A comparative of Imputation Techniques for Missing Data in Collaborative Filtering Using Apache Mahout

Morad Ali Hassan – faculty of Science – Bani Waleed University

Mohamed Abdo ulwahad Alsharaa – Faculty of Education – Bani Waleed University

Abstract :

Recommender systems are a powerful tool that can be used to improve the user experience in a variety of applications. However, the issue of missing data in the user-item rating matrix is a common problem that affects the performance of these systems. To solve this problem, imputation techniques are used to estimate the missing values in the matrix. Apache Mahout is one of the popular open-source libraries that provide various algorithms for building recommender systems. It also provides an implementation of several imputation techniques to handle missing data in the user-item rating matrix. This paper aims to improve the accuracy and the performance of user-based collaborative filtering (UB-CF) by applying the imputation technique with Apache Mahout. The experiments are carried out on real world data sets Movielens. The results proved that our proposed method is effective in handling and identifying missing and noisy data in the user-item rating matrix. We demonstrate that our approach led to considerable enhancement compared with other previous approaches.

KEYWORDS: Recommender systems; collaborative filtering, Apache Mahout, Imputation techniques, Big Data, missing and noisy data; predicting.

1. INTRODUCTION

Recommender systems are a type of artificial intelligence technology used to predict the preferences and interests of users based on their past behavior. These systems make personalized recommendations based on user data such as purchase history, search queries, and ratings. They are used in a variety of applications such as online shopping, streaming services, and social networks. Recommender systems use algorithms such as collaborative filtering, content-based filtering, and matrix factorization to generate accurate recommendations tailored to each user. They can improve user satisfaction by presenting the most relevant products, videos, music, or other content to all users.

Table 1. Sample of user-item rating matrix

	King of Rings	Beautiful Mind	Star Wars	Titanic
Smith	5	2	5	4
Adam	2	5		3
Natasha	2	2	4	2
John	5	1	5	?

2. RELATED WORK

Imputation is a technique for filling in missing values in a dataset. It can be used to improve the accuracy of collaborative filtering (CF) recommender systems, which are a type of recommendation system that recommends items to users based on their past behavior [1,2]. CF recommender systems can be inaccurate when the user preference data used in the recommendation process is sparse. Imputation can alleviate this problem by substituting a virtual part of the missing user preferences [3,4].

According to recent research, imputation techniques such as matrix factorization and deep learning-based methods have shown promising results in improving collaborative filtering performance. For instance, a study by Zhang et al. (2019) found that using a deep learning-based imputation method improved the accuracy of collaborative filtering by up to 5% [5]. Similarly, another study by Wang et al. (2021) showed that incorporating matrix factorization-based imputation techniques led to significant improvements in recommendation quality compared to traditional collaborative filtering methods [6]. Overall, these findings suggest that imputation techniques can be an effective approach for enhancing the performance of collaborative filtering systems.

Ching et al. (2018) discussed the use of collaborative filtering techniques to create a movie recommendation system. Collaborative filtering is a technique that uses the preferences and behavior of similar users to make recommendations for a particular user [7]. The authors of the paper used Apache Mahout, an open-source machine learning library, to implement their recommendation system. They also used imputation techniques to handle missing data in their dataset. Imputation is the process of estimating missing values in a dataset based on other available data. The authors evaluated their recommendation system using two different datasets and found that it performed well in terms of accuracy and efficiency. They also compared their results with other state-of-the-art recommendation systems and found that their system performed comparably or better. Overall, this paper provides a detailed overview of how collaborative filtering can be used to create an effective movie recommendation system, and highlights the importance of imputation techniques and tools like Apache Mahout in building such systems. Also, Alessio et al. (2018) in their focused on collaborative filtering, imputation, and Apache Mahout [8]. The authors propose a distributed neural network approach for imputing missing data in large datasets. They use Apache Mahout to implement the proposed approach and evaluate its performance on several real-world datasets. The results show that the proposed approach outperforms existing methods for missing data imputation. Overall, the paper presents a promising solution for handling missing data in large datasets using distributed neural networks and Apache Mahout.

3. PROPOSED FRAMEWORK

The proposed framework can be applied to two different scenarios.

3.1 Scenario 1: APACHE MAHOUT

In this paper, Apache Mahout [9] is used as a software framework that enables developers to create scalable and effective recommender systems. Mahout is an open-source machine learning library that was originally developed by Apache Lucene in 2008. Mahout is a collection of open-source software tools for scalable machine learning. When the amount of data is too large, Apache Mahout is a popular choice for collaborative filtering (CF) libraries. Mahout is written in the Java programming language and does not provide a user interface or installer. After coding, it is up to the developer to create interfaces for their specific application. The Mahout library contains a variety of recommender systems. User-Based CF to adapt out proposed work is chosen for this study. Fig1 shows the block diagram of this scenario.

3.1.1 DATASET

In this paper, the authors applied various algorithms to the MovieLens-100K dataset, which contains 100,000 ratings from 943 users on 1,682 movies. The dataset only included users who had rated at least 20 movies. The dataset is composed of two files:

- u.data: This file contains the user ID, item ID, rating, and timestamp of each rating.
- u.item: This file contains information about each movie, such as the movie ID, title, release date, and genres.

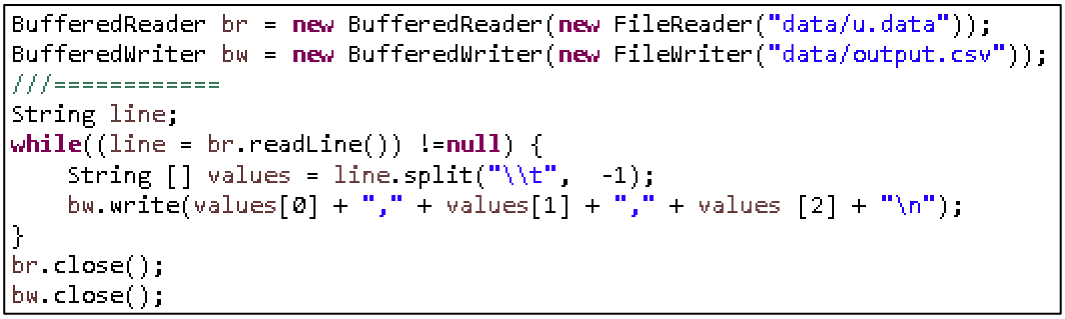
We merged the two data sets in our experiments because the movie IDs were the same in both sets. We selected u.data for this paper.

3.1.2 CONVERT THE FORMAT OF DATASET FILE

To build the inputs, the first step is to convert the datasets to a CSV file format. This file contains the user ID, item ID, and the given preferences (ratings). In Mahout, IDs are always integers, and the preferences have the property that a larger number indicates a stronger positive preference. In the MovieLens datasets, the preferences are integers between 1 and 5. After converting the data file to a CSV file, the information included by csv file will be formatted as shown in Table 2:

User ID	**Movie ID**	**Rates**
1	102	3
2	35	2
2	75	5
91	102	3
101	54	3
101	102	4

The following code is how to convert the .data into csv file in Java:



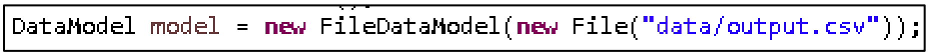
```
BufferedReader br = new BufferedReader(new FileReader("data/u.data"));
BufferedWriter bw = new BufferedWriter(new FileWriter("data/output.csv"));
///============
String line;
while((line = br.readLine()) !=null) {
    String [] values = line.split("\\t",  -1);
    bw.write(values[0] + "," + values[1] + "," + values [2] + "\n");
}
br.close();
bw.close();
```

3.1.3 Creating a Recommender

The following codes are presented by Mahout to recommend some items for the active user as following:

- **Load Data:**

This code is to upload the converted csv file from the previous step:



```
DataModel model = new FileDataModel(new File("data/output.csv"));
```

- **Create Similarity and Neighbor Selection:**

To predict and provide suggestions using the collaborative filtering method, we first need to identify the most similar users to the active user. The active user is the user for whom we want to predict the score of a target item. We can use the Pearson Correlation Coefficient equation to measure the similarity between users. The Pearson Correlation Coefficient is a measure of the linear relationship between two variables. It ranges from -1 to 1, with 1 indicating a perfect positive correlation, -1 indicating a perfect negative correlation, and 0 indicating no correlation. [11]. Where the neighbor size here is chosen as 50 neighbors as an example.

```
RecommenderBuilder recommenderBuilder = new RecommenderBuilder() {
    public Recommender buildRecommender(DataModel model) throws TasteException {

        UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
        UserNeighborhood neighborhood = new NearestNUserNeighborhood (50, similarity, model);
        return new GenericUserBasedRecommender(model, neighborhood, similarity);
    }
};
```

- **Create Recommender Engine:**

In this step we are going to recommend 5 items to the active user that not seen or viewed yet. User ID 1 considered as the active user.

```
Recommender recommender = recommenderBuilder.buildRecommender(model);
List<RecommendedItem> recomendations = recommender.recommend(1, 5);
for (RecommendedItem recommendedItem : recomendations) {
    System.out.println(recommendedItem);
}
```

- **Evaluation Metrics:**

Once a model has been built, the most important question that arises is how well does it perform? Therefore, evaluating any model is the most important task in a data science project, as it determines how accurate the model's predictions are [12].

Evaluation is a part of Mahout's framework. It is to measure the error percentage from obtained results. Four evaluation metrics are used, Root Mean Square Erroe (RMSE), Precision, Recall, and F1 Score.

```
RecommenderEvaluator evaluator = new RMSRecommenderEvaluator();
double score = evaluator.evaluate(recommenderBuilder, null, model, 0.7, 1.0);
System.out.println("RMSE: " + score);

RecommenderIRStatsEvaluator statsEvaluator = new GenericRecommenderIRStatsEvaluator();
IRStatistics stats = statsEvaluator.evaluate(recommenderBuilder, null, model, null, 10, 4, 0.7);

System.out.println("Precision: " + stats.getPrecision());
System.out.println("Recall    : " + stats.getRecall());
System.out.println("F1 Score : " + stats.getF1Measure());
```

To train the dataset, the training perecentage can be chosen as shown in this code where 70% presented here as an example and can be modified according to the requirements of experiment.

3.2 Scenario 2: Predicting Missing values

Part of the proposed approach is presented in early published paper [13]. The block diagram presented bellow shows two scenarios of proposed work. Scenario 1 is described in 3.1. After converting the csv file, both scenarios used same converted file which contains the original data. Scenario 2 creates a new dataset that contains original data with fixed missing values. Next, the new dataset is applied to Apache Mahout Library (combine scenario 1) and a new recommendatios are obtained.

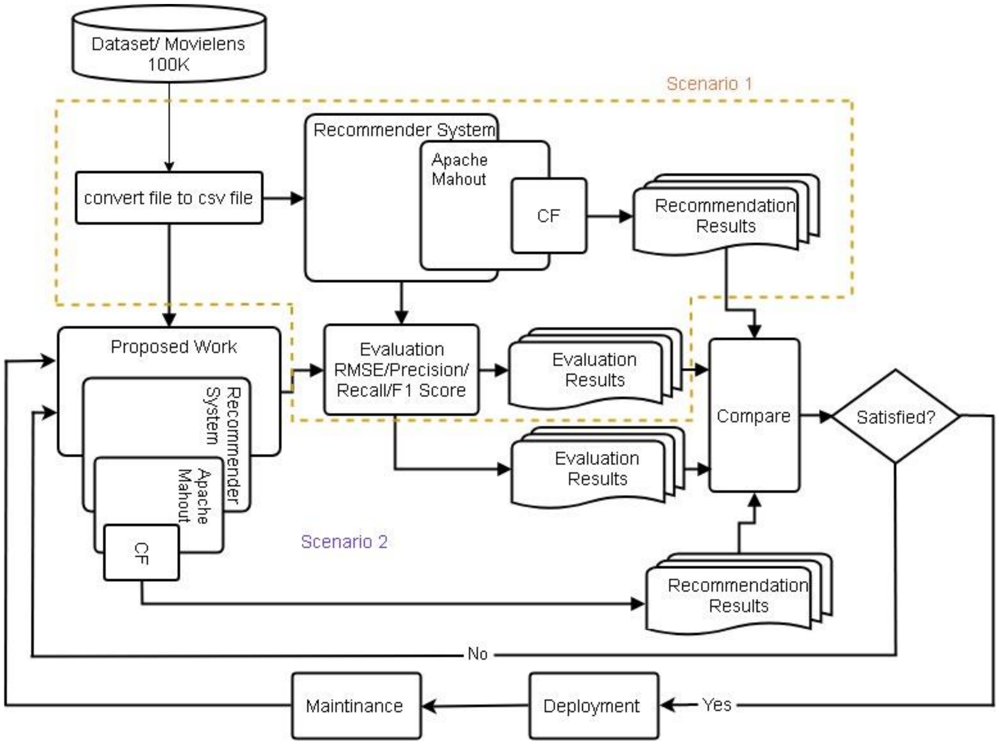


Fig1. Framework shows the relationship between our Method and Apache Mahout

The results of both scenarios and previous studies are compared. For better results, some experiments adopted (neighbors' size, percentage of training set).

The steps of this scenario are presented as follow:

3.2.1 Filtering User-Item Rating Matrix

In this step, the user-item matrix is filtered to identify users who have strong relationships with the active user. This is done by finding users who have rated a similar number of items as the active user and who have rated those items similarly. The users who are identified in this step are considered to be the active user's neighbors. Hence, this filtering process prevents any user who does not have any shared preferences with the active user to be part of the new user-item rating matrix.

$$\textit{User-Item matrix} = \begin{cases} filter \quad if \;\; r_{a,i} \cap r_{b,i} = \theta \\ \qquad\qquad 0 \le \delta < K \\ keep \quad if \;\; r_{a,i} \cap r_{b,i} = \theta \\ \qquad\qquad \delta \ge K \end{cases}$$

Where: $r_{a,i}$ is the rating given to item i by Ua; $r_{b,i}$ is the rating given to item $\boldsymbol{i}$ by Ub; *K* is defined by admin to describe if the number of items in common with active user less than *K*, then the user-item matrix must be filtered and θ is the threshold value.

3.2.2 Finding Distance between the Active User and Neighbors:

This step illustrates how much neighbors' opinion is different from that of active user on the items they both rated.

$$\boldsymbol{Diff}(\boldsymbol{Ua}, \boldsymbol{Ub}) = \sum_{i=1}^{m} \left| r_{a,i} - r_{b,i} \right| \qquad (1)$$

Where: Ua is the active user; Ub describes all the users in dataset; and m is the total number of items.

3.2.3 Identifying the Average Rating Between Ua and Ub:

The weights are determined by the similarity between the active user and each neighbor. The number of items that both the active user and his neighbors rated is used to normalize the weights. This ensures that the ratings of neighbors who have rated more items in common with the active user have a greater influence on the prediction.

$$\boldsymbol{AVG\ (Ua, Ub)} = \frac{Diff(Ua,Ub)}{n}$$

(2)

3.2.4 Imputing the Missing Value of Item $r_{u,i}$ for *Ub*:

Earlier approaches relied on imputation [14-18] to fill in missing ratings and make the rating matrix dense. However, imputation can be very expensive as it significantly increases the amount of data. This paper proposed a method that can overcome this issue by adding a new parameter to choose the best neighbors in the user-item rating matrix [17-19].

The imputation process in this step predicts a rating to items which the neighbors are not rated. On the other hand, the active user rated them.

$$\boldsymbol{impute\ (Ub, i \rightarrow 0)} = |r_{a,i} - AVG(\boldsymbol{Ua, Ub})|$$

(3)

Where: $\boldsymbol{impute\ (Uib \rightarrow 0)}$ is the missing value of *Ub* on item *i*.

3.2.5 Neighbor Selection and Similarity between Ua and Ub:

In order to predict and provide suggestions using the collaborative filtering method, we must first identify the most similar users to the active user. The active user is the user for whom we want to predict the score of a target item. We can create a set of neighbors for the active user by using the standard Pearson Correlation Coefficient [11]. This coefficient measures the linear relationship between two variables. In our case, the two variables are the ratings that the active user and each other user have given to the target item [20].

The Pearson Correlation Coefficient is calculated as follows:

$$\boldsymbol{Sim_{Ua,Ub}} = \frac{\sum_{i=1}^{m}(r_{a,i}-\bar{r}_a) \times (r_{b,i}-\bar{r}_b)}{\sqrt{\sum_{i=1}^{m}(r_{a,i}-\bar{r}_a)^2 \times \sum_{i=1}^{m}(r_{b,i}-\bar{r}_b)^2}}$$

(4)

Where $\bar{r}_a$ is the mean rating given by user *a*, $\bar{r}_b$ is the mean rating given by user *b*.

The above formula can be used to calculate the similarity between two users. This similarity can be used to determine which users are most similar to each other. Users who are similar to each other are likely to have similar interests and preferences. This means that they are likely to rate items in a similar way. As a result, users who are similar to each other can be considered to be reliable neighbors.

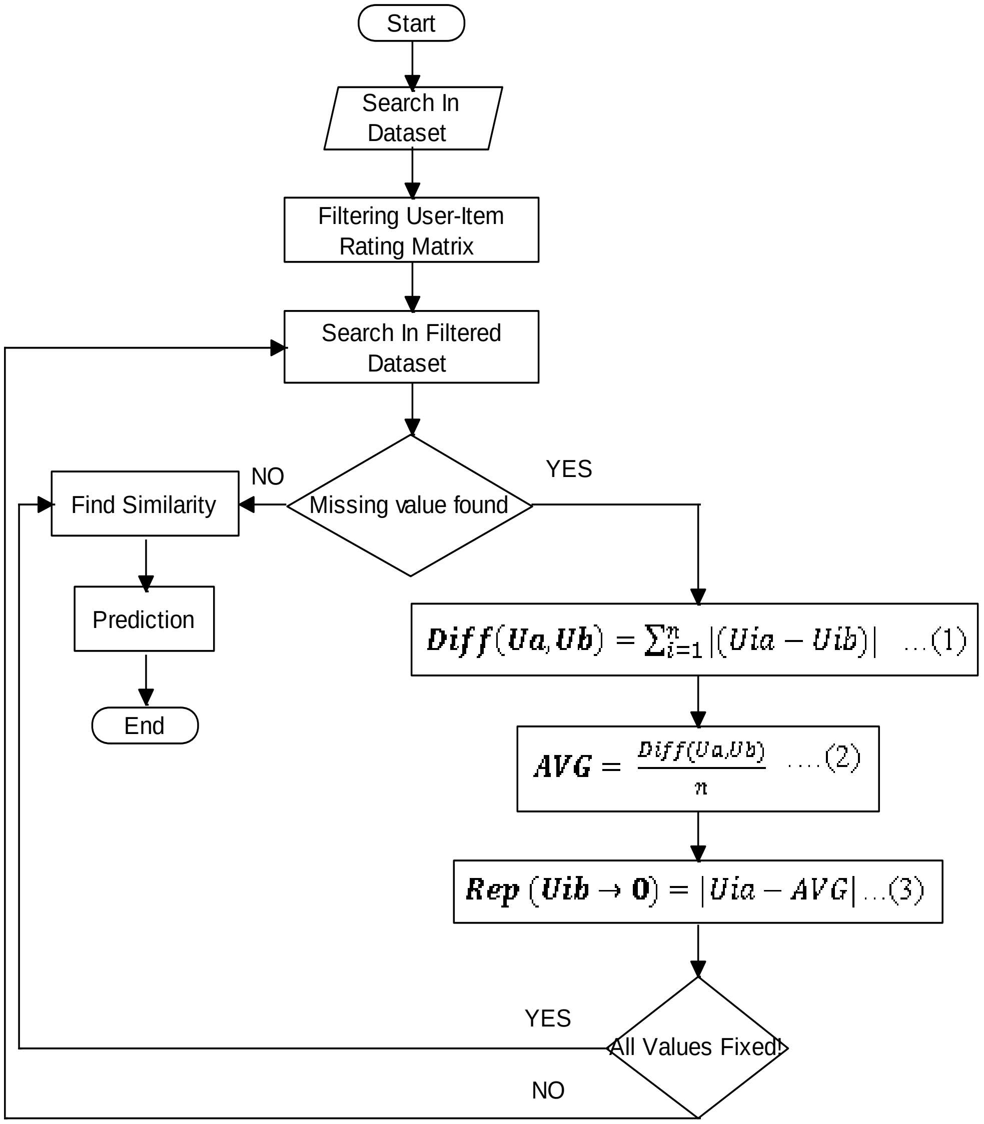
3.2.6 Prediction

In this step, we use the results obtained from the previous step to predict the rating that the active user will give to the target item. We do this by taking the weighted average of the ratings of the neighbors on the same item. The weight for each neighbor is calculated using the similarity between the active user and the neighbor [20].

$$\boldsymbol{p_{Ua,i}} = \bar{r}_a + \frac{\sum_{b=1}^{m}(r_{b,i}-\bar{r}_b) \times Sim_{a,b}}{\sum_{u=1}^{n}|Sim_{a,b}|}$$

(5)

Where, $p_{Ua,i}$ is the prediction for the active user *a* for item *i*, $Sim_{a,b}$ is the similarity between users *a* and *b* ; *n* is the number of users in the neighborhood.


Start
Search In Dataset
Filtering User-Item Rating Matrix
Search In Filtered Dataset
Missing value found
NO
YES
Find Similarity
Prediction
End
$Diff(Ua,Ub) = \sum_{i=1}^{n} |(Uia - Uib)| \quad ...(1)$
$AVG = \frac{Diff(Ua,Ub)}{n} \quad ....(2)$
$Rep\ (Uib \to 0) = |Uia - AVG| ...(3)$
All Values Fixed?
YES
NO

4. Experimental Analysis

4.1. Experimental Steps

The dataset was divided into a training and test portion. In the experiments, 0.7 and 0.9 training–test ratios were used to calculate and compare the prediction accuracy. For each similarity measure and collaborative filtering technique, evaluation was coded to find the mean absolute error, precision, recall, and F1 score.

4.2. Experimental Platform

All of our experiments were implemented in the Java programming language. The experiments were run on a Windows-based PC with an Intel Core i5 processor running at 1.8 GHz and 8 GB of RAM.

4.3 Experimental Results

The experimental results of user-based collaborative filtering (CF) for creating predictions are shown and the parameters to be determined as follow::

- Neighborhood size: The number of users to consider as neighbors of the active user.
- Training/test ratio: The fraction of the data to use for training and the fraction to use for testing.
- Effects of different number of items for users in common with active user: The impact of the number of items that users have in common with the active user on the accuracy of the predictions.

All of the classes that contain evaluation metrics were run separately. The results were then recorded in order to compare them. Using this information, tables and histograms were created. The table 3 below shows the parameters that were used in the experiments.

Parameter	Description	Scale
Neighbor size	The number of users who have high similarity with active user.	10, 20, 30, 50, 70, 80, 100
Training test	The percentage of dataset that be used in training test.	70% and 90%
Items in common	Taking into account only the	

with active user	users who have rated the same items the active user rated too.	20, 50, 100, 150

In the case of neighbors' size, the size of the neighbors affect the prediction quality. By changing the number of neighbors, sensitivity of neighborhood is determined. Many studies noticed that increasing the number of neighbors' size leads to increasing in the quality of prediction [10, 18, 19]. Most of these studies control the number of neighbors to get high accuracy for their prediction. In this study, we added another parameter before we measure the similarity and before selecting the neighbors' size. We first filter users according the users who have certain number of rated items in common with active user.

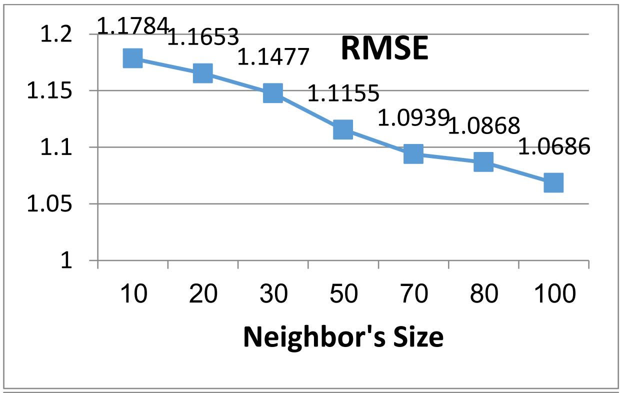
4.4 Experiment 1:

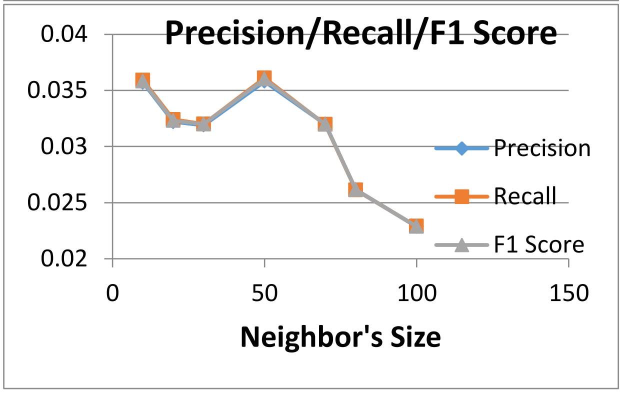
The results presented here describe the effect of neighbor's size on traditional User-Based collaborative filtering using Apache Mahout as shown in Fig1. The results illustrate the accuracy of prediction using RMSE, Precision, Recall, and F1 metrics with training/test are 0.7 and 0.9.

Neighbor Size	Training / Test	RMSE	Precision	Recall	F1
10	0.7	1.1784	0.0357	0.0359	0.0358
20	0.7	1.1653	0.0322	0.0324	0.0323

30	**0.7**	**1.1477**	**0.0319**	**0.0320**	**0.0320**
50	**0.7**	**1.1155**	**0.0358**	**0.0361**	**0.0360**
70	**0.7**	**1.0939**	**0.03198**	**0.03195**	**0.03197**
80	**0.7**	**1.0868**	**0.02615**	**0.02613**	**0.02614**
100	**0.7**	**1.0686**	**0.02286**	**0.02289**	**0.02288**
10	0.9	1.1625	0.0352	0.0354	0.0354
20	0.9	1.1679	0.0326	0.0328	0.0327
30	0.9	1.1507	0.0313	0.0314	0.0313
50	0.9	1.1119	0.0343	0.0346	0.0345
70	0.9	1.0844	0.0308	0.0308	0.0308
80	0.9	1.0750	0.0264	0.0264	0.0264
100	0.9	1.0601	0.0226	0.0227	0.0227

As can be seen in Fig3, the obtained results prove the fact presented by many studies, as we increase the number of neighbors, the accuracy of prediction is also increases. As known, the lower the RMSE, the higher prediction accuracy. In the case of Precision, Recall, and F1 metrics, the relationship between the size of neighbors and the accuracy is different from RMSE. Table shows that the lowest the number of neighbors the highest the prediction accuracy





The sample presented bellow describes 5 recommended items to user 1 and neighbors with size equal to 10 neighbors:

```
RecommendedItem[item:879, value:5.0]
RecommendedItem[item:312, value:5.0]
RecommendedItem[item:990, value:5.0]
RecommendedItem[item:902, value:5.0]
RecommendedItem[item:984, value:5.0]
```

4.5 Experiment 2:

At this stage, different procedures are adopted. A new parameter (ɳ) added to our proposed method. As described in scenario 2, the first step is filtering users according to the items they shared or rated in common with active user. The developer decides the number of shared items. This number considered as the new parameter we added in this study. Experiment 2 has for cases, the neighbor's size scaled as 10, 20, 30, 50, 70, 80, 100 for each case, and the training/Test is 0.9 for all case where achieved better accuracy than 0.7. Table 4 presents the obtained results for all cases after running each case separately.

The observation we should take into account after running our Java code is the dataset. The table below shows the statistics of dataset (Original Data) for experiment 1 compared with the dataset of experiment 2. Each case we count the number of users, items, and ratings after applying the proposed algorithm. Consequently, the number of users, items, and ratings decreased as the ɳ increased. Therefore, we noted that the executing time of Java program becomes faster. Fig4 shows the fastest and lowest cases according to the increasing in ɳ.

Table 4. Statistics of Dataset

		ɳ	**Users**	**Items**	**Ratings**
Experiment 1		**0**	**943**	**1682**	**100,000**
Experiment 2	**Case I**	**20**	**582**	**1669**	**190332**
	Case II	**50**	**320**	**1642**	**116956**
	Case III	**100**	**116**	**1572**	**49750**
	Case IV	**150**	**21**	**1471**	**11212**

The fastest and slowest executing time of Java code for all cases is presented in Fig4. As shown here, as we increase the value of η the executing time become faster. Experiment 1 where Apache Mahout is applied and $\eta = 0$ conducted the slowest executing time because the dataset not filtered yet. So, the compiler takes a bit more time to read all records. In addition, the executing time becomes faster as the number of records decreases.

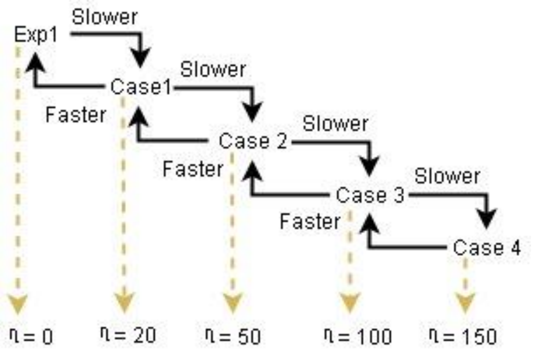


Fig4. The executed time

Table 5 presents the main results of our experiments. We found that the ratio 0.9 of training set gives better results than those of 0.7.

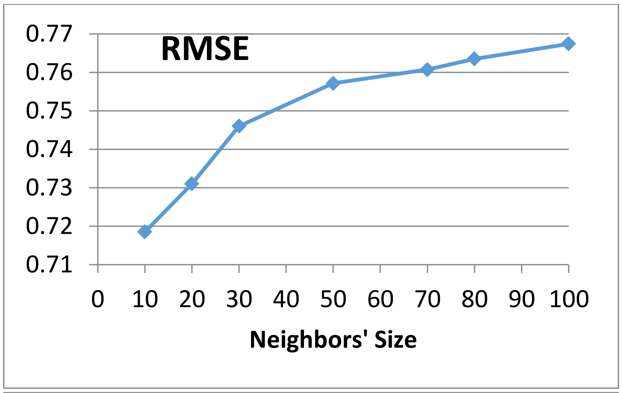
Table 5. RMSE, Precision, and Recall of Scenarios

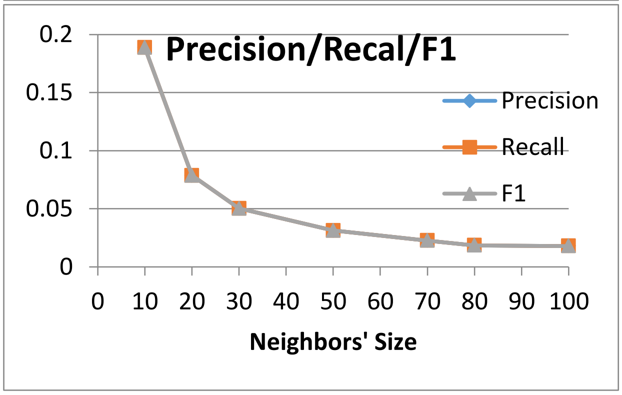
Cases	**Neighbor Size**	**Training / Test**	η	**RMSE**	**Precision**	**Recall**	**F1**
Case I	10	0.9	20	0.7185	0.1886	0.18861	0.18861
	20	0.9	20	0.7310	0.0787	0.0787	0.0787

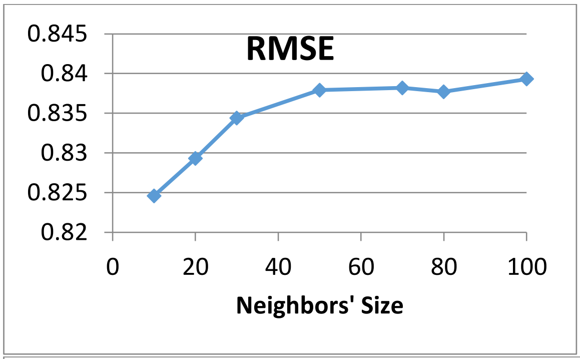
	30	0.9	20	0.7460	0.0504	0.0504	0.0504
	50	0.9	20	0.7571	0.0313	0.0313	0.0313
	70	0.9	20	0.7607	0.0227	0.0227	0.0227
	80	0.9	20	0.7635	0.0185	0.0185	0.0185
	100	0.9	20	0.7674	0.0180	0.0180	0.0180
	10	0.9	50	0.8246	0.0942	0.0942	0.0942
	20	0.9	50	0.8293	0.0574	0.0574	0.0574
	30	0.9	50	0.8344	0.0411	0.0411	0.0411
Case II	50	0.9	50	0.8379	0.0278	0.0278	0.0278
	70	0.9	50	0.8382	0.0221	0.0221	0.0221
	80	0.9	50	0.8377	0.0180	0.0180	0.0180
	100	0.9	50	0.8393	0.0170	0.0170	0.0170
Case III	10	0.9	100	0.8985	0.1300	0.1300	0.1300
	20	0.9	100	0.8980	0.0718	0.0718	0.0718

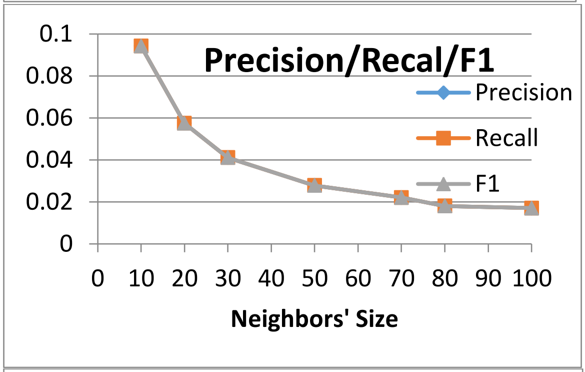
	30	0.9	100	0.8953	0.0650	0.0650	0.0650
	50	0.9	100	0.8993	0.0398	0.0398	0.0398
	70	0.9	100	0.9072	0.0339	0.0339	0.0339
	80	0.9	100	0.9101	0.0475	0.0475	0.0475
	100	0.9	100	0.9164	0.0524	0.0524	0.0524
Case IV	10	0.9	150	0.9543	0.1277	0.1277	0.1277
	20	0.9	150	0.9712	0.1111	0.1111	0.1111
	30	0.9	150	0.9712	0.1111	0.1111	0.1111
	50	0.9	150	0.9712	0.1111	0.1111	0.1111
	70	0.9	150	0.9712	0.1111	0.1111	0.1111
	80	0.9	150	0.9712	0.1111	0.1111	0.1111
	100	0.9	150	0.9712	0.1111	0.1111	0.1111

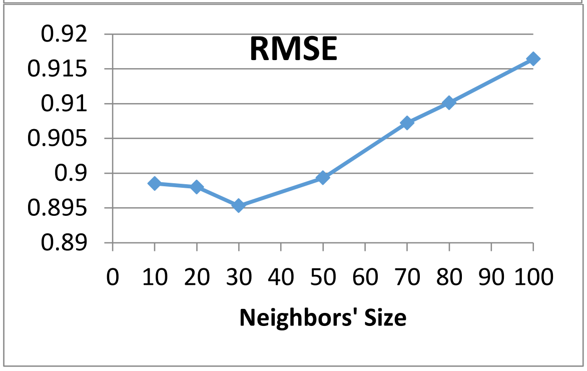
The relation between the size of neighbors and the accuracy of RMSE, Precesion, and Recall is illustrated in Fig5 (a,b), (c,d), (e,f), and (g,h). In most cases, the accuracy of RMSE shows that if the number of neighbors is low then the accuracy of RMSE is high. As can be seen in Table 5 The best recommendation accuracy of lowest RMSE and Highest Precision and Recall we got is in Case 1 where the parameters of neighbors' size = 10 and *n* = 20. This is because the proposed method filters the dataset into 10 neighbors who shared same preferences as an active user on 20 items. This action prevent any user who does not have any shared preferences with active user to be part of the training/test and leads to adding more ratings records in our dataset. This increases similarity measures between the active user and his/her chosen 10 neighbors. Meanwhile, leads to high prediction accuracy.






RMSE
0.845
0.84
0.835
0.83
0.825
0.82
0
20
40
60
80
100
Neighbors' Size


Precision/Recal/F1
Precision
Recall
F1
0.1
0.08
0.06
0.04
0.02
0
0 10 20 30 40 50 60 70 80 90 100
Neighbors' Size


RMSE
0.92
0.915
0.91
0.905
0.9
0.895
0.89
0 10 20 30 40 50 60 70 80 90 100
Neighbors' Size

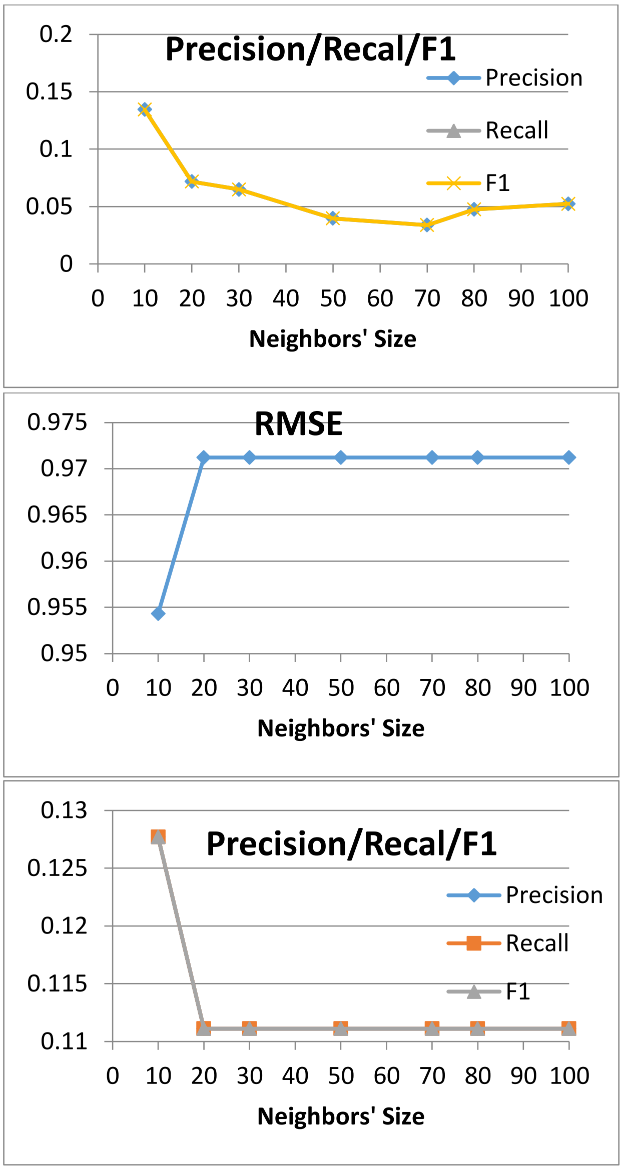


Fig5. The relation between the size of neighbors and the accuracy of RMSE, Precesion, and Recall

Table 6a. The Lowest RMSE of all Cases

Neighbors size	Training	θ	RMSE

Case I	10	0.7, 0.9	20	0.7185
Case II	10	0.9	50	0.8246
Case III	10	0.9	100	0.8953
Case IV	10	0.9	150	0.9543

Table 6b.The Highest Precision, Recall, and F1 of all Cases

	Neighbors size	Training	θ	Precision, Recall, & F1
Case I	10	0.7, 0.9	20	0.1886
Case II	10	0.9	50	0.0965
Case III	10	0.9	100	0.1345
Case IV	10	0.9	150	0.1384

4.6 Models and Comparison

To evaluate its performance, our proposed method is compared against several recommender algorithms that widely used: DSTNMF and DSMMF [23], GWNMF [24], OWNMF [25–27], WNMF [28], ALS [29], SCoR [30], and Apache Mahout [9], Slope One [31], and SVD++ [14]. Most chosen algorithms for the comparison are Matrix Factorization, since it is the most recent approach that offers more accurate predictions compared to Collaborative filtering.

Name	Algorithm	RMSE
DSTNMF	Matrix Factorization	0.9385
DSMMF	Matrix Factorization	0.9483
GWNMF	Matrix Factorization	0.9660
OWNMF	Matrix Factorization	1.0168
WNMF	Matrix Factorization	1.0042
ALS	Alternating Least Squares	0.964
SCoR	Synthetic Coordinate	0.875
Apache Mahout	User-Based CF	1.0601
Slope One	Smoothing Bagging	0.9460
SVD++	Smoothing Bagging	0.9137
Proposed	User-Based CF	0.7185

Table 6 shows the performance of the compared recommender systems for the Movielens 100K dataset. According to these results, the proposed method gives the lowest value of RMSE which means performance.

Algorithm	precision	Recall
DSTNMF	0.0442	0.0259
DSMMF	0.0495	0.0272
GWNMF	0.0447	0.0266
OWNMF	0.1252	0.0625
WNMF	0.1208	0.0611
Apache Mahout	0.0358	0.0361
Proposed	0.1886	0.1886

5. Conclusion

We presented an algorithm that enhance the efficiency of collaborative filtering technique in recommendation systems. Apache Mahout which is a part of Hadoop is used in this work as a powerful tool to deal with the analysis of big data. The improved collaborative filtering which is based on the neighbors' preferences that shared in common with the active user. In other word, we count the tastes between the neighbors and the active user. The proposed method has been tested on real dataset. The proposed work is compared against literatures explained in pervious sections, proving its effectiveness, simplicity and higher performance. Under several cases, our proposed work is the first in performance. Apart from the high performance and simplicity of this work, other advantages of the proposed method compared to some literatures algorithms are that it does not require any additional data. Furthermore, the parameter chosen here is simple in very easy for execution. Additionally, the proposed system outperforms the previous literature in finding the best users chosen to be the active user’s neighbors list which provided more accurate prediction.

In future work, we plan to extend the framework to test more real datasets to prove its effectiveness. In addition, to better handle sparsity problem as well as cold-start. An important axis, Item-Based CF will be applied too in this method for future work.

6. References

1- Su, X., & Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. Artificial Intelligence Review, 31(1), 103-132.

2- Zhang, X., & Liu, H. (2016). A technique of recursive reliability-based missing data imputation for collaborative filtering. Applied Sciences, 6(8), 3719.

3- Su, X., Khoshgoftaar, T. M., Zhu, X., & Greiner, R. (2008). Imputation-boosted collaborative filtering using machine learning classifiers. In Proceedings of the 2008 ACM symposium on applied computing (pp. 111-117). ACM.

4- Deng, J., Ye, Z., Shan, L., You, D., & Liu, G. (2022). Imputation method based on collaborative filtering and clustering for the missing data of the squeeze casting process parameters. Integrating Materials and Manufacturing Innovation, 11(1), 95-108.

5- Zhang, Y., Chen, L., & Liang, Y. (2019). Deep learning-based imputation for missing values in high-dimensional data sets: A review. Neurocomputing, 370, 77-87.

6- Wang, X., Liu, J., & Liang, Y. (2021). Collaborative Filtering with Matrix Factorization and Imputation Techniques for Sparse Data Sets. IEEE Access, 9, 10762-10772.

7- Wu, C.-S. M., Garg, D., & Bhandary, U. (2018). Movie recommendation system using collaborative filtering. In 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS) (pp. 11-15). IEEE. DOI: 10.1109/ICSESS.2018.8663822.

8- Petrozziello, A., Jordanov, I., & Sommeregger, C. (2018). Distributed neural networks for missing big data imputation. In 2018 International Joint Conference on Neural Networks (IJCNN) (pp. 1-8). IEEE. DOI: 10.1109/IJCNN.2018.8489488

9- S. Owen and S. Owen, "Mahout in action," 2012.

10- G. Research, "Movielens Dataset 100K," 1998.

11- N. Rastin and M. Z. Jahromi, "Using content features to enhance performance of user-based collaborative filtering performance of user-based collaborative filtering," *arXiv preprint arXiv:1402.2145,* 2014.

12- J. Renuka. (2016). *Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures*.

13- A. Morad, Hassan; Ali, Mansoor, Alsahaq; Salem, Alsseed, Alatrash, "A Novel Imputation-Boosted Technique to Overcome the Unrated Items Issue and Improving the Performance of Collaborative Filtering," presented at the 11th International Conference on Data Mining, Computers, Communication and Industrial Applications (DMCCIA), Kuala Lumpur, 2017.

14- B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*, 2001, pp. 285-295.

15- M. S. G. Badrul, Karypis; Joseph, A. Konstan; John, T. Riedl, "Application of Dimensionality Reduction in Recommender System – A Case Study," *Proc. KDD Workshop on Web Mining for e-Commerce: Challenges and Opportunities (WebKDD), ACM Press.,* pp. 1-12, 2000.

16- D. M. R. N. Jasson, Srebro, "Fast maximum margin matrix factorization for collaborative prediction," in *ICML '05 Proceedings of the 22nd international conference on Machine learning*, Bonn, Germany, 2005, pp. 713-719.

17- Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer,* vol. 42, 2009.

18- T. Luo, S. Chen, G. Xu, and J. Zhou, *Trust-based collective view prediction*: Springer, 2013.

19- F. Ricci, L. Rokach, and B. Shapira, *Introduction to recommender systems handbook*: Springer, 2011.

20- P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: an open architecture for collaborative filtering of netnews," in *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, 1994, pp. 175-186.

21- T. Arsan, E. Köksal, and Z. Bozkus, "Comparison of Collaborative Filtering Algorithms with Various Similarity Measures for Movie Recommendation," *International Journal of Computer Science, Engineering and Applications (IJCSEA),* vol. 6, pp. 1-20, 2016.

22- E. Rashid and E. Rashid, "Enhancing Software Fault Prediction With Machine Learning: Emerging Research and Opportunities," 2017.

23- Y. Li, D. Wang, H. He, L. Jiao, and Y. Xue, "Mining intrinsic information by matrix factorization-based approaches for collaborative filtering in recommender systems," *Neurocomputing,* vol. 249, pp. 48-63, 2017.

24- Q. Gu, J. Zhou, and C. Ding, "Collaborative filtering: Weighted nonnegative matrix factorization incorporating user and item graphs," in *Proceedings of the 2010 SIAM International Conference on Data Mining*, 2010, pp. 199-210.

25- G. Chen, F. Wang, and C. Zhang, "Collaborative filtering using orthogonal nonnegative matrix tri-factorization," *Information Processing & Management,* vol. 45, pp. 368-379, 2009.

26- C. Ding, X. He, and H. D. Simon, "On the equivalence of nonnegative matrix factorization and spectral clustering," in *Proceedings of the 2005 SIAM International Conference on Data Mining*, 2005, pp. 606-610.

27- C. D. T. L. W. P. H. Park, "Orthogonal Nonnegative Matrix Tri-Factorizations for Clustering," presented at the Proceeding of the 2006 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2006.

28- S. Zhang, W. Wang, J. Ford, and F. Makedon, "Learning from incomplete ratings using non-negative matrix factorization," in *Proceedings of the 2006 SIAM International Conference on Data Mining*, 2006, pp. 549-553.

29- Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan, "Large-scale parallel collaborative filtering for the netflix prize," in *International Conference on Algorithmic Applications in Management*, 2008, pp. 337-348.

30- H. Papadakis, C. Panagiotakis, and P. Fragopoulou, "SCoR: A Synthetic Coordinate based Recommender system," *Expert Systems with Applications,* vol. 79, pp. 8-19, 2017.

31- D. Lemire and A. Maclachlan, "Slope one predictors for online rating-based collaborative filtering," in *Proceedings of the 2005 SIAM International Conference on Data Mining*, 2005, pp. 471-475.