33. doi:10.1145/2700485

[10] Dooms, S., Bellogín, A., & Pessemier, T. (2016). A Framework for Dataset Benchmarking and Its Application to a New Movie Rating Dataset. Retrieved July 03, 2017, from http://dl.acm.org/citation.cfm?id=2751565

[11] Dooms, S., & Martens, L. (2014). "Harvesting movie ratings from structured data in social media" by Simon Dooms and Luc Martens with Ching-man Au Yeung as coordinator. ACM SIGWEB Newsletter, (Winter), 1-5. doi:10.1145/2559858.2559862

[12] Thung, F., Oentaryo, R. J., Lo, D., & Tian, Y. (2017). WebAPIRec: Recommending Web APIs to Software Projects via Personalized Ranking. IEEE Transactions on Emerging Topics in Computational Intelligence, 1(3), 145-156. doi:10.1109/tetci.2017.2699222

[13] Walunj, S. G., & Sadafale, K. (2013). An online recommendation system for e-commerce based on apache mahout framework. Proceedings of the 2013 annual conference on Computers and people research - SIGMIS-CPR 13. doi:10.1145/2487294.2487328

[14] Kabiljo, M., & Ilic, A. (2015, June 2). Recommending items to more than a billion people. Retrieved September 29, 2017, from https://code.facebook.com/posts/861999383875667/recommending-items-to-more-than-a-billion-people/

[15] Nathanson, T., Bitton, E., & Goldberg, K. (2007). Eigentaste 5.0:constant-time adaptability in a recommender system using item clustering. Proceedings of the 2007 ACM conference on Recommender systems - RecSys 07. doi:10.1145/1297231.1297258

[16] Ge, M., Delgado-Battenfeld, C., & Jannach, D. (2010). Beyond accuracy:Evaluating Recommender Systems by Coverage and Serendipity. Proceedings of the fourth ACM conference on Recommender systems - RecSys 10. doi:10.1145/1864708.1864761

[17] Campos, L., M.Fernández-Luna, J., F.Huete, J., & Rueda-Morales, F. (2010, April 11). Combining content-based and collaborative recommendations: A hybrid approach based on Bayesian networks. Retrieved September 18, 2017, from http://www.sciencedirect.com/science/article/pii/S0888613X10000460

[18]Neves, G. A. (2015, June 29). Empirical study of the behavior of several Recommender System methods on SAPO Videos. Retrieved June 25, 2017, from https://sigarra.up.pt/fcnaup/pt//pub_geral.pub_view?pi_pub_base_id=36000

[19] Gunawardana, A., & Shani, G. (2009). A Survey of Accuracy Evaluation Metrics of Recommendation Tasks. The Journal of Machine Learning Research, 10, 2935–2962. doi: 10.1145/1577069.1755883

[20] Baltrunas, L., Makcinskas, T., & Ricci, F. (2010). Group Recommendations with Rank Ag- 8 gregation and Collaborative Filtering. In Acm conference on recommender systems (pp. 119–126)

[21] Channamsetty, S., & Ekstrand, M. (2017, May 15). Recommender Response to Diversity and Popularity Bias in User Profiles. In Proceedings of the 30th International Florida Artificial Intelligence Research Society Conference. Retrieved September 8, 2017. doi:10.18122/B2W012

[22] Bokde, D., Girase, S., & Mukhopadhyay, D. (2015). Matrix Factorization Model in Collaborative Filtering Algorithms: A Survey. Procedia Computer Science, 49, 136-146. doi:10.1016/j.procs.2015.04.237

[23] Vargas, S., & Castells, P. (2011). Rank and relevance in novelty and diversity metrics for recommender systems. Proceedings of the fifth ACM conference on Recommender systems - RecSys 11. doi:10.1145/2043932.2043955

[24] Zhang, Y. C., Séaghdha, D. Ó, Quercia, D., & Jambor, T. (2012). Auralist. Proceedings of the fifth ACM international conference on Web search and data mining - WSDM 12. doi:10.1145/2124295.2124300

[25] Seminario, C., & Wilson, D. (2012). Case Study Evaluation of Mahout as a Recommender Platform. Retrieved September 5, 2017, from

https://www.researchgate.net/profile/Carlos_Seminario/publication/259261455_Case_study_evaluation_of_Mahout_as_a_recommender_platform/links/560309c708ae596d2591c57d.pdf

[26] Beel, J., Genzmehr, M., Langer, S., Nürnberger, A., & Gipp, B. (2013). A comparative analysis of offline and online evaluations and discussion of research paper recommender system evaluation. Proceedings of the International Workshop on Reproducibility and Replication in Recommender Systems Evaluation - RepSys ’13, 7–14.

[27] Ozok, A. A., Fan, Q., & Norcio, A. F. (2010). Design guidelines for effective recommender system interfaces based on a usability criteria conceptual model: Results from a college student population. Behaviour and Information Technology, 29(1), 57-83. DOI: 10.1080/01449290903004012

[28] Ricci, F., Rokach, L., & Shapira, B. (2010). Introduction to Recommender Systems Handbook. Recommender Systems Handbook, 1-35. doi:10.1007/978-0-387-85820-3_1

[29] Schafer, J. B., Konstan, J. A., & Riedl, J. (2001). E-Commerce Recommendation Applications. Retrieved September 13, 2017, from https://link.springer.com/article/10.1023/A:1009804230409

[30] Yang, X., Guo, Y., Liu, Y., & Steck, H. (2014, March). A survey of collaborative filtering based social recommender systems. Computer Communications, 41, 1–10.

[31] Koren, Y., Bell, R., & Volinsky, C. (2009). MATRIX FACTORIZATION TECHNIQUES FOR RECOMMENDER SYSTEMS. COVER FEATURE. Retrieved September 7, 2017, from https://datajobs.com/data-science-repo/Recommender-Systems-%5BNetflix%5D.pdf.

[32] Koenigstein, N., & Koren, Y. (2013). Towards scalable and accurate item-oriented recommendations. Proceedings of the 7th ACM conference on Recommender systems - RecSys 13. doi:10.1145/2507157.2507208

[33] Koren, Y. (2008). Factorization meets the neighborhood. Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08. doi:10.1145/1401890.1401944

[34] Dooms, S., Pessemier, T., & Martens, L. (2013). MovieTweetings: a Movie Rating Dataset Collected From Twitter. Retrieved September 22, 2017, from http://crowdrec2013.noahlab.com.hk/papers/crowdrec2013_Dooms.pdf

[35] Rashid, A. M., Karypis, G., & Riedl, J. (2008). Learning preferences of new users in recommender systems. ACM SIGKDD Explorations Newsletter, 10(2), 90. doi:10.1145/1540276.1540302

[36] Ekstrand, M., & Konstan, J. (2017). Recommender Systems: Evaluation and Metrics by University of Minnesota. Lecture. Retrieved September 17, 2017, from https://www.coursera.org/learn/recommender-metrics

[37] Harper, M., & Konstan, J. A. (2016). The MovieLens Datasets: History and Context. Retrieved September 17, 2017, from http://dl.acm.org/citation.cfm?id=2827872&CFID=810425874&CFTOKEN=59686369

[38] Su, X., & Khoshgoftaar, T. (2009, October 27). A Survey of Collaborative Filtering Techniques. Advances in Artificial Intelligence Volume 2009. doi:10.1155/2009/421425

[39] Ziegler, C., Mcnee, S. M., Konstan, J. A., & Lausen, G. (2005). Improving recommendation lists through topic diversification. Proceedings of the 14th international conference on World Wide Web - WWW 05. doi:10.1145/1060745.1060754

[40] Kumar, R., Verma, B. K., & Rastogi, S. S. (2014). Social Popularity based SVD Recommender System. International Journal of Computer Applications, 87(14), 33-37. doi:10.5120/15279-4033

[41] Plumbr website (2017). Java heap space. Retrieved September 22, 2017, from https://plumbr.eu/outofmemoryerror/java-heap-space#solution

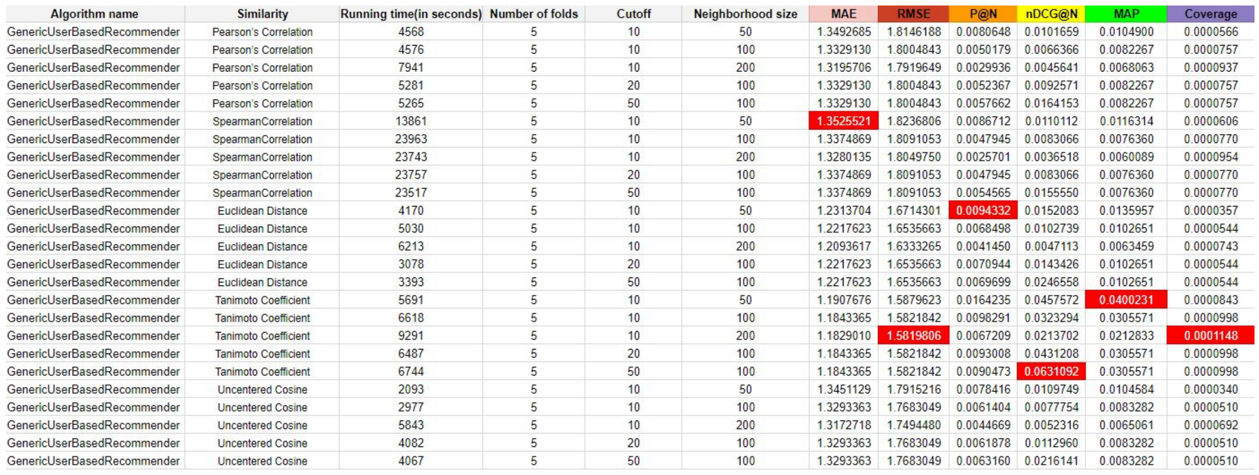
[42] Burke, R. (2007). Hybrid Web Recommender Systems. The Adaptive Web Lecture Notes in Computer Science, 377-408. doi:10.1007/978-3-540-72079-9_12

[43] Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). Recommender systems survey. Knowledge-Based Systems, 46, 109-132. doi:10.1016/j.knosys.2013.03.012

APPENDIX A

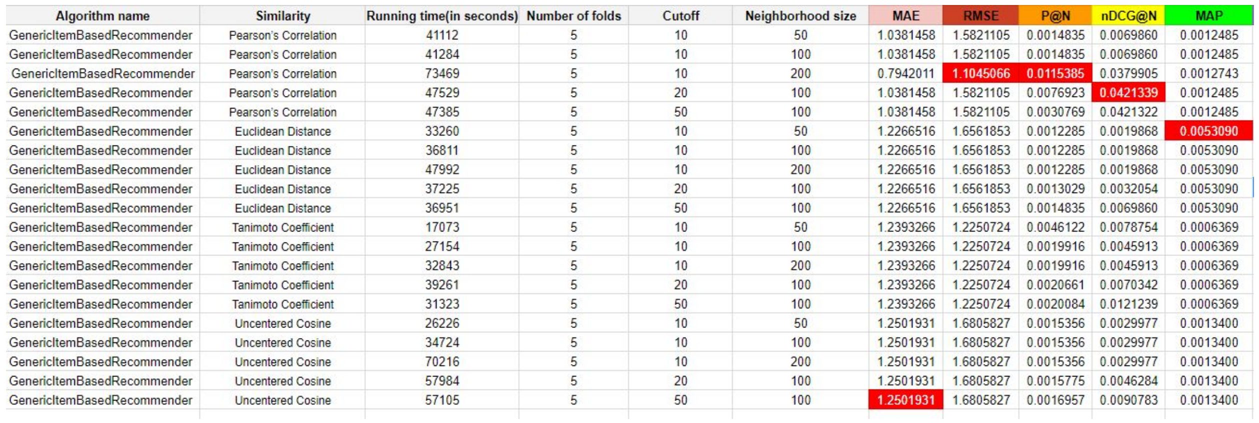
- **MovieTweetings latest**

https://docs.google.com/spreadsheets/d/14AmO-MMUOr1N4GnYSkO2bcSpF-YFo5-9hyrU_tkO-Do/edit#gid=0



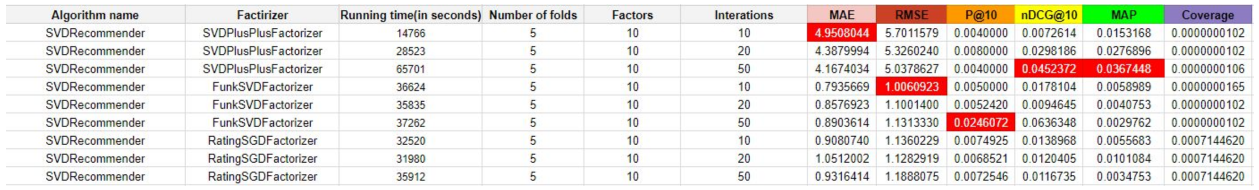
Algorithm name	Similarity	Running time(in seconds)	Number of folds	Cutoff	Neighborhood size	MAE	RMSE	P@N	nDCG@N	MAP	Coverage
GenericUserBasedRecommender	Pearson's Correlation	4568	5	10	50	1.3492685	1.8146188	0.0080648	0.0101659	0.0104900	0.0000566
GenericUserBasedRecommender	Pearson's Correlation	4576	5	10	100	1.3329130	1.8004843	0.0050179	0.0066366	0.0082267	0.0000757
GenericUserBasedRecommender	Pearson's Correlation	7941	5	10	200	1.3195706	1.7919649	0.0029936	0.0045641	0.0068063	0.0000937
GenericUserBasedRecommender	Pearson's Correlation	5281	5	20	100	1.3329130	1.8004843	0.0052367	0.0092571	0.0082267	0.0000757
GenericUserBasedRecommender	Pearson's Correlation	5265	5	50	100	1.3329130	1.8004843	0.0057662	0.0164153	0.0082267	0.0000757
GenericUserBasedRecommender	SpearmanCorrelation	13861	5	10	50	1.3525521	1.8236806	0.0086712	0.0110112	0.0116314	0.0000606
GenericUserBasedRecommender	SpearmanCorrelation	23963	5	10	100	1.3374869	1.8091053	0.0047945	0.0083066	0.0076360	0.0000770
GenericUserBasedRecommender	SpearmanCorrelation	23743	5	10	200	1.3280135	1.8049750	0.0025701	0.0036518	0.0060089	0.0000954
GenericUserBasedRecommender	SpearmanCorrelation	23757	5	20	100	1.3374869	1.8091053	0.0047945	0.0083066	0.0076360	0.0000770
GenericUserBasedRecommender	SpearmanCorrelation	23517	5	50	100	1.3374869	1.8091053	0.0054565	0.0155550	0.0076360	0.0000770
GenericUserBasedRecommender	Euclidean Distance	4170	5	10	50	1.2313704	1.6714301	0.0094332	0.0152083	0.0135957	0.0000357
GenericUserBasedRecommender	Euclidean Distance	5030	5	10	100	1.2217623	1.6535663	0.0068498	0.0102739	0.0102651	0.0000544
GenericUserBasedRecommender	Euclidean Distance	6213	5	10	200	1.2093617	1.6333265	0.0041450	0.0047113	0.0063459	0.0000743
GenericUserBasedRecommender	Euclidean Distance	3078	5	20	100	1.2217623	1.6535663	0.0070944	0.0143426	0.0102651	0.0000544
GenericUserBasedRecommender	Euclidean Distance	3393	5	50	100	1.2217623	1.6535663	0.0069699	0.0246558	0.0102651	0.0000544
GenericUserBasedRecommender	Tanimoto Coefficient	5691	5	10	50	1.1907676	1.5879623	0.0164235	0.0457572	0.0400231	0.0000843
GenericUserBasedRecommender	Tanimoto Coefficient	6618	5	10	100	1.1843365	1.5821842	0.0098291	0.0323294	0.0305571	0.0000998
GenericUserBasedRecommender	Tanimoto Coefficient	9291	5	10	200	1.1829010	1.5819806	0.0067209	0.0213702	0.0212833	0.0001148
GenericUserBasedRecommender	Tanimoto Coefficient	6487	5	20	100	1.1843365	1.5821842	0.0093008	0.0431208	0.0305571	0.0000998
GenericUserBasedRecommender	Tanimoto Coefficient	6744	5	50	100	1.1843365	1.5821842	0.0090473	0.0631092	0.0305571	0.0000998
GenericUserBasedRecommender	Uncentered Cosine	2093	5	10	50	1.3451129	1.7915216	0.0078416	0.0109749	0.0104584	0.0000340
GenericUserBasedRecommender	Uncentered Cosine	2977	5	10	100	1.3293363	1.7683049	0.0061404	0.0077754	0.0083282	0.0000510
GenericUserBasedRecommender	Uncentered Cosine	5843	5	10	200	1.3172718	1.7494480	0.0044669	0.0052316	0.0065061	0.0000692
GenericUserBasedRecommender	Uncentered Cosine	4082	5	20	100	1.3293363	1.7683049	0.0061878	0.0112960	0.0083282	0.0000510
GenericUserBasedRecommender	Uncentered Cosine	4067	5	50	100	1.3293363	1.7683049	0.0063160	0.0216141	0.0083282	0.0000510

Figure 1. MovieTweetings latest (GenericUserBasedRecommender)



Algorithm name	Similarity	Running time(in seconds)	Number of folds	Cutoff	Neighborhood size	MAE	RMSE	P@N	nDCG@N	MAP
GenericItemBasedRecommender	Pearson's Correlation	41112	5	10	50	1.0381458	1.5821105	0.0014835	0.0069860	0.0012485
GenericItemBasedRecommender	Pearson's Correlation	41284	5	10	100	1.0381458	1.5821105	0.0014835	0.0069860	0.0012485
GenericItemBasedRecommender	Pearson's Correlation	73469	5	10	200	0.7942011	1.1045066	0.0115385	0.0379905	0.0012743
GenericItemBasedRecommender	Pearson's Correlation	47529	5	20	100	1.0381458	1.5821105	0.0076923	0.0421339	0.0012485
GenericItemBasedRecommender	Pearson's Correlation	47385	5	50	100	1.0381458	1.5821105	0.0030769	0.0421322	0.0012485
GenericItemBasedRecommender	Euclidean Distance	33260	5	10	50	1.2266516	1.6561853	0.0012285	0.0019868	0.0053090
GenericItemBasedRecommender	Euclidean Distance	36811	5	10	100	1.2266516	1.6561853	0.0012285	0.0019868	0.0053090
GenericItemBasedRecommender	Euclidean Distance	47992	5	10	200	1.2266516	1.6561853	0.0012285	0.0019868	0.0053090
GenericItemBasedRecommender	Euclidean Distance	37225	5	20	100	1.2266516	1.6561853	0.0013029	0.0032054	0.0053090
GenericItemBasedRecommender	Euclidean Distance	36951	5	50	100	1.2266516	1.6561853	0.0014835	0.0069860	0.0053090
GenericItemBasedRecommender	Tanimoto Coefficient	17073	5	10	50	1.2393266	1.2250724	0.0046122	0.0078754	0.0006369
GenericItemBasedRecommender	Tanimoto Coefficient	27154	5	10	100	1.2393266	1.2250724	0.0019916	0.0045913	0.0006369
GenericItemBasedRecommender	Tanimoto Coefficient	32843	5	10	200	1.2393266	1.2250724	0.0019916	0.0045913	0.0006369
GenericItemBasedRecommender	Tanimoto Coefficient	39261	5	20	100	1.2393266	1.2250724	0.0020661	0.0070342	0.0006369
GenericItemBasedRecommender	Tanimoto Coefficient	31323	5	50	100	1.2393266	1.2250724	0.0020084	0.0121239	0.0006369
GenericItemBasedRecommender	Uncentered Cosine	26226	5	10	50	1.2501931	1.6805827	0.0015356	0.0029977	0.0013400
GenericItemBasedRecommender	Uncentered Cosine	34724	5	10	100	1.2501931	1.6805827	0.0015356	0.0029977	0.0013400
GenericItemBasedRecommender	Uncentered Cosine	70216	5	10	200	1.2501931	1.6805827	0.0015356	0.0029977	0.0013400
GenericItemBasedRecommender	Uncentered Cosine	57984	5	20	100	1.2501931	1.6805827	0.0015775	0.0046284	0.0013400
GenericItemBasedRecommender	Uncentered Cosine	57105	5	50	100	1.2501931	1.6805827	0.0016957	0.0090783	0.0013400

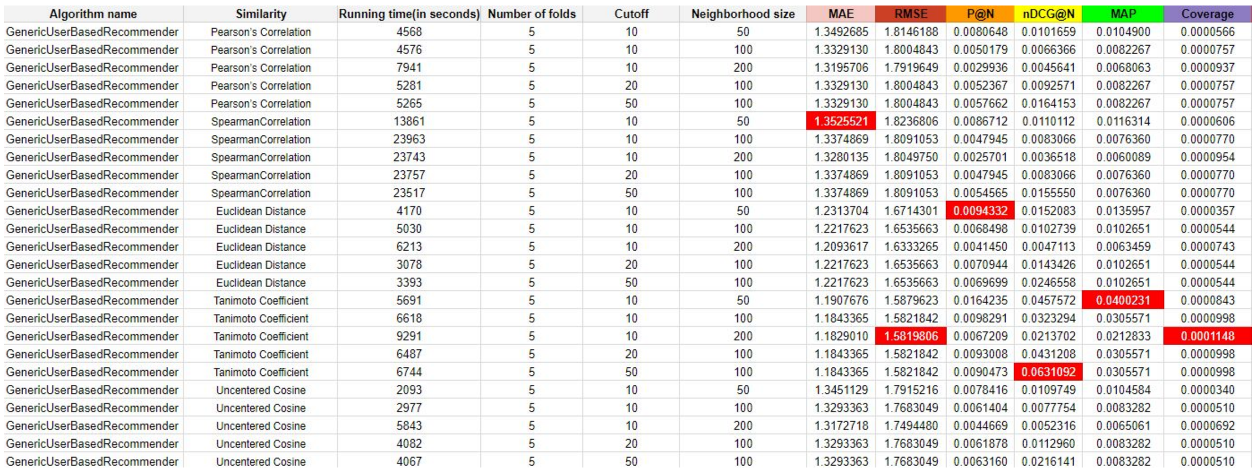
Figure 2. MovieTweetings latest (GenericItemBasedRecommender)



Algorithm name	Factirizer	Running time(in seconds)	Number of folds	Factors	Interations	MAE	RMSE	P@10	nDCG@10	MAP	Coverage
SVDRecommender	SVDPlusPlusFactorizer	14766	5	10	10	4.9508044	5.7011579	0.0040000	0.0072614	0.0153168	0.0000000102
SVDRecommender	SVDPlusPlusFactorizer	28523	5	10	20	4.3879994	5.3260240	0.0080000	0.0298186	0.0276896	0.0000000102
SVDRecommender	SVDPlusPlusFactorizer	65701	5	10	50	4.1674034	5.0378627	0.0040000	0.0452372	0.0367448	0.0000000106
SVDRecommender	FunkSVDFactorizer	36624	5	10	10	0.7935669	1.0060923	0.0050000	0.0178104	0.0058989	0.0000000165
SVDRecommender	FunkSVDFactorizer	35835	5	10	20	0.8576923	1.1001400	0.0052420	0.0094645	0.0040753	0.0000000102
SVDRecommender	FunkSVDFactorizer	37262	5	10	50	0.8903614	1.1313330	0.0246072	0.0636348	0.0029762	0.0000000102
SVDRecommender	RatingSGDFactorizer	32520	5	10	10	0.9080740	1.1360229	0.0074925	0.0138968	0.0055683	0.0007144620
SVDRecommender	RatingSGDFactorizer	31980	5	10	20	1.0512002	1.1282919	0.0068521	0.0120405	0.0101084	0.0007144620
SVDRecommender	RatingSGDFactorizer	35912	5	10	50	0.9316414	1.1888075	0.0072546	0.0116735	0.0034753	0.0007144620

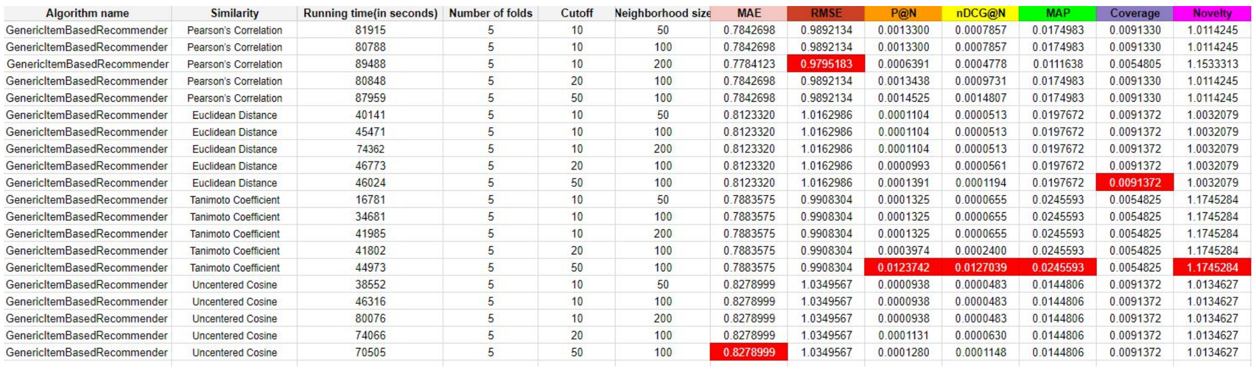
Figure 3. MovieTweetings latest (SVDRecommender)

https://docs.google.com/spreadsheets/d/1PNiV1NpkYfbtHzGrrDxWG8R6nTYc05P9_p4jz5hfWU8/edit#gid=0



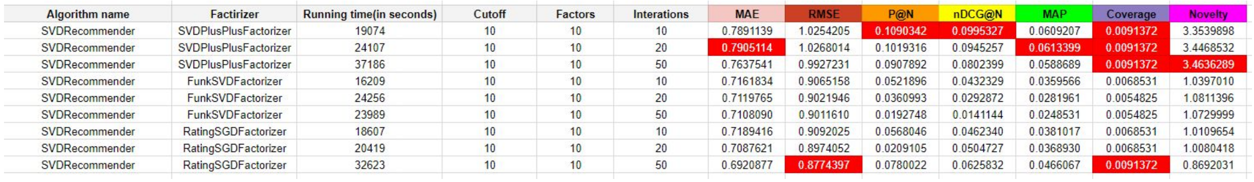
Algorithm name	Similarity	Running time(in seconds)	Number of folds	Cutoff	Neighborhood size	MAE	RMSE	P@N	nDCG@N	MAP	Coverage
GenericUserBasedRecommender	Pearson's Correlation	4568	5	10	50	1.3492685	1.8146188	0.0080648	0.0101659	0.0104900	0.0000566
GenericUserBasedRecommender	Pearson's Correlation	4576	5	10	100	1.3329130	1.8004843	0.0050179	0.0066366	0.0082267	0.0000757
GenericUserBasedRecommender	Pearson's Correlation	7941	5	10	200	1.3195706	1.7919649	0.0029936	0.0045641	0.0068063	0.0000937
GenericUserBasedRecommender	Pearson's Correlation	5281	5	20	100	1.3329130	1.8004843	0.0052367	0.0092571	0.0082267	0.0000757
GenericUserBasedRecommender	Pearson's Correlation	5265	5	50	100	1.3329130	1.8004843	0.0057662	0.0164153	0.0082267	0.0000757
GenericUserBasedRecommender	SpearmanCorrelation	13861	5	10	50	1.3525521	1.8236806	0.0086712	0.0110112	0.0116314	0.0000606
GenericUserBasedRecommender	SpearmanCorrelation	23963	5	10	100	1.3374869	1.8091053	0.0047945	0.0083066	0.0076360	0.0000770
GenericUserBasedRecommender	SpearmanCorrelation	23743	5	10	200	1.3280135	1.8049750	0.0025701	0.0036518	0.0060089	0.0000954
GenericUserBasedRecommender	SpearmanCorrelation	23757	5	20	100	1.3374869	1.8091053	0.0047945	0.0083066	0.0076360	0.0000770
GenericUserBasedRecommender	SpearmanCorrelation	23517	5	50	100	1.3374869	1.8091053	0.0054565	0.0155550	0.0076360	0.0000770
GenericUserBasedRecommender	Euclidean Distance	4170	5	10	50	1.2313704	1.6714301	0.0094332	0.0152083	0.0135957	0.0000357
GenericUserBasedRecommender	Euclidean Distance	5030	5	10	100	1.2217623	1.6535663	0.0068498	0.0102739	0.0102651	0.0000544
GenericUserBasedRecommender	Euclidean Distance	6213	5	10	200	1.2093617	1.6333265	0.0041450	0.0047113	0.0063459	0.0000743
GenericUserBasedRecommender	Euclidean Distance	3078	5	20	100	1.2217623	1.6535663	0.0070944	0.0143426	0.0102651	0.0000544
GenericUserBasedRecommender	Euclidean Distance	3393	5	50	100	1.2217623	1.6535663	0.0069699	0.0246558	0.0102651	0.0000544
GenericUserBasedRecommender	Tanimoto Coefficient	5691	5	10	50	1.1907676	1.5879623	0.0164235	0.0457572	0.0400231	0.0000843
GenericUserBasedRecommender	Tanimoto Coefficient	6618	5	10	100	1.1843365	1.5821842	0.0098291	0.0323294	0.0305571	0.0000998
GenericUserBasedRecommender	Tanimoto Coefficient	9291	5	10	200	1.1829010	1.5819806	0.0067209	0.0213702	0.0212833	0.0001148
GenericUserBasedRecommender	Tanimoto Coefficient	6487	5	20	100	1.1843365	1.5821842	0.0093008	0.0431208	0.0305571	0.0000998
GenericUserBasedRecommender	Tanimoto Coefficient	6744	5	50	100	1.1843365	1.5821842	0.0090473	0.0631092	0.0305571	0.0000998
GenericUserBasedRecommender	Uncentered Cosine	2093	5	10	50	1.3451129	1.7915216	0.0078416	0.0109749	0.0104584	0.0000340
GenericUserBasedRecommender	Uncentered Cosine	2977	5	10	100	1.3293363	1.7683049	0.0061404	0.0077754	0.0083282	0.0000510
GenericUserBasedRecommender	Uncentered Cosine	5843	5	10	200	1.3172718	1.7494480	0.0044669	0.0052316	0.0065061	0.0000692
GenericUserBasedRecommender	Uncentered Cosine	4082	5	20	100	1.3293363	1.7683049	0.0061878	0.0112960	0.0083282	0.0000510
GenericUserBasedRecommender	Uncentered Cosine	4067	5	50	100	1.3293363	1.7683049	0.0063160	0.0216141	0.0083282	0.0000510

Figure 4. MovieLens 1M (GenericUserBasedRecommender)



Algorithm name	Similarity	Running time(in seconds)	Number of folds	Cutoff	Neighborhood size	MAE	RMSE	P@N	nDCG@N	MAP	Coverage	Novelty
GenericItemBasedRecommender	Pearson's Correlation	81915	5	10	50	0.7842698	0.9892134	0.0013300	0.0007857	0.0174983	0.0091330	1.0114245
GenericItemBasedRecommender	Pearson's Correlation	80788	5	10	100	0.7842698	0.9892134	0.0013300	0.0007857	0.0174983	0.0091330	1.0114245
GenericItemBasedRecommender	Pearson's Correlation	89488	5	10	200	0.7784123	0.9795183	0.0006391	0.0004778	0.0111638	0.0054805	1.1533313
GenericItemBasedRecommender	Pearson's Correlation	80848	5	20	100	0.7842698	0.9892134	0.0013438	0.0009731	0.0174983	0.0091330	1.0114245
GenericItemBasedRecommender	Pearson's Correlation	87959	5	50	100	0.7842698	0.9892134	0.0014525	0.0014807	0.0174983	0.0091330	1.0114245
GenericItemBasedRecommender	Euclidean Distance	40141	5	10	50	0.8123320	1.0162986	0.0001104	0.0000513	0.0197672	0.0091372	1.0032079
GenericItemBasedRecommender	Euclidean Distance	45471	5	10	100	0.8123320	1.0162986	0.0001104	0.0000513	0.0197672	0.0091372	1.0032079
GenericItemBasedRecommender	Euclidean Distance	74362	5	10	200	0.8123320	1.0162986	0.0001104	0.0000513	0.0197672	0.0091372	1.0032079
GenericItemBasedRecommender	Euclidean Distance	46773	5	20	100	0.8123320	1.0162986	0.0000993	0.0000561	0.0197672	0.0091372	1.0032079
GenericItemBasedRecommender	Euclidean Distance	46024	5	50	100	0.8123320	1.0162986	0.0001391	0.0001194	0.0197672	0.0091372	1.0032079
GenericItemBasedRecommender	Tanimoto Coefficient	16781	5	10	50	0.7883575	0.9908304	0.0001325	0.0000655	0.0245593	0.0054825	1.1745284
GenericItemBasedRecommender	Tanimoto Coefficient	34681	5	10	100	0.7883575	0.9908304	0.0001325	0.0000655	0.0245593	0.0054825	1.1745284
GenericItemBasedRecommender	Tanimoto Coefficient	41985	5	10	200	0.7883575	0.9908304	0.0001325	0.0000655	0.0245593	0.0054825	1.1745284
GenericItemBasedRecommender	Tanimoto Coefficient	41802	5	20	100	0.7883575	0.9908304	0.0003974	0.0002400	0.0245593	0.0054825	1.1745284
GenericItemBasedRecommender	Tanimoto Coefficient	44973	5	50	100	0.7883575	0.9908304	0.0123742	0.0127039	0.0245593	0.0054825	1.1745284
GenericItemBasedRecommender	Uncentered Cosine	38552	5	10	50	0.8278999	1.0349567	0.0000938	0.0000483	0.0144806	0.0091372	1.0134627
GenericItemBasedRecommender	Uncentered Cosine	46316	5	10	100	0.8278999	1.0349567	0.0000938	0.0000483	0.0144806	0.0091372	1.0134627
GenericItemBasedRecommender	Uncentered Cosine	80076	5	10	200	0.8278999	1.0349567	0.0000938	0.0000483	0.0144806	0.0091372	1.0134627
GenericItemBasedRecommender	Uncentered Cosine	74066	5	20	100	0.8278999	1.0349567	0.0001131	0.0000630	0.0144806	0.0091372	1.0134627
GenericItemBasedRecommender	Uncentered Cosine	70505	5	50	100	0.8278999	1.0349567	0.0001280	0.0001148	0.0144806	0.0091372	1.0134627

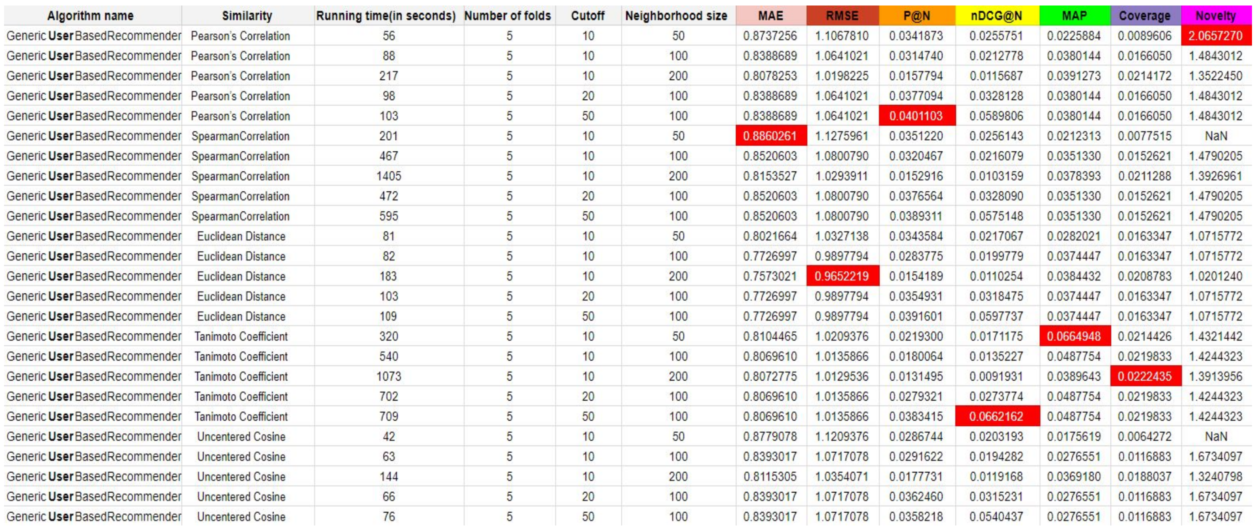
Figure 5. MovieLens 1M (GenericItemBasedRecommender)



Algorithm name	Factrizer	Running time(in seconds)	Cutoff	Factors	Interations	MAE	RMSE	P@N	nDCG@N	MAP	Coverage	Novelty
SVDRecommender	SVDPlusPlusFactorizer	19074	10	10	10	0.7891139	1.0254205	0.1090342	0.0995327	0.0609207	0.0091372	3.3539898
SVDRecommender	SVDPlusPlusFactorizer	24107	10	10	20	0.7905114	1.0268014	0.1019316	0.0945257	0.0613399	0.0091372	3.4468532
SVDRecommender	SVDPlusPlusFactorizer	37186	10	10	50	0.7637541	0.9927231	0.0907892	0.0802399	0.0588689	0.0091372	3.4636289
SVDRecommender	FunkSVDFactorizer	16209	10	10	10	0.7161834	0.9065158	0.0521896	0.0432329	0.0359566	0.0068531	1.0397010
SVDRecommender	FunkSVDFactorizer	24256	10	10	20	0.7119765	0.9021946	0.0360993	0.0292872	0.0281961	0.0054825	1.0811396
SVDRecommender	FunkSVDFactorizer	23989	10	10	50	0.7108090	0.9011610	0.0192748	0.0141144	0.0248531	0.0054825	1.0729999
SVDRecommender	RatingSGDFactorizer	18607	10	10	10	0.7189416	0.9092025	0.0568046	0.0462340	0.0381017	0.0068531	1.0109654
SVDRecommender	RatingSGDFactorizer	20419	10	10	20	0.7087621	0.8974052	0.0209105	0.0504727	0.0368930	0.0068531	1.0080418
SVDRecommender	RatingSGDFactorizer	32623	10	10	50	0.6920877	0.8774397	0.0780022	0.0625832	0.0466067	0.0091372	0.8692031

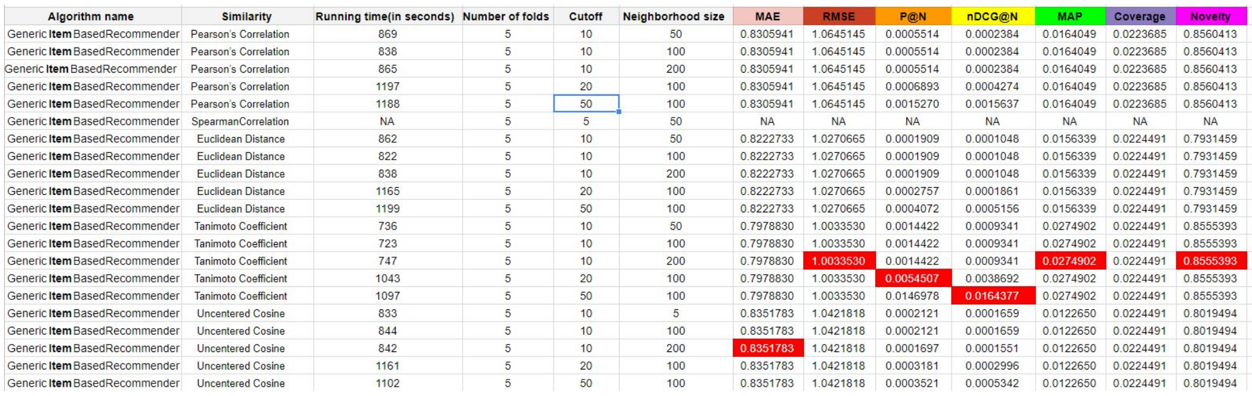
Figure 6. MovieLens 1M (SVDRecommender)

https://docs.google.com/spreadsheets/d/1lT1KJA8wc6Q1uQ0nMsWySM_VWlPyssgjPWGwD2E5SoA/edit#gid=0



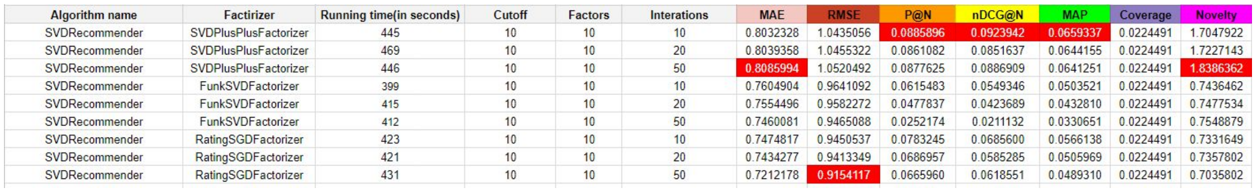
Algorithm name	Similarity	Running time(in seconds)	Number of folds	Cutoff	Neighborhood size	MAE	RMSE	P@N	nDCG@N	MAP	Coverage	Novelty
Generic **User**BasedRecommender	Pearson's Correlation	56	5	10	50	0.8737256	1.1067810	0.0341873	0.0255751	0.0225884	0.0089606	2.0657270
Generic **User**BasedRecommender	Pearson's Correlation	88	5	10	100	0.8388689	1.0641021	0.0314740	0.0212778	0.0380144	0.0166050	1.4843012
Generic **User**BasedRecommender	Pearson's Correlation	217	5	10	200	0.8078253	1.0198225	0.0157794	0.0115687	0.0391273	0.0214172	1.3522450
Generic **User**BasedRecommender	Pearson's Correlation	98	5	20	100	0.8388689	1.0641021	0.0377094	0.0328128	0.0380144	0.0166050	1.4843012
Generic **User**BasedRecommender	Pearson's Correlation	103	5	50	100	0.8388689	1.0641021	0.0401103	0.0589806	0.0380144	0.0166050	1.4843012
Generic **User**BasedRecommender	SpearmanCorrelation	201	5	10	50	0.8860261	1.1275961	0.0351220	0.0256143	0.0212313	0.0077515	NaN
Generic **User**BasedRecommender	SpearmanCorrelation	467	5	10	100	0.8520603	1.0800790	0.0320467	0.0216079	0.0351330	0.0152621	1.4790205
Generic **User**BasedRecommender	SpearmanCorrelation	1405	5	10	200	0.8153527	1.0293911	0.0152916	0.0103159	0.0378393	0.0211288	1.3926961
Generic **User**BasedRecommender	SpearmanCorrelation	472	5	20	100	0.8520603	1.0800790	0.0376564	0.0328090	0.0351330	0.0152621	1.4790205
Generic **User**BasedRecommender	SpearmanCorrelation	595	5	50	100	0.8520603	1.0800790	0.0389311	0.0575148	0.0351330	0.0152621	1.4790205
Generic **User**BasedRecommender	Euclidean Distance	81	5	10	50	0.8021664	1.0327138	0.0343584	0.0217067	0.0282021	0.0163347	1.0715772
Generic **User**BasedRecommender	Euclidean Distance	82	5	10	100	0.7726997	0.9897794	0.0283775	0.0199779	0.0374447	0.0163347	1.0715772
Generic **User**BasedRecommender	Euclidean Distance	183	5	10	200	0.7573021	0.9652219	0.0154189	0.0110254	0.0384432	0.0208783	1.0201240
Generic **User**BasedRecommender	Euclidean Distance	103	5	20	100	0.7726997	0.9897794	0.0354931	0.0318475	0.0374447	0.0163347	1.0715772
Generic **User**BasedRecommender	Euclidean Distance	109	5	50	100	0.7726997	0.9897794	0.0391601	0.0597737	0.0374447	0.0163347	1.0715772
Generic **User**BasedRecommender	Tanimoto Coefficient	320	5	10	50	0.8104465	1.0209376	0.0219300	0.0171175	0.0664948	0.0214426	1.4321442
Generic **User**BasedRecommender	Tanimoto Coefficient	540	5	10	100	0.8069610	1.0135866	0.0180064	0.0135227	0.0487754	0.0219833	1.4244323
Generic **User**BasedRecommender	Tanimoto Coefficient	1073	5	10	200	0.8072775	1.0129536	0.0131495	0.0091931	0.0389643	0.0222435	1.3913956
Generic **User**BasedRecommender	Tanimoto Coefficient	702	5	20	100	0.8069610	1.0135866	0.0279321	0.0273774	0.0487754	0.0219833	1.4244323
Generic **User**BasedRecommender	Tanimoto Coefficient	709	5	50	100	0.8069610	1.0135866	0.0383415	0.0662162	0.0487754	0.0219833	1.4244323
Generic **User**BasedRecommender	Uncentered Cosine	42	5	10	50	0.8779078	1.1209376	0.0286744	0.0203193	0.0175619	0.0064272	NaN
Generic **User**BasedRecommender	Uncentered Cosine	63	5	10	100	0.8393017	1.0717078	0.0291622	0.0194282	0.0276551	0.0116883	1.6734097
Generic **User**BasedRecommender	Uncentered Cosine	144	5	10	200	0.8115305	1.0354071	0.0177731	0.0119168	0.0369180	0.0188037	1.3240798
Generic **User**BasedRecommender	Uncentered Cosine	66	5	20	100	0.8393017	1.0717078	0.0362460	0.0315231	0.0276551	0.0116883	1.6734097
Generic **User**BasedRecommender	Uncentered Cosine	76	5	50	100	0.8393017	1.0717078	0.0358218	0.0540437	0.0276551	0.0116883	1.6734097

Figure 7. MovieLens 100k (GenericUserBasedRecommender)



Algorithm name	Similarity	Running time(in seconds)	Number of folds	Cutoff	Neighborhood size	MAE	RMSE	P@N	nDCG@N	MAP	Coverage	Novelty
Generic **Item**BasedRecommender	Pearson's Correlation	869	5	10	50	0.8305941	1.0645145	0.0005514	0.0002384	0.0164049	0.0223685	0.8560413
Generic **Item**BasedRecommender	Pearson's Correlation	838	5	10	100	0.8305941	1.0645145	0.0005514	0.0002384	0.0164049	0.0223685	0.8560413
Generic **Item**BasedRecommender	Pearson's Correlation	865	5	10	200	0.8305941	1.0645145	0.0005514	0.0002384	0.0164049	0.0223685	0.8560413
Generic **Item**BasedRecommender	Pearson's Correlation	1197	5	20	100	0.8305941	1.0645145	0.0006893	0.0004274	0.0164049	0.0223685	0.8560413
Generic **Item**BasedRecommender	Pearson's Correlation	1188	5	50	100	0.8305941	1.0645145	0.0015270	0.0015637	0.0164049	0.0223685	0.8560413
Generic **Item**BasedRecommender	SpearmanCorrelation	NA	5	5	50	NA	NA	NA	NA	NA	NA	NA
Generic **Item**BasedRecommender	Euclidean Distance	862	5	10	50	0.8222733	1.0270665	0.0001909	0.0001048	0.0156339	0.0224491	0.7931459
Generic **Item**BasedRecommender	Euclidean Distance	822	5	10	100	0.8222733	1.0270665	0.0001909	0.0001048	0.0156339	0.0224491	0.7931459
Generic **Item**BasedRecommender	Euclidean Distance	838	5	10	200	0.8222733	1.0270665	0.0001909	0.0001048	0.0156339	0.0224491	0.7931459
Generic **Item**BasedRecommender	Euclidean Distance	1165	5	20	100	0.8222733	1.0270665	0.0002757	0.0001861	0.0156339	0.0224491	0.7931459
Generic **Item**BasedRecommender	Euclidean Distance	1199	5	50	100	0.8222733	1.0270665	0.0004072	0.0005156	0.0156339	0.0224491	0.7931459
Generic **Item**BasedRecommender	Tanimoto Coefficient	736	5	10	50	0.7978830	1.0033530	0.0014422	0.0009341	0.0274902	0.0224491	0.8555393
Generic **Item**BasedRecommender	Tanimoto Coefficient	723	5	10	100	0.7978830	1.0033530	0.0014422	0.0009341	0.0274902	0.0224491	0.8555393
Generic **Item**BasedRecommender	Tanimoto Coefficient	747	5	10	200	0.7978830	1.0033530	0.0014422	0.0009341	0.0274902	0.0224491	0.8555393
Generic **Item**BasedRecommender	Tanimoto Coefficient	1043	5	20	100	0.7978830	1.0033530	0.0054507	0.0038692	0.0274902	0.0224491	0.8555393
Generic **Item**BasedRecommender	Tanimoto Coefficient	1097	5	50	100	0.7978830	1.0033530	0.0146978	0.0164377	0.0274902	0.0224491	0.8555393
Generic **Item**BasedRecommender	Uncentered Cosine	833	5	10	5	0.8351783	1.0421818	0.0002121	0.0001659	0.0122650	0.0224491	0.8019494
Generic **Item**BasedRecommender	Uncentered Cosine	844	5	10	100	0.8351783	1.0421818	0.0002121	0.0001659	0.0122650	0.0224491	0.8019494
Generic **Item**BasedRecommender	Uncentered Cosine	842	5	10	200	0.8351783	1.0421818	0.0001697	0.0001551	0.0122650	0.0224491	0.8019494
Generic **Item**BasedRecommender	Uncentered Cosine	1161	5	20	100	0.8351783	1.0421818	0.0003181	0.0002996	0.0122650	0.0224491	0.8019494
Generic **Item**BasedRecommender	Uncentered Cosine	1102	5	50	100	0.8351783	1.0421818	0.0003521	0.0005342	0.0122650	0.0224491	0.8019494

Figure 8. MovieLens 100k (GenericItemBasedRecommender)



Algorithm name	Factirizer	Running time(in seconds)	Cutoff	Factors	Interations	MAE	RMSE	P@N	nDCG@N	MAP	Coverage	Novelty
SVDRecommender	SVDPlusPlusFactorizer	445	10	10	10	0.8032328	1.0435056	0.0885896	0.0923942	0.0659337	0.0224491	1.7047922
SVDRecommender	SVDPlusPlusFactorizer	469	10	10	20	0.8039358	1.0455322	0.0861082	0.0851637	0.0644155	0.0224491	1.7227143
SVDRecommender	SVDPlusPlusFactorizer	446	10	10	50	0.8085994	1.0520492	0.0877625	0.0886909	0.0641251	0.0224491	1.8386362
SVDRecommender	FunkSVDFactorizer	399	10	10	10	0.7604904	0.9641092	0.0615483	0.0549346	0.0503521	0.0224491	0.7436462
SVDRecommender	FunkSVDFactorizer	415	10	10	20	0.7554496	0.9582272	0.0477837	0.0423689	0.0432810	0.0224491	0.7477534
SVDRecommender	FunkSVDFactorizer	412	10	10	50	0.7460081	0.9465088	0.0252174	0.0211132	0.0330651	0.0224491	0.7548879
SVDRecommender	RatingSGDFactorizer	423	10	10	10	0.7474817	0.9450537	0.0783245	0.0685600	0.0566138	0.0224491	0.7331649
SVDRecommender	RatingSGDFactorizer	421	10	10	20	0.7434277	0.9413349	0.0686957	0.0585285	0.0505969	0.0224491	0.7357802
SVDRecommender	RatingSGDFactorizer	431	10	10	50	0.7212178	0.9154117	0.0665960	0.0618551	0.0489310	0.0224491	0.7035802

Figure 9. MovieLens 100k (SVDRecommender)

APPENDIX B All graphics

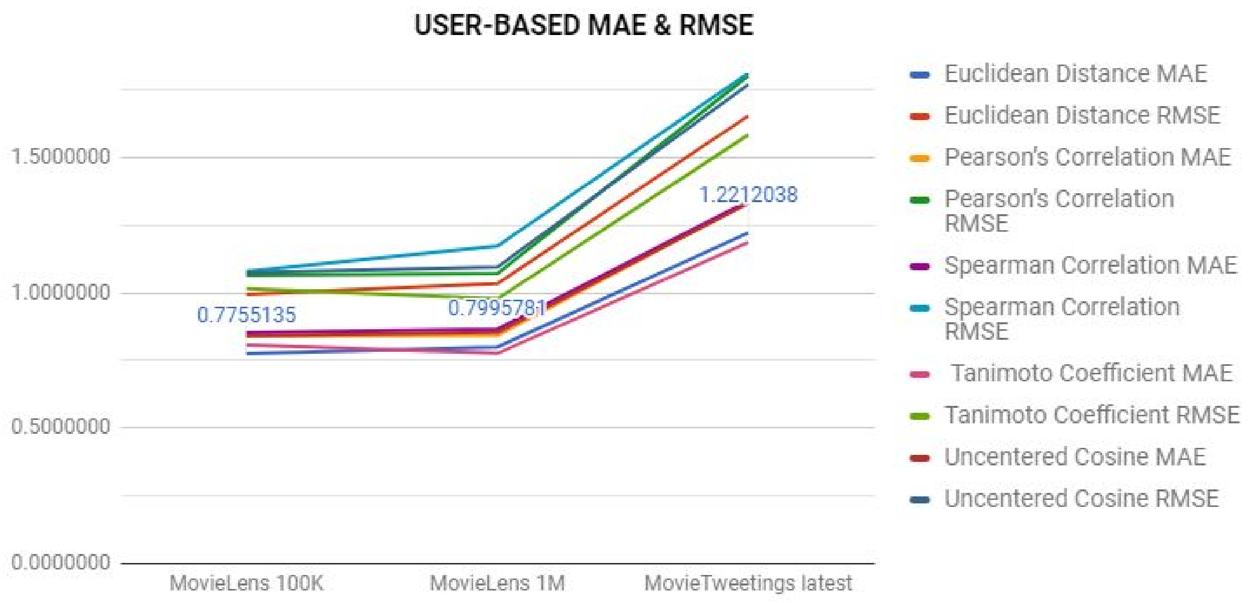


Figure 10. Mean of MAE and RMSE (User-based)

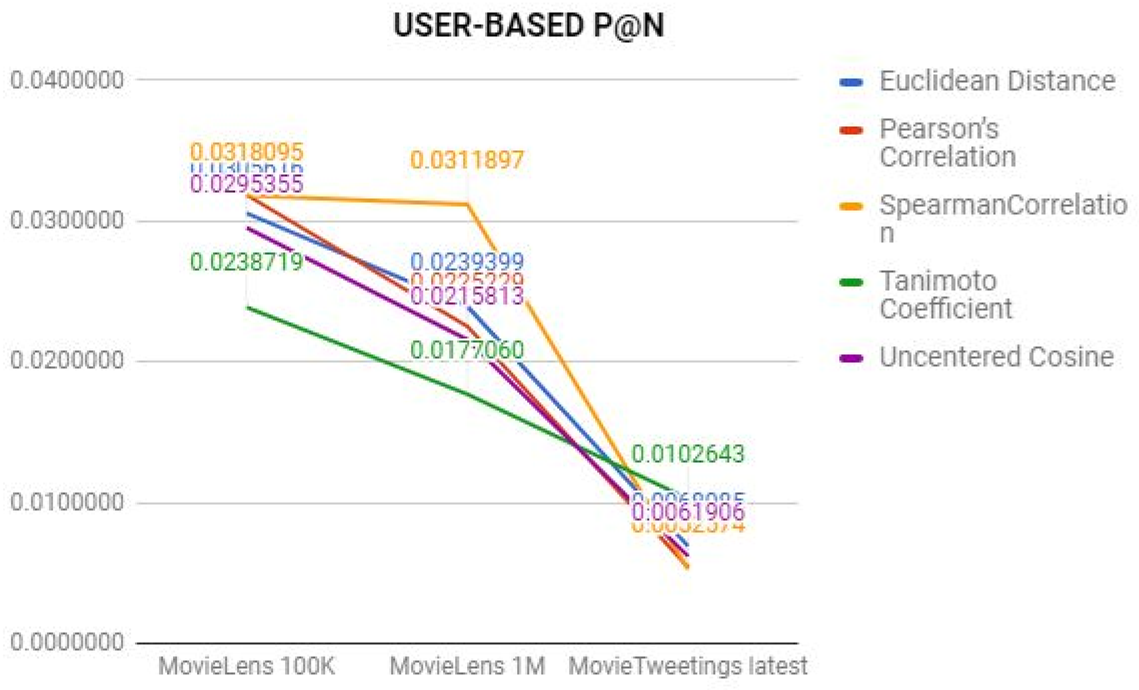


Figure 11. Mean of Precision@N (User-based)

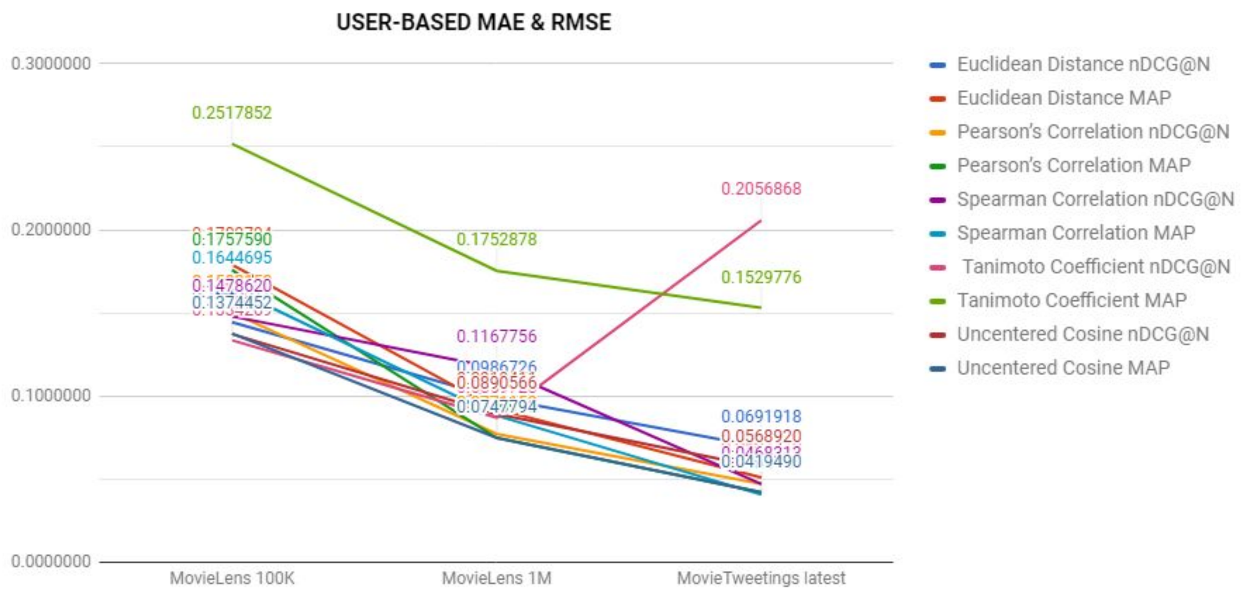


Figure 12. Mean of nDCG and MAP (User-based)

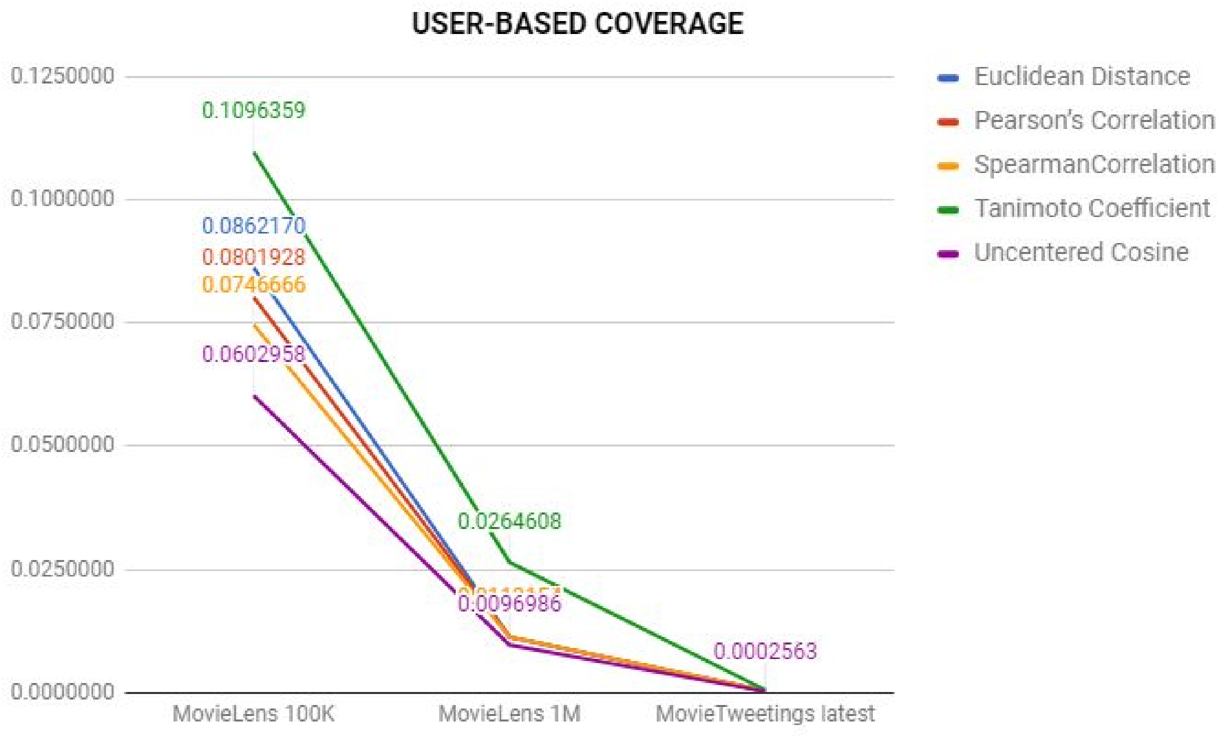


Figure 13. Mean of Coverage (User-based)
Figure 14. Mean of MAE and RMSE(Item-based recommenders)
Figure 15. Mean of P@N (Item-based recommenders)
Figure 16. Mean of nDCG and MAP (Item-based recommenders)
Figure 17. Mean of Coverage (Item-based recommenders)

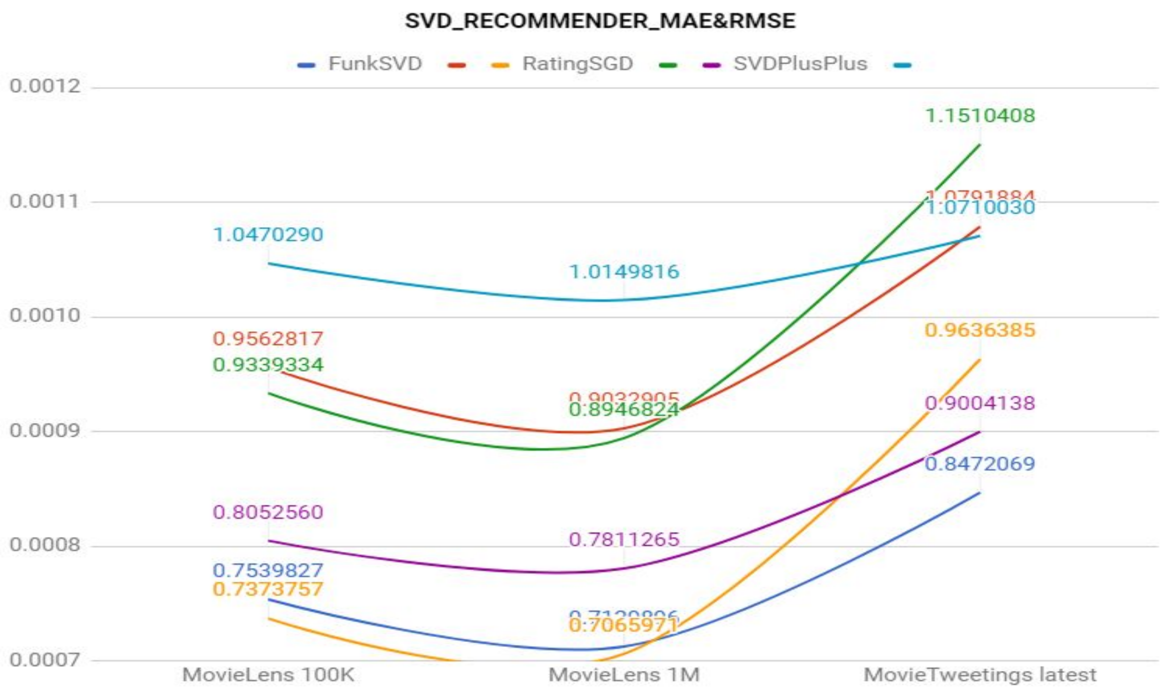


Figure 18. Mean of MAE and RMSE (SVD recommenders)

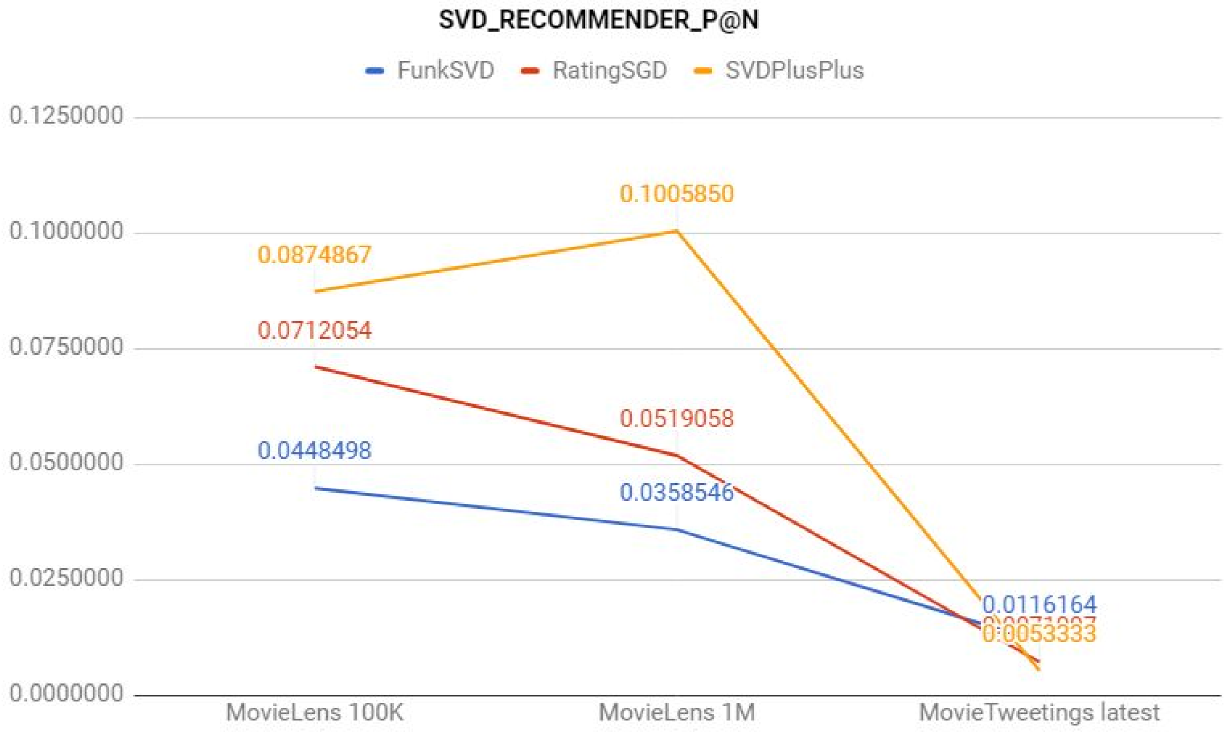


Figure 19. Mean of P@N (SVD recommenders)

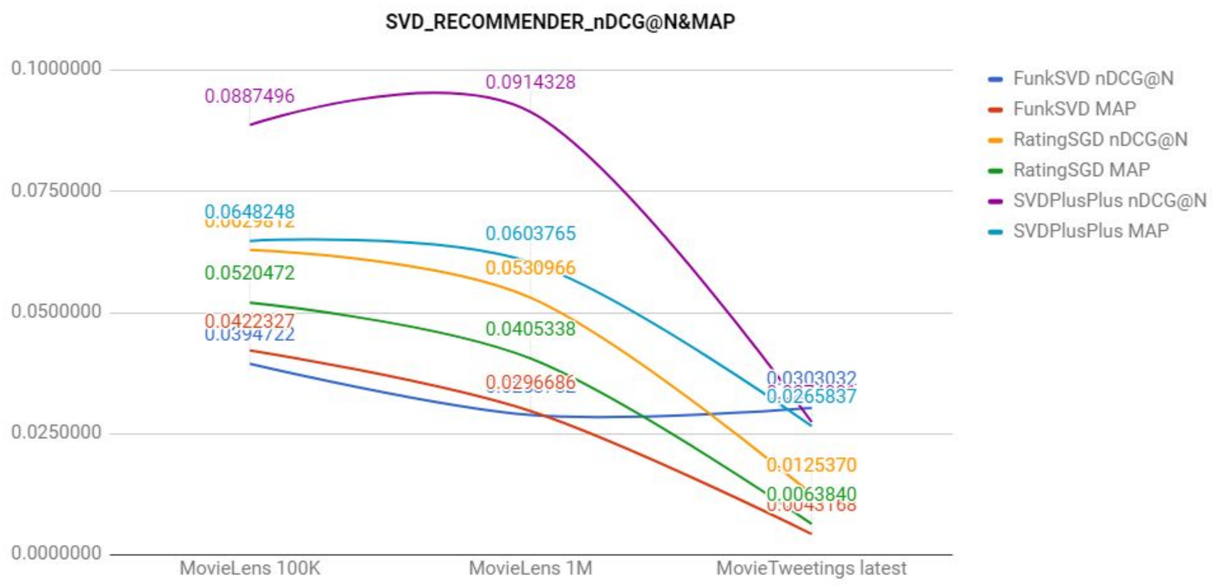


Figure 20. Mean of MAP and nDCG (SVD recommenders)

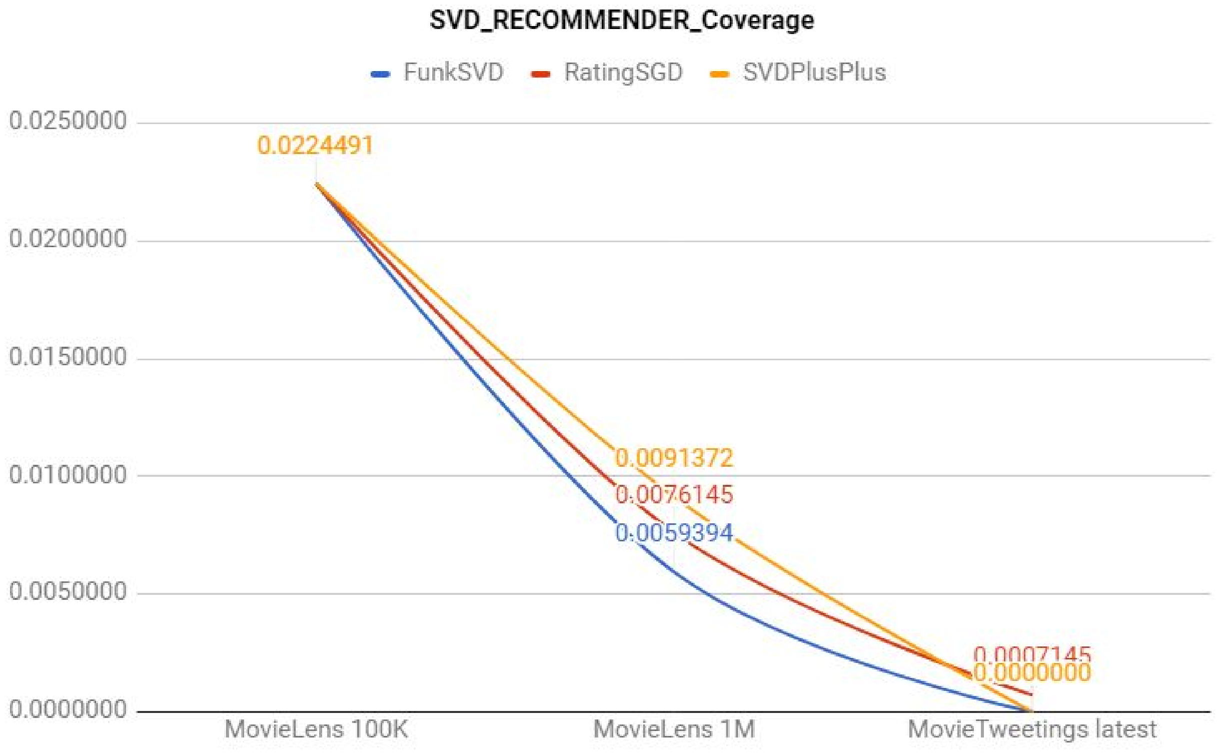


Figure 21. Mean of Coverage (SVD recommenders)