PROJECT REPORT

SONG RECOMMENDATION SYSTEM

Team Members:

Maneesh Boruthalupula

Nikhil Satish Pai

Pallavi Mahajan

Poojan Umesh Desai

Shashank Goud Pulimamidi

Table of Contents

1. Project Idea and Description.....……...1
2. Data and Methodology ……..1
    2.1. Dataset Description ……1
    2.2. Classification Methodology ……...2
        2.2.1. K- Nearest Neighbours Algorithm ……..2
        2.2.2. Collaborative Filtering Techniques ……3
        2.2.3. Top N Popular Songs ……..4
3. Results ……...4
    3.1. Methodology for Evaluation of Algorithms ……..4
    3.2. Recommender Systems Performance ……5
        3.2.1. K-NN Based Recommender System ……...5
        3.2.2. User-based CF Recommender System ……6
        3.2.3. Item-based CF Recommender System ……6
    3.3. Comparison of Recommender Systems ……7
4. Discussion ……7
    4.1. Expected Versus Unexpected ……7
    4.2. Future Scope ……..8
5. References ……9

1. Project Idea and Description

In recent years, the music industry has shifted more and more towards digital distribution through online music stores and streaming services such as Pandora[1], iTunes[2], Spotify[3] and Google Play[4]. As a result, people now have access to music collections on an unprecedented scale.

In order to help users cope with the rapidly expanding catalogue of music, these services try to suggest music which the user would be more interested in buying and listening to. Thus, a good recommendation system should be able to minimize user's effort required to search for music and will also help the service providers in deriving a lot of financial gain and improving their customer base.

There are many interesting hurdles to tackle when building a music recommdation system. One such issue is that there is no straightforward similarity measure that can be derived between songs unlike the products of an ecommerce website. That is because two songs which may sound similar could have completely different audiences. Hence care must be taken while deriving the similarity between the users by considering various attributes. A lot of papers on music recommondation sysem talk about how a specific implementation will help recommend songs to the user and there is less work done in comparing these algorithms.

Our project – **"Song recommendation system"** will attempt to generate a list of top-n songs for each user by learning their listening history and deriving similar songs. We used three main classification techniques namely, K-Nearest Neigbour[5], User-based and Item-based collaborative filtering[6] algorithms to learn the historical data and make recommendations. We also used a simple top-n algorithm which can be used as a base case. We also compare the precision of each algorithm based on the songs recommended to the user. We used a Million Song Dataset[7] which is a freely available collection of data for a million of contemporary popular music tracks.

The rest of this paper is organized as follows. In Section 2, we describe the dataset and the algorithms that we used to recommend the songs. In Section 3 we provide the results and compare the precision of the different classification methods used. In Section 4 we discuss the behaviour of the algorithms and conclude by discussing possible future work.

2. Data and Methodology

2.1 Dataset Description

We are using the Million Song Dataset [8] provided by Kaggle [9] in our experiments. The data set consists of the listening history of 1 million users. It has the following three attributes.

user_id - 40 digit alpha numeric unique identifier for each user.
song_id - 18 digit alpha numeric identifier for each song.
play_count - number of times a given song is played by a user.

user_id	song_id	Play_count
00014a76ed063e1a749171a253bca9d9a0ff1782	SOCCSRQ12AB01828CE	10
001599607ba12f1aa6d6ffe3237e00c4dda0055c	SOUFXCG12A8C138150	1

Table 2.1 Example

Due to constraints on memory and processing time, only a subset of the data was considered for processing.

We divided the original data set into three sets as 'Test Visible Data', 'Test Invisible Data' and 'Train Data' where **Train data** is the part of data set which is used to build the model based on the listening history of users .**Test Visible data**, it is the part of the test data using which our recommendation system recommends or predicts the remaining songs or listening history of the user. **Test Invisible Data** is the part of test data which should be predicted by the recommender system.

Type of Data	Number of Records
Train Data	4,59,700
Test Data	12,148

Table 2.2 Data Summary

2.2 Classification Methodology

We have used three algorithms to build the recommendation system – K-NN, Collaborative filtering and Top-n. All three classifiers are built using R programming language.

2.2.1 K- Nearest Neighbours Algorithm

For a given test user 'U' we determine 'K' users with similar interest in songs. After the list of K users has been formulated the vector consists of the similarity scores for each song. Now we merge the listening scores to form a cumulative score for the test user U. Finally sorting the scores for this user will give the list of interesting or potentially wanted songs.

The algorithm is described below

- Build an attribute vector for all users in the dataset. This will consist of songs and their corresponding play counts.
- Find the nearest neighbours of the test data in the training data by considering the cosine similarity [10] as the distance metric between corresponding attribute vectors.
- The top 'K' most similar users are selected and their listening histories are merged by summing up the play counts of each corresponding song to form the recommended songs list.
- The recommended songs list is sorted in decreasing order of the play counts and the songs are returned as the recommendations.

2.2.2 Collaborative Filtering Techniques

The collaborative filtering is the method and process used to match data of one user with data for other similar users

We have used memory-based collaborative filtering technique since it is efficient and works well with large data sets. It tries to find users that are similar to the test user (i.e. the users we want to make predictions for), and uses their listening history to predict songs for the test user based on a similarity measure. We are using cosine similarity to find the similar users/songs to predict the list of songs

We have used two variants of memory-based collaborative Filtering as:

i. User-based CF
ii. Item-based CF

i. User -based CF

In this approach, we find other users in memory which are similar to the given test user, then based on the listening history of the similar users a list of songs is recommended.

The algorithm is described below:

- First we build a user-to-user similarity matrix. The distance is measured using the cosine similarity.
- Then using the user-to-user similarity matrix, we find a set of users that are a close match to the test user with respect to the listening history.
- Make the list of all songs of all similar users found in above step and remove the songs which are present in visible list of test user.
- Rank all the songs in the list based on their similarity scores and return the top N songs as the recommended songs for the test user.

The similarity between two users say 'U_1' and 'U_2' is calculated using the formula

$$\textbf{Similarity } (U_1, U_2) = \frac{\textbf{count of common items between U1 and U2}}{(\textbf{Number of items of U1})^{\alpha} * (\textbf{number of items of U2})^{(1-\alpha)}}$$

Where α is a weight factor bounded by [0, 1) [10] which can be varied according to the listening history of the user, larger the listening history of the user higher is α value chosen.

We calculate the similarity score for each song 'S' using the formula
Weight(S) =∑Similarity $(U_1, U_2)^{\Upsilon}$ for all U_i in the training dataset

A normalization factor 'ϒ' is used to maximize the impact of higher weights and minimize that of lower weights. We finally recommend the top songs by sorting them in the decreasing order of their scores.

ii. Item-based CF

When a user listens to a particular song then the probability of him liking another similar song is high and this can be used to recommend him songs.

The algorithm is described below:

- Build the song-to-song similarity matrix based on cosine similarity for all of the users.
- Recommend similar songs to by checking the songs in the users listening history with the values in the song-to-song similarity matrix
- Sort the similarity score in descending order and return the recommended songs.

The formula for cosine similarity is given by

$$\textbf{Similarity } (S_1, S_2) = \frac{\textbf{number of common users for S1,S2}}{(\textbf{number of users for S1})^{0.5} * (\textbf{number of users for S2})^{(0.5)}}$$

S_1 – set of songs in the users listening history
S_2 – set of songs in the training data

The final score is calculated as a weighted sum given by **Weight** $= \sum$ **Similarity**(S_1, S_i) *for all* S_i in the listening history of the user.

2.2.3 Top N Popular Songs:

It is a classification method based frequency of songs instead of the similarity measure. In this algorithm we return the top N songs listened by the users in the Train Data Set. These songs are sorted in the decreasing order of their frequencies.

3. Results

We have used a single machine, with the configuration 16GB RAM, Intel(R) Core 'i7' @3.4 GHZ to build and test all the models.

3.1 Methodology for Evaluation of Algorithms

We evaluate the performance of different algorithms by taking the list of recommended songs for a user and comparing it against the list of songs in the test invisible data for that user. We find the precision for each of our recommendation system by finding the ratio of number of recommended songs present in the Test Invisible Data to the total number of songs present in the Test Invisible Data. Since a lower value of precision means higher degree of error in our recommendation system and vice versa which is the right measure to evaluate the performance of each algorithm.

3.2 Recommender Systems Performance

We tested the different recommender systems against different parameters and evaluated their performance.

3.2.1. K-NN Based Recommender System

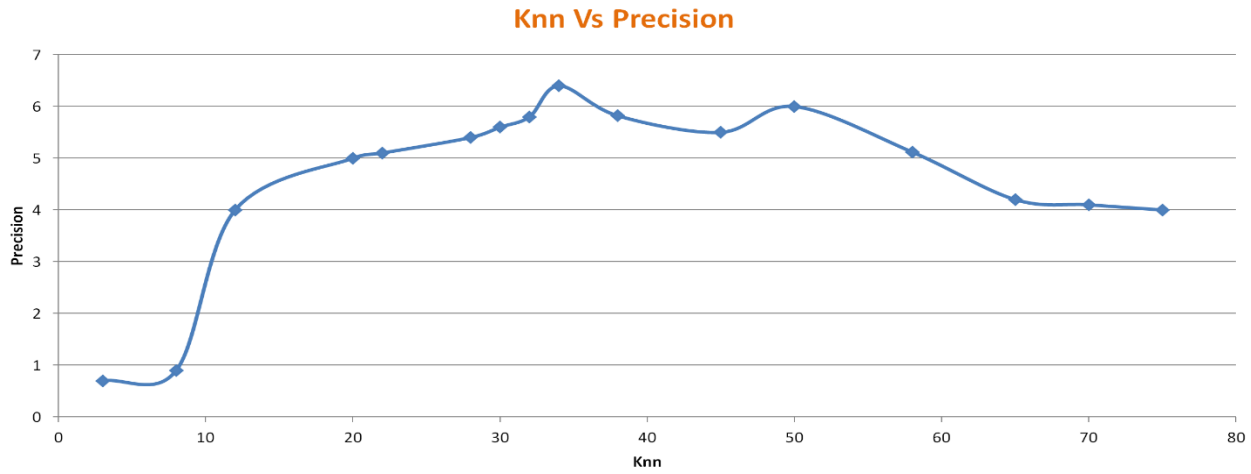

Figure 4.1: KNN based system for Precision versus value of k

For this system we have tested the model to find the optimal value of 'K' neighbours value that would give us maximum precision. It was seen that the precision of recommendation was low for small values of 'K'. The reason for this could be that the recommendation system is more susceptible to the noise in the training model. However with increase in value of 'K' there was increase in the precision of recommendation. But again with larger values of 'K' again low precision in recommendation was observed which could be because of inclusion less similar users in the set, as 'K' increases more users are considered and more songs are added into recommended list which degrades the performance. The graph in figure 1 shows the results of this observation.

3.2.2. User-based CF Recommender System

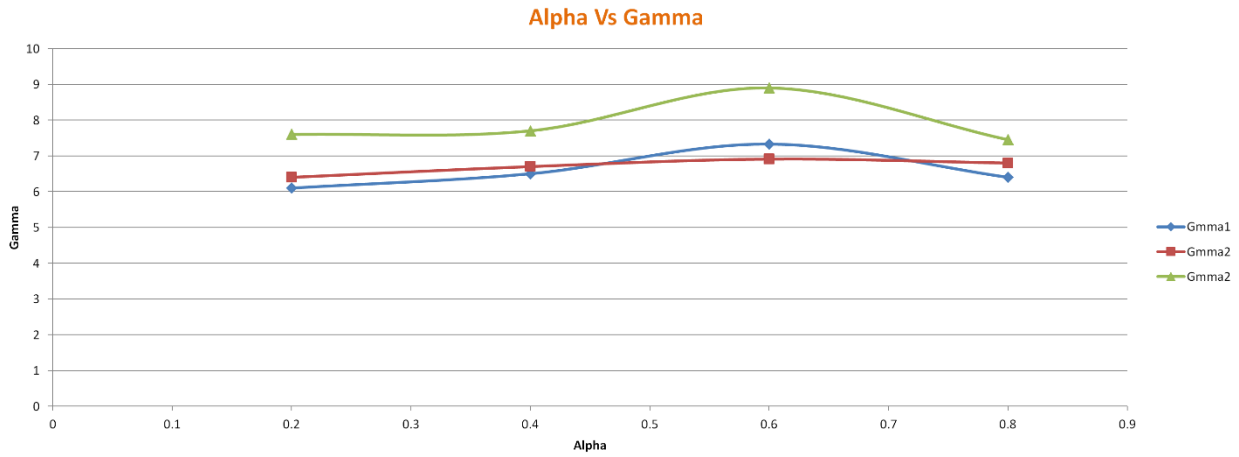

Figure 4.2: User-Based CF showing Alpha versus Precision for different values of Gamma

For this system we try to find the optimal values of the coefficients 'α' and 'ϒ' and see how the precision varies with varying their values. The graph in figure 2 shows the results of this observation. From the graph it's found that an optimal value for 'α' is 0.8 and that of 'ϒ' is 0.8.

3.2.3. Item-based CF Recommender System

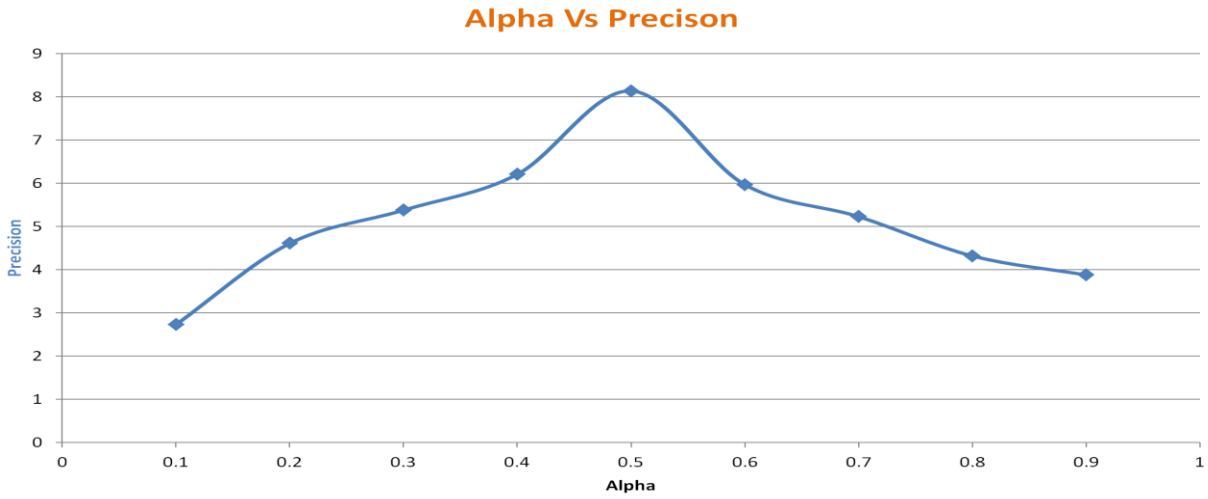

Figure 4.2: Item-Based CF showing Alpha versus Precision

For this system we try to find the optimal values of 'α' and see how the precision varies with varying its values. The graph in figure 3 shows the results of this observation. From the graph it's found that an optimal value for 'α' is 0.5.

3.3 Comparison of Recommender Systems

We evacuated the performance of three different algorithms user to build our recommender based on the Precision. The precision is computed by using the following formula,

$$\text{Precision} = \frac{\text{Number of songs matching with songs in } \textbf{Test Invisible Data for a user}}{\text{Total number of songs in } \textbf{Test Invisible Data} \text{ for a user}}$$

Algorithm	Precision
Top-N	0.7%
K-NN	6.4%
Item Based CF	8.14%
User Based CF	8.9%

Table 4.1: - Precision Summary of different algorithms

As visible from the above table, Collaborative approaches perform better than K-NN algorithm. This is because in K-NN we recommend the songs to a test user based on the listening history of 'K' neighbors by giving equal weights to each one of them, whereas in collaborative approaches we use the full dataset for predicting the songs by calculating the similarity matrix which can be with respect to either users or songs with varying weights, thus making it a better system. Among the two CF approaches user-based approach outperforms item-based approach. This is because item similarity matrix is much sparser than user similarity matrix because we used a small subset of the data set.

We found that for 10,000 users, each user listens to approximately 45-48 songs whereas each song is listened by only 4-5 users. Therefore user based approach works better here and item based would work in the case where data is dense and number of songs are comparable to number of users.

4. Discussion

4.1 Expected Versus Unexpected

Our project worked fairly as expected as the values of the hidden song list could be found in the predicted list when the visible was passed as parameters to the algorithm for a particular user.

Our expectations for the model behavior was as:

- Recommend similar songs to users based on their listening history.
- System should be able to predict the songs from invisible test data.
- The recommendation system would behave better on large and dense data set.

But the observed results of our system were as:

- The system classified songs into similar and dissimilar based on listening history.
- Using the similarity, the predicted songs were present in the invisible data set.
- The recommendation system did not show very high precision because of the sparse nature of the data set.

Due to memory constraints, we could not use the entire data set to train our model. The change in alpha value has a significant impact on the precision of the system. Ideally it is perceived that for small values of alpha one factor will diminish and the other will increase and the effect will be nullified because to a product rule. But in our system when the difference between 'α' and '1-α' is higher, precision decreases and error increases. Hence, we get a better precision when 'α' is closer to 0.5.

4.2 Future Scope

Our model has few limitations such as sparsely distributed dataset and the new songs in test data which are never seen in training data. Our present system can be extended for future by adding the following features:

Using the larger data set: Due to memory constraints we have build our model using a subset of the actual data set. We tested our code on 16GB of memory but with better systems, we could build a higher precision recommendation system.

More number of attributes in the data set: Using more descriptive attributes like genre, artist, lyrics of the song can help to build a better recommender system. They can add more weight to the similarity measure and may help to predict a better list of songs.

Using distributed programming model: We could use a distributed programming model [11] like map-reduce to perform parallel processing in future.

Using Hybrid Model: In our project we have used each algorithm to build a model separately, we could combine two approaches like user and item based CF to build a higher precision recommender system in future.

5. References

[1] http://www.pandora.com/
[2] https://www.apple.com/itunes/
[3] https://www.spotify.com/us/
[4] https://play.google.com/store/apps/details?id=com.google.android.music&hl=en
[5] http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
[6] http://en.wikipedia.org/wiki/Collaborative_filtering
[7] http://labrosa.ee.columbia.edu/millionsong/
[8] The million song dataset challenge" Proc. of the 4th International Workshop on Advances in Music Information Research (AdMIRe), 2012
[9] http://www.kaggle.com/c/msdchallenge
[10] http://en.wikipedia.org/wiki/Cosine_similarity
[11] http://webee.technion.ac.il/~idish/sigactNews/#column%2032