A REPORT ON

“Group-based Collaborative Filtering Supported by Multiple User’s Feedback to Improve Personalized Ranking”

Submitted to: **Dr. Poonam Goyal**

Department of Computer Science,

Birla Institute of Technology and Science Pilani, Pilani campus

Submitted by:

KAPIL AGRAWAL (2014B3A30579P) **MANISH SHARMA (2014A3PS0181P)**

PROBLEM STATEMENT

Recommending music artists to users using user based collaborative filtering on groups made using clustering approach.

Source Code:

The entire source code, dataset and related files of this assignment is available on the following link.

https://github.com/iammangod96/Group_Based_Recommender_System

DESCRIPTION OF SELECTED APPLICATION DOMAIN

On the Internet, where the number of choices is overwhelming, there is need to filter, prioritize and efficiently deliver relevant information in order to alleviate the problem of information overload, which has created a potential problem to many Internet users. Recommender systems solve this problem by searching through large volume of dynamically generated information to provide users with personalized content and services. Recommender systems were created to represent user preferences for the purpose of suggesting items to purchase or examine. Also, recommender system was defined from the perspective of E-commerce as a tool that helps users search through records of knowledge which is related to users' interest and preference.

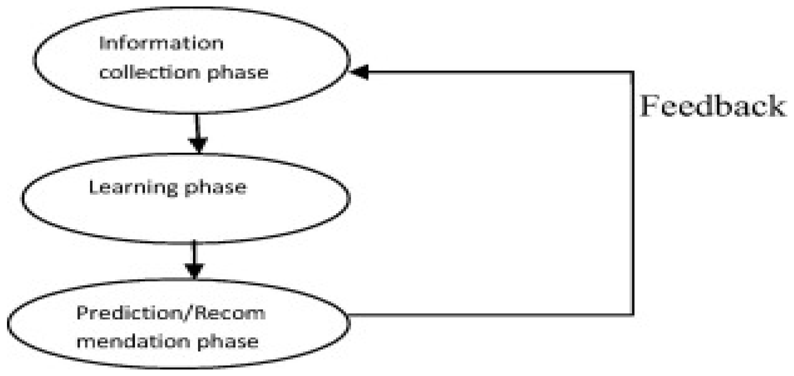
Phases in any recommendation problem:

Information Collection Phase – Different types of feedback information is collected in this phase. The types of feedback are implicit, explicit, hybrid.

Learning Phase – We need to apply any learning algorithm here to filter and extract user's features from the information and feedback gathered in the previous information collection phase.

Recommendation Phase – In this phase, we try to recommends or predicts what type of items the user may prefer. This can be done either directly on the basis of the set of data collected in the information gathering phase, which can be based on the memory or the model or through the activities observed by the user.

Flowchart representing all the phrases

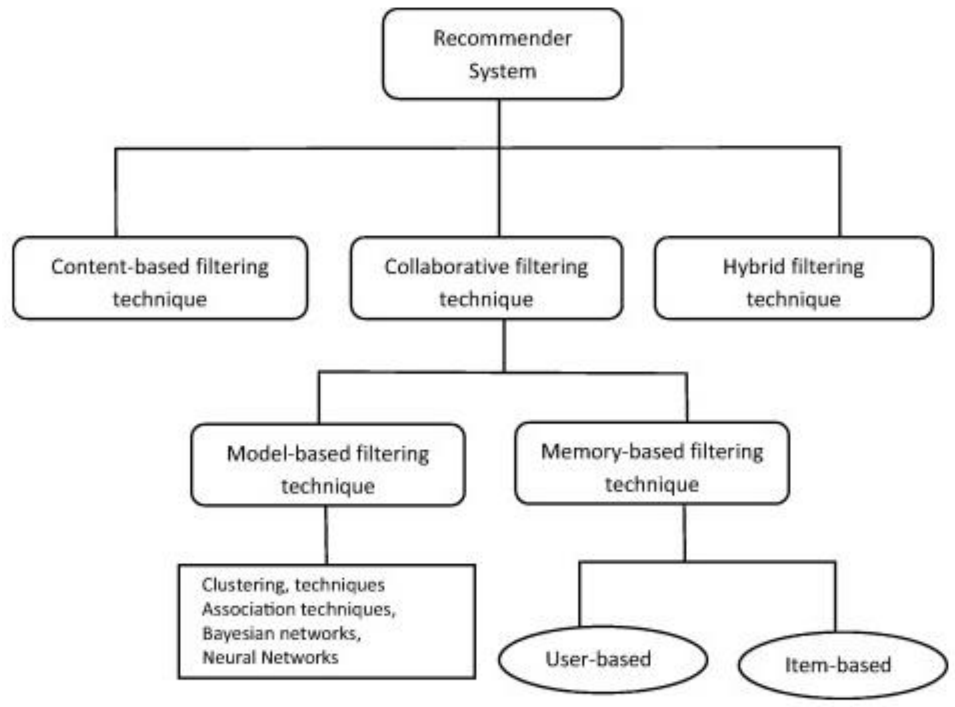

Information collection phase
Learning phase
Prediction/Recom mendation phase
Feedback

Recommendation systems use several different technologies. We canclassify these systems into two broad groups:

• Content-based systems examine the properties of recommended articles. Byexample, if a Netflix user has seen many cowboy movies, then recommend a movie classified in the database that has the genre "cowboy".

• Collaborative filtering systems recommend articles based on measures of similarity between users and / or articles. The recommended elements for a user are those preferred by similar users.

Recommendation filtering techniques


Recommender System
Content-based filtering technique
Collaborative filtering technique
Hybrid filtering technique
Model-based filtering technique
Memory-based filtering technique
Clustering, techniques
Association techniques,
Bayesian networks,
Neural Networks
User-based
Item-based

MOTIVATION OF THE PROBLEM

Here in this paper, we propose a collaborative filtering approach based on preferences of groups of users to improve the accuracy of recommendation,where the distance among users is computed using multiple types of users' feedback. The traditional methods used in recommendation system does not use clustering techniques, but if we can use it to make the model more precise and accurate. The main advantage of using this approach is that relevant items will be suggested based only on the subjects of interest of each group of users. Using this technique, we use a state-of-art collaborative filtering algorithm to generate a personalized ranking of items according to the preferences of an individual within each cluster. The experimental results show that the proposed technique has a higher precision than the traditional models without clustering.

TECHNICAL ISSUES

1) We initially tried to use sparse matrix to represent the user-item matrix. But computing recommendation algorithm over that was taking too much time aside from the fact that it was also space intensive. We used pivot_table of python for that.
2) R stores all variables in RAM, so we had to switch to python for programming. (RAM limited to 4GB in our case)
3) Artists.dat is not complete dataset. We could not use it since artists dataset had only 14036 artists compared to 17632 artists in other datasets.

RELATED WORK: LITERATURE SURVEY

Multiple Interactions Based Approaches:With the increasing number of interactions between users and content, several studies have emerged in order to work with the integration of these interactions, so that more information about the user’s preferences are gathered by the systems. The recommendation systems can be extended in various ways in order to improve the understanding of users and items, including, for example, new types of interaction in the recommendation process and making the combination of them [13]. The SVD algorithm proposed uses explicit (ratings) and implicit (viewing history) information from users in a factorization model, called SVD++. Another factorization model, called Factorization Machines, can consider many types of information regarding users, items and/or their interactions. These techniques have the drawback that they process only certain types of interactions, with little capability of extension to other different types.

Data Mining Based Approaches:In this study, we have used data mining techniques to cluster users with similar preferences to generate recommendations based on their group. Several researchers have proposed recommender systems for online personalization through data mining to provide recommendation services. This kind of recommendation system is used to predict the user navigation behavior and their preferences using web log data. The algorithm uses the K-Means Grouping method to reduce the search space. Next, use a graphic approach to the best cluster with respect to a given test client in the selection of neighbors with greater similarities as well as minor similarities. The graphic approach allows us exploit the transitivity of similarity. A threshold of similarity limits the similarity between clusters. Calculating the similarity between the current element and the center of the group, choose the best similitude cluster, and then find the target elements' closest neighbors.

SYSTEM DESCRIPTION:

Here,theapproach used is different than the above related work in the sense that we analyze various paradigms of users' interactions on a particular item, in order to create groups of users with similar preferences and thus, a more accurate personal profile. The advantages of this approach is the ease of extending the template for insertion of other types of interactions; the reduced time and computational processing, considering that the sets of data would be reduced to a single cluster before computing the recommendation. Our contribution, therefore, can be considered the proposal of a recommender system based on multiple feedback types of similar users' groups.

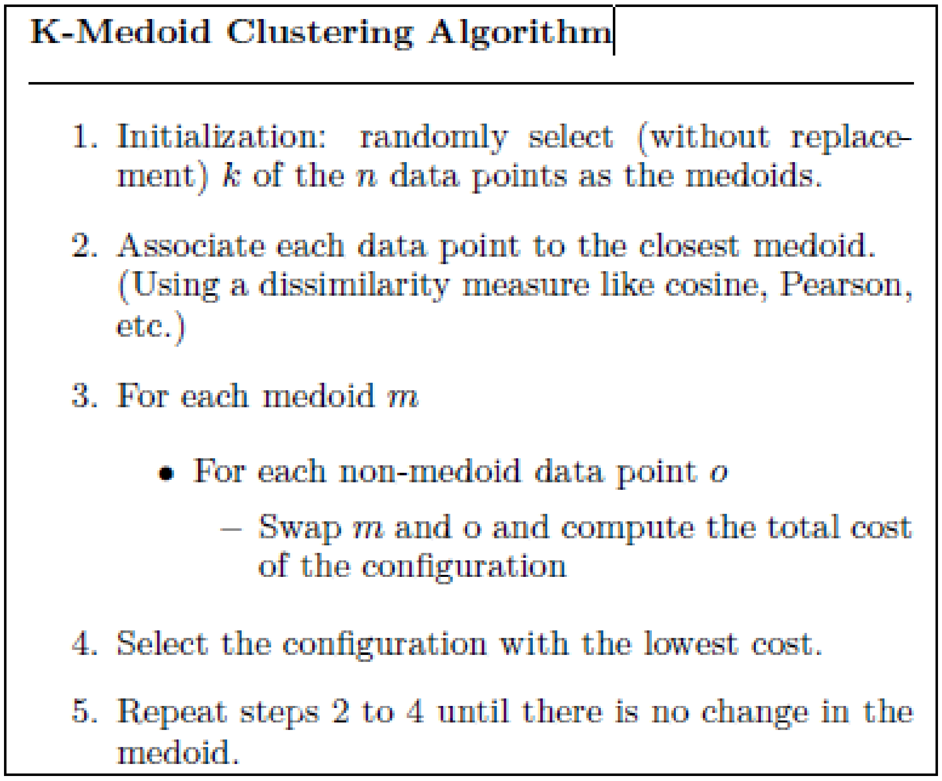
Notations:

We use special indexing letters to distinguish users and items: a user is indicated as u and an item is referred as i, j; and rui is used to refer to either explicit or implicit feedback from a user u to an item i. In the first case, it is an integer provided by the user indicating how much he liked the content; in the second, it is just a boolean indicating whether the user consumed or visited the content or not. The prediction of the system about the preference of user u to item i is represented by ˆrui, which is a floating point value guessed by the recommender algorithm. The set of pairs (u, i) for which rui is known are represented by the set K = {(u, i)|rui is known}. Additional sets used in this paper are: N(u) to indicate the set of items for which user u provided an implicit feedback, and ¯N (u) to indicate the set of items that are unknown to user u.

k-medoids Clustering Algorithm:

Clustering is the process of grouping a set of objects into clusters so that objects within a cluster are similar to each other but are dissimilar to objects in other clusters. Kmeans clustering and partitioning around medoids are well known techniques for performing non-hierarchical clustering. K-means clustering iteratively finds the *k* centroids and assigns every object to the nearest centroid, where the coordinate of each centroid is the mean of the coordinates of the objects in the cluster. Unfortunately, K-means clustering is known to be sensitive to the outliers although it is quite efficient in terms of the computational time. For this reason, K-medoids clustering are sometimes used, where representative objects called medoids are considered instead of centroids. Because it is based on the most centrally located object in a cluster, it is less sensitive to outliers in comparison with the K-means clustering.

The k-medoids clustering algorithm is a partition-based cluster algorithm based on k-means. Try to minimize the distance between the points labeled to be in a cluster and a point designated as the center of that group. In contrast with the k-means algorithm, k-medoids choose data points as centers (or medoids) and works with an arbitrary matrix of distances between data points. It is more robust to noise and atypical compared to k-means because it minimizes a sum of dissimilarities by pairs in place of a sum of Euclidean square distances. A medoid can be defined as the object of a group whose average dissimilarity for all objects in the cluster is minimal, that is, it is a most central point located in the group. The implementation of the K-medoids used in this document is explained as follow:



K-Medoid Clustering Algorithm

1. Initialization: randomly select (without replacement) k of the n data points as the medoids.
2. Associate each data point to the closest medoid. (Using a dissimilarity measure like cosine, Pearson, etc.)
3. For each medoid m
    - For each non-medoid data point o
        - Swap m and o and compute the total cost of the configuration
4. Select the configuration with the lowest cost.
5. Repeat steps 2 to 4 until there is no change in the medoid.

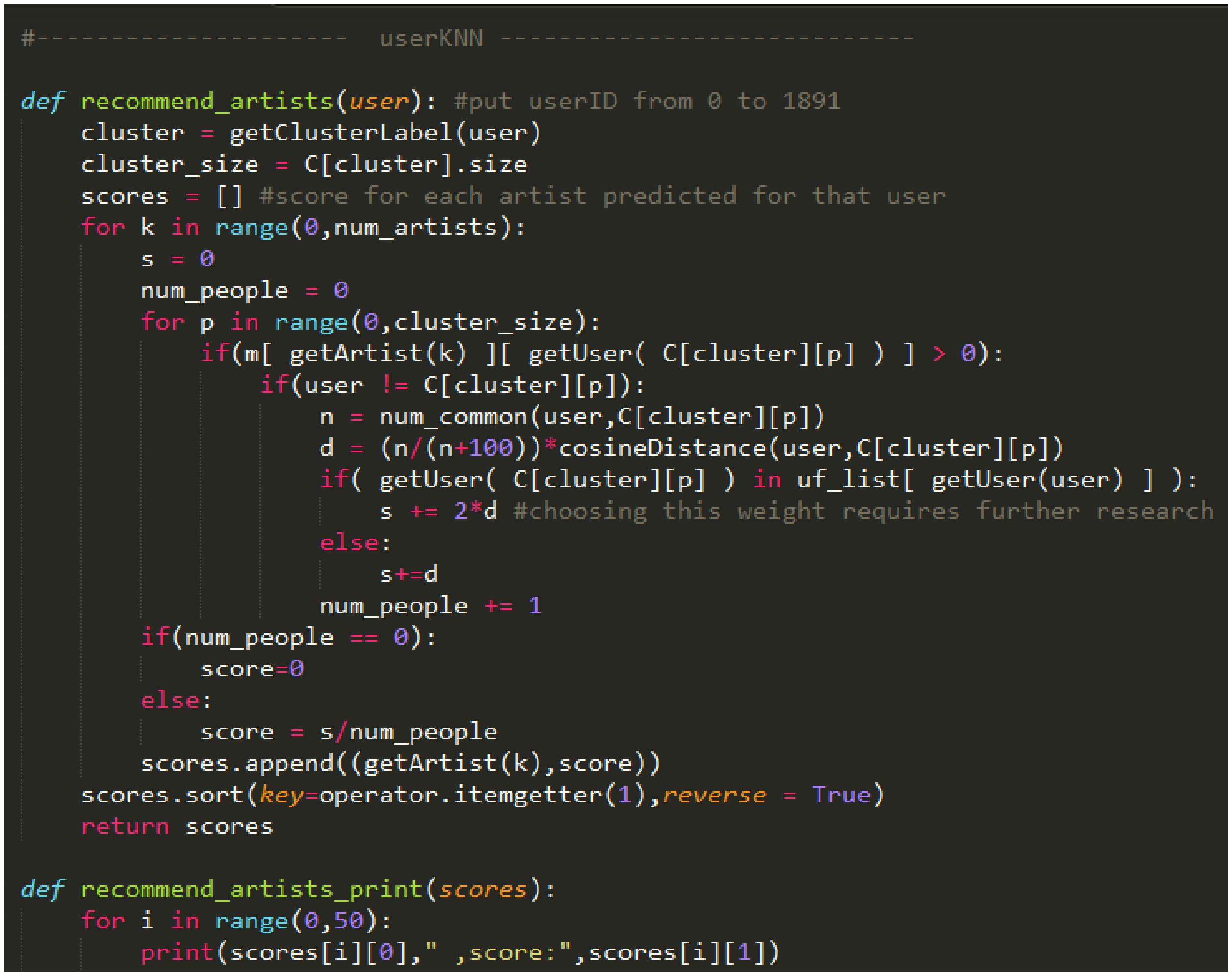
The advantage of this implementation of k-medoids is that large datasets can be classified efficiently and their convergence it is tested independently of the measure of dissimilarity. Further, it is simpler and faster. In this work, to generate user groups, we use the k-medoids algorithm to group users with similar preferences. This algorithm was chosen to accept different types of distance and not just metrics, like Manhattan and Euclidean, that They do not work well for recommendation approaches.

UserKNN Recommendation Algorithm:

We are using User kNN clustering algorithm. The first one is used to generate groups based on the similarity among users, while the second and the third are used to generate recommendations based on interactions of each group.

$$s_{uv} = \frac{n_{uv}}{n_{uv} + \lambda_1} p_{uv},$$

$$\hat{r}_{ui} = \frac{\sum_{v \in S^k(i;u)} s_{uv}}{Number\ of\ neighbors} \quad ,$$

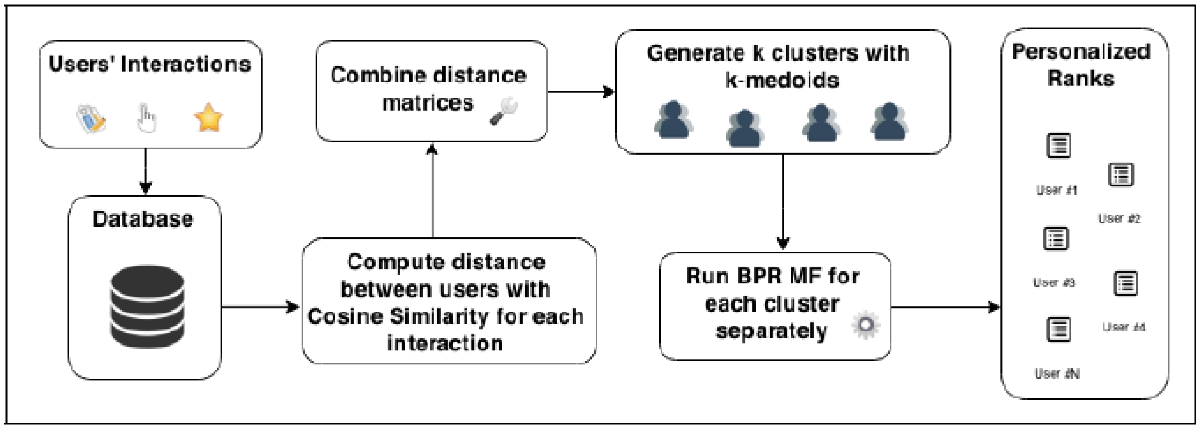


```python
#---------------------  userKNN ----------------------------

def recommend_artists(user): #put userID from 0 to 1891
    cluster = getClusterLabel(user)
    cluster_size = C[cluster].size
    scores = [] #score for each artist predicted for that user
    for k in range(0,num_artists):
        s = 0
        num_people = 0
        for p in range(0,cluster_size):
            if(m[ getArtist(k) ][ getUser( C[cluster][p] ) ] > 0):
                if(user != C[cluster][p]):
                    n = num_common(user,C[cluster][p])
                    d = (n/(n+100))*cosineDistance(user,C[cluster][p])
                    if( getUser( C[cluster][p] ) in uf_list[ getUser(user) ] ):
                        s += 2*d #choosing this weight requires further research
                    else:
                        s+=d
                    num_people += 1
        if(num_people == 0):
            score=0
        else:
            score = s/num_people
        scores.append((getArtist(k),score))
    scores.sort(key=operator.itemgetter(1),reverse = True)
    return scores

def recommend_artists_print(scores):
    for i in range(0,50):
        print(scores[i][0]," ,score:",scores[i][1])
```

Block Diagram of the proposed technique



Users' Interactions
Database
Combine distance matrices
Compute distance between users with Cosine Similarity for each interaction
Generate k clusters with k-medoids
Run BPR MF for each cluster separately
Personalized Ranks
User #1
User #2
User #3
User #4
User #N

DATASET USED

Dataset Last.fm website, http://www.lastfm.com

This dataset contains social networking, tagging, and music artist listening information from a set of 2K users from Last.fm online music system. http://www.last.fm.com

Files used-

user_artists.dat - This file contains the artists listened by each user. It also provides a listening count for each [user, artist] pair.

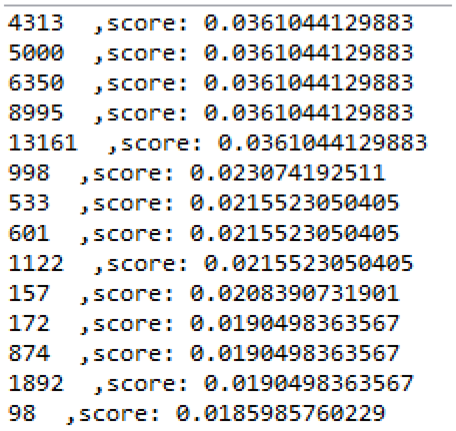
user_friends.dat - These files contain the friend relations between users in the database.

EVALUATION STRATEGY

The recommendation system is evaluated using **precision** metric. For a new user instance, the system recommends music artists using decreasing order of score.

To calculate precision, we check how many of the recommended artists did the new user actually listened to.

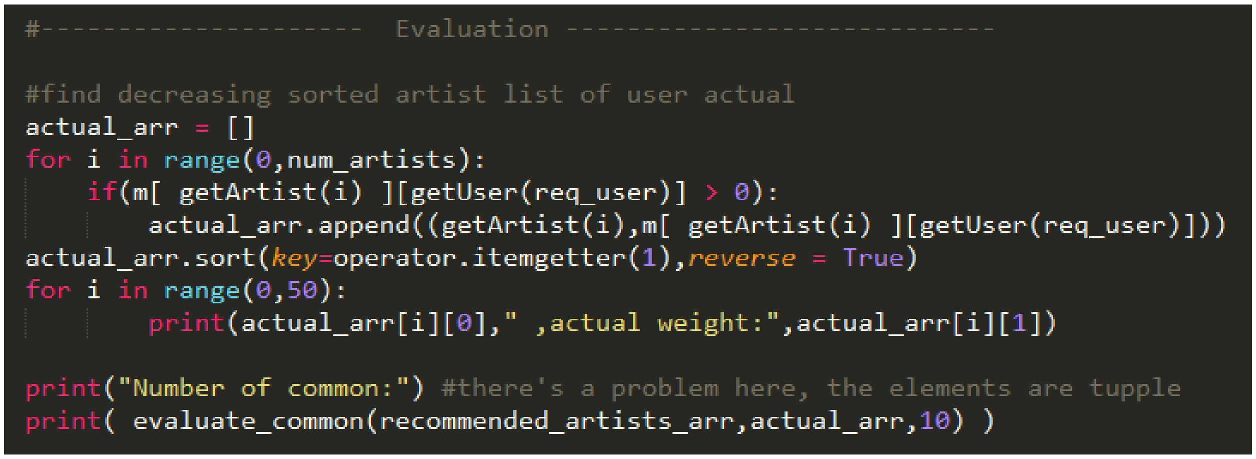
EXPERIMENTAL RESULTS AND EVALUATION



```
4313  ,score: 0.0361044129883
5000  ,score: 0.0361044129883
6350  ,score: 0.0361044129883
8995  ,score: 0.0361044129883
13161  ,score: 0.0361044129883
998  ,score: 0.023074192511
533  ,score: 0.0215523050405
601  ,score: 0.0215523050405
1122  ,score: 0.0215523050405
157  ,score: 0.0208390731901
172  ,score: 0.0190498363567
874  ,score: 0.0190498363567
1892  ,score: 0.0190498363567
98  ,score: 0.0185985760229
```

Recommended artists with corresponding scores

Precision comes around 0.1.



```
#--------------------  Evaluation ---------------------------

#find decreasing sorted artist list of user actual
actual_arr = []
for i in range(0,num_artists):
    if(m[ getArtist(i) ][getUser(req_user)] > 0):
        actual_arr.append((getArtist(i),m[ getArtist(i) ][getUser(req_user)]))
actual_arr.sort(key=operator.itemgetter(1),reverse = True)
for i in range(0,50):
        print(actual_arr[i][0]," ,actual weight:",actual_arr[i][1])

print("Number of common:") #there's a problem here, the elements are tupple
print( evaluate_common(recommended_artists_arr,actual_arr,10) )
```

IMPROVEMENTS DONE

The last.fm dataset contains data of user friends. Every userID has a set of friendID associated with it. We have incorporated this data in the recommendation algorithm by giving it a weight.

We propose this improvement since friends can have an influence on the person's likes and music taste aside from the fact that many a times friends suggest new music themselves.

Therefore using it can lead to better recommendation. And further research can be done on how can we better incorporate this data or how can we find the best weight for this factor in the recommendation algorithm.

CONCLUSION AND FUTURE WORK

It is very crucial for a recommender system to have the capability of making accurate prediction by analyzing and retrieving customer's preferences. Although collaborative filtering is widely used in recommender systems, some efforts to overcome its drawbacks have to be made to improve the prediction quality. Selecting the proper neighbors plays an important role to improving the prediction quality. We have proposed a technique that yields better recommendation accuracy using users' groups, generated from the similarity between them. It considers different type of interactions of each user for customer's preferences and low similarities as well. The experimental results showed that our algorithm provides the better prediction accuracy than baselines in two datasets. The main advantages of our approach are extensibility and flexibility, once it enables developers to use different recommender and clustering algorithms, dissimilarity metrics and different types of feedback. As future work, we plan to evaluate our approach with additional datasets from other domains in order to check the accuracy with different information types. Furthermore, we plan to consider community detection in graphs to select better users' groups to make the recommendation.