

MUSIC RECOMMENDATION SYSTEM

Nachiket Bhagwat

ID: 105828558

University of Colorado, Boulder

nabh1518@colorado.edu

(CSCI 5502)

Dheeraj Chinni Ranga

ID: 104362724

University of Colorado, Boulder

rach9569@colorado.edu

(CSCI 5502)

Uppendra Sabnis

ID: 104379965

University of Colorado, Boulder

upsa3702@colorado.edu

(CSCI 5502)

1. INTRODUCTION

A recommendation system allows people to make a choice based on the patterns observed in general. These patterns can be based on history or current trending data. Such a choice made through recommendation has a high probability of being liked by the user. Advertisements in general help the product to reach out to masses. But in the long run, recommendations make a difference by targeting the advertisement to more relevant sources. For any business, most of the major decisions are based on the acceptance of its product or service by people.

One of the major use cases of recommendation systems is to help the sellers to have a deeper analysis of the history of customer's interactions with the products. This will help them to understand the kind of individuals in a particular region and cater to their specific needs. Sometimes, the customers are apprehensive to try out things that are relatively new in the market. In case of personal items, they tend to stick to their routine choices. Whereas in case of decisions on songs, they tend to go with widely accepted choice. Thus making an accurate and precise recommendation system is a merit to both producers and customers. Sellers get to increase their sales and reputation by suggesting choices which will be liked by the customers.

2. MOTIVATION

Almost all the fields have some kind of recommendation system in place today. Be it about buying a soap or visiting a restaurant in the city. Music is one such area which is most widely favourite. Many people listen to music from various artists of different genres. In normal scenario, individual has to listen to a particular song to decide if he liked it or ask his acquaintances with similar preferences for suggestions. We can definitely improve this experience, if we can predict the songs which he may like in advance. Thus there is a demanding necessity for an accurate and efficient songs recommendation system. Songs recommendation systems are increasing in number. Most of these systems ask users to create a profile and rate a few songs. Then the system would find similar songs based on the ratings by other users with similar preferences or the inherent qualities of the songs.

A system which has the ability to give best suggestions to the users without allowing them to make any conscious effort explicitly, will prove to be highly useful. We aim to build such a system in this project. Once it is developed, implementing it in various fields is a trivial task. It could be used with a multitude of other areas by just tweaking the system accordingly.

There are billions of diverse songs, which cater to wide variety of music lovers. Pandora, Saavn and Spotify are the most widely used applications these days. They are known for recommending the most accurately likely songs to users.

Depending on the approach that they follow, these applications have their own drawbacks. We propose to design a customized system for users, which would consider various factors such as user's interests as well as the songs popularity. We think users generally do not stick to the same genre and can have a liking towards two or more genres that are similar in some ways. This way, the recommendations would be based on various aspects of the song that the user likes and the songs which are trending.

3. LITERATURE SURVEY

As mentioned earlier, recommender systems typically produce a list of recommendations in one of two ways - through collaborative or content-based filtering. Both have its pros and cons.

3.1 Collaborative filtering

The goal of a collaborative filtering algorithm is to suggest new items or to predict the utility of a certain item for a particular user based on the user's previous likings and the opinions of other like-minded users. In a typical CF scenario, there is a list of m users $U = \{u_1, u_2, \dots, u_m\}$ and a list of n items $I = \{i_1, i_2, \dots, i_n\}$. Each user u_i has a list of items I_{u_i} , which the user has expressed his/her opinions about. Opinions can be explicitly given by the user as a rating score, generally within a certain numerical scale, or can be implicitly derived from purchase records, by analyzing timing logs, by mining web hyperlinks and so on. Note that $I_{u_i} \subseteq I$ and it is possible for I_{u_i} to be a null-set. There exists a distinguished user $u_a \in U$ called the active user for whom the task of a collaborative filtering algorithm is to find an item likeliness that can be of two forms.

Prediction is a numerical value, $P_{a,j}$, expressing the predicted likeliness of item $i_j \in I_{u_a}$ for the active user u_a . This predicted value is within the same scale (e.g., from 1 to 5) as the opinion values provided by u_a .

Recommendation is a list of N items, $I_r \subset I$, that the active user will like the most. Note that the recommended list must be on items not already purchased by the active user, i.e., $I_r \cap I_{u_a} = \emptyset$. This interface of CF algorithms is also known as Top-N recommendation.

Memory-based Collaborative Filtering Algorithms:

Memory-based algorithms utilize the entire user-item data-base to generate a prediction. These systems employ statistical techniques to find a set of users, known as neighbors, that have a history of agreeing with the target user (i.e., they either rate different items similarly or they tend to buy similar set of items).

Model-based Collaborative Filtering Algorithms: Model-based collaborative filtering algorithms provide item recommendation by first developing a model of user ratings. Algorithms in this category take a probabilistic approach and envision the collaborative filtering process as computing the expected value of a user prediction, given his/her ratings on other items.

Challenges of User-based Collaborative Filtering Algorithms:

User-based collaborative filtering systems have been very successful in past, but their widespread use has revealed some potential challenges such as:

Sparsity: In practice, many commercial recommender systems are used to evaluate large item sets (e.g., Amazon.com recommends books and CDnow.com recommends music albums). In these systems, even active users may have purchased well under 1% of the items (1% of 2 million books is 20,000 books). Accordingly, a recommender system based on nearest neighbor algorithms may be unable to make any item recommendations for a particular user. As a result the accuracy of recommendations may be poor.

Scalability: Nearest neighbor algorithms require computation that grows with both the number of users and the number of items. With millions of users and items, a typical web-based recommender system running existing algorithms will suffer serious scalability problems.

3.2 Content Based filtering

Systems implementing a content-based recommendation approach analyze a set of documents and/or descriptions of items previously rated by a user, and build a model or profile of user interests based on the features of the objects rated by that user. The profile is a structured representation of user interests, adopted to recommend new interesting items. The recommendation process basically consists in matching up the attributes of the user profile against the attributes of a content object. The result is a relevance judgment that represents the user's level of interest in that object.

Advantages:

- **USER INDEPENDENCE** - Content-based recommenders exploit solely ratings provided by the active user to build her own profile. We can add item frequency, ratings etc as well.
- **TRANSPARENCY** - Explanations on how the recommender system works can be provided by explicitly listing content features or descriptions that caused an item to occur in the list of recommendations. Those features are indicators to consult in order to decide whether to trust a recommendation.
- **NEW ITEM** - Content-based recommenders are capable of recommending items not yet rated by any user. As a consequence, they do not suffer from the first-rater problem, which affects collaborative recommenders which rely solely on users' preferences to make recommendations. Therefore, until the new item is rated by a substantial number of users, the system would not be able to recommend it.

Disadvantages:

- **LIMITED CONTENT ANALYSIS** - Content-based techniques have a natural limit in the number and type of features that are associated, whether automatically or manually, with the objects they recommend. Domain knowledge is often needed. Some representations capture only

certain aspects of the content, but there are many others that would influence a user's experience.

- **OVER-SPECIALIZATION** - Content-based recommenders have no inherent method for finding something unexpected.
- The system suggests items whose scores are high when matched against the user profile, hence the user is going to be recommended items similar to those already rated.
- **NEW USER** - Enough ratings have to be collected before a content-based recommender system can really understand user preferences and provide accurate recommendations. Therefore, when few ratings are available, as for a new user, the system will not be able to provide reliable recommendations.

3.3 Pandora

Pandora's recommendations are based on the inherent qualities (Content based filtering) of the music. Give Pandora an artist or song, and it will find similar music in terms of melody, harmony, lyrics, orchestration, vocal character and so on. Pandora likes to call these musical attributes "genes" and its database of songs, classified against hundreds of such attributes, the "Music Genome Project."

Pandora is less subject to the echo chamber of overly like minds, but it has its own fundamental challenge in its reliance on matching songs' "genes." This rules out connections between songs or artists that don't fit Pandora's modeling and matching of musical qualities—which, in turn, puts enormous pressure on Pandora's specific approach to be correct. In other words, Pandora's success hinges on a theory, and a specific implementation of that theory, about why music recommendations work.

3.4 Last.fm

Last.fm music-discovery systems have been social recommenders, also known as collaborative filters. Although much of the academic work in the area has focused on improving the matching algorithms, Last.fm's innovation has been in improving the data the algorithms work on. Last.fm does so by providing users an optional plug-in that automatically monitors your media-player software so whatever you listen to whether it came from Last.fm or not can be incorporated into your Last.fm profile and thus be used as the basis for recommendations. Compared to relying on users to manually provide preferences, this automatic and comprehensive data capture leads to far better grist for the data mill. Last.fm simply describes what goes together according to its audience and then makes relatively simple inferences from that. So if there are hidden factors that Pandora isn't explicitly capturing, Last.fm is at least capturing them indirectly.

The difference between Pandora and Last.fm can be seen with the simple scenarios.

- Assume that both services put new artists into their database at the same rate, Last.fm will be slower in surfacing them as recommendations. This is due to the “cold start” problem that afflicts social recommenders: Before something new can become recommendable, it needs time to accumulate enough popularity to rise above the system’s noise level. In contrast, because Pandora is only comparing songs’ inherent qualities—not who they’re popular with—it should be able to recommend a new artist the first day that artist is in the system.
- A major problem with Pandora is “locked loop,” whereby a content based recommender gets stuck in certain genres and styles. But with a collaborative recommender, a truly locked loop is unlikely. The reason is “leakage”: A population that shares the same core musical tastes will have enough variance in secondary tastes to allow for a continually expanding spectrum, albeit with much slower expansion in certain genres than others.

4. PROPOSED WORK

We propose to design a system that bases its recommendations on both content based and collaborative filtering. This can be achieved based on some important factors like the Artists, Loudness, Dance ability, Tempo, Energy, Song hotness. The values for these attributes are extracted from the songs dataset.

We are planning to map 2 datasets as follows:

Song dataset: This data set contains the information about number of songs, along with the list of 50 odd attributes associated with each track.

User dataset: This dataset contains detailed information of all the users with attributes like Artist Name, Song Title, and Timestamp.

Our first step is to cluster the songs. We can find relations between songs, based on content. Content is their metadata information given in Song dataset.

Second step is to cluster the users using both datasets. Each user cluster contains users with similar preferences based on the frequent genres.

When we want to recommend user some songs, we will use both the song clusters and user clusters.

A user has profile based on the frequencies of songs heard by him. We can match it with the closest song cluster and recommend the songs from that cluster. We can also recommend the songs from profiles of other users from the cluster the user belongs to.

5. DESIGN AND IMPLEMENTATION

Final goal of the system is to recommend a user with songs that will match his interest. Accuracy of this recommendation depends on the cluster formation. Clustering will be good if we are able to find better correlation values. Hence each process from data collection to song recommendation is crucial.

5.1 Initial setup

We started implementation of the project with data collection. During data collection phase we realized that we need better resources to handle large dataset. Hence we decided to host our project on cloud. Moreover the dataset that we are using is available as one of the volumes on EC2.

Our initial datasets were in variety of forms. Songs data is in HDF5 while jams data was in TSV. Reading from multiple sources is time consuming. Hence we copied all our data in nosql database mongodb on cloud. Instead of reading a particular record from file each time, more memory in cloud enabled us to load more data in the memory. This transformation indeed helped us in improving performance.

5.2 Data collection

Songs data was available on one of the EC2 volumes. We attached this volume to our server, read data and stored it in mongodb. Jams data was downloaded separately in TSV format and then stored in mongodb. Required wrappers were written.

5.3 Understanding data

The website from which downloaded data had ample amount of information. We understood each and every attribute in detail. Blogs on the same site covers the experiences of people who have worked on it before. It was useful in deciding which attributes will be useful. Considering both datasets, it has sufficient information to map jams and songs. But it lacks information about individual users. Hence artist_terms and song_title was considered for mapping

5.4 Data Overview

As discussed earlier, we used two datasets in our project.

Songs dataset: Original data in HDF5 form had about 50 attributes for each song like song id, song title, artist id, artist name, artist location, artist mbtags, song loudness, song energy, song tempo, song release year, danceability, similar artists, artist terms. We had to remove unnecessary attributes. Mainly three types of values: string, numeric and categorical. Attributes used for the project are,

song_id: uniquely identify the song (ex: SOCWJDB12A58A776AF)

song_title: title of the song (ex: Never Gonna Give You Up)

artist_name: name of the artist (ex: Rick Astley)

artist_terms: categories of different genres that particular song belongs to (ex: [“power metal”, “progressive metal”, “heavy metal”])

artist_mbtags: collection of various tags corresponding to particular song has. These tags were general compared to artist_terms. (ex: [“Denver”, “American”])

bars_confidence: confidence measure of bars throughout each song. Length of this attribute depends on the duration of the song. (ex: [0.169, 0.319, 0.223, 0.327,])

beats_confidence: confidence measure of beats throughout each song (ex: [0.548, 0, 0, 0, 0.071, 0.217, 0.121,])

artist_familiarity: algorithmic estimation based on number of songs listened by users (ex: 0.7804617487770407)

artist_hottness: algorithmic estimation based on number of songs listened and tagged by users (ex: 0.5742747305168561.)

similar_artist: artist with same genres (ex: ["AR2O6M21187FB39A28", "AR5FUEA11C8A4152CE"])

release: name of the album in which particular song was released (ex: "Call of the Mastodon")

duration: duration of song in minutes and seconds (ex: 4:25)

song_hottness: algorithmic estimation based on tags by users (ex: 0.5976407977147769)

danceability: algorithmic estimation of bars confidence, beats confidence, tempo and energy

energy: energy of the song from listener point of view

loudness: overall loudness in dB (ex: 0.8516730831848915)

tempo: estimated tempo in BPM (ex: 173.205)

User dataset: Original dataset in TSV format. It had attributes like jam_id, user_id, artist_name, song_title, creation_date, link, spotify_uri. We had to remove unnecessary attributes. This dataset has all string values. Attributes used for this project are,

jam_id: uniquely identifies jam_id (ex: "c2e76bb92c7fa733fd9c9be40bb0e4ea")

user_id: id representing user whose jam it is (ex: b99ebf68a8d93f024e56c65e2f949b57)

artist_name: name of the artist to which this jam belongs to (ex: "Orange Juice")

song_title: title of the song (ex: "Rip It Up")

creation_date: date on which this jam was created by corresponding user (ex: "2011-08-26")

Mapping between 2 datasets: To find out relationship between users, song and artist we have mapped two datasets. Mapping attributes are artist_name and song_title.

5.5 Data cleaning

5.5.1 Deciding attributes: In the complete database of songs there are few attributes which are not useful while finding out recommendations. Hence depending on domain knowledge and blogs we short listed few attributes which will contribute in final recommendations.

5.5.2 Missing values: More than half of the attributes in datasets are numeric. Moreover these values are fairly distributed. Hence we decided to replace the missing values with global mean of the respective attribute. While loading data from HDF5 we replaced the missing values with the largest possible float value. Then calculated mean for each attribute without considering these float values. In the next run replaced all the large float value with global mean. We are not using any string value in prediction. Hence there was need to replace missing string values with anything.

5.5.3 Normalization: All the values corresponding to float attributes belonged to separate ranges. To make them fall under one range we used data transformation technique. We used simple min max normalization method to convert all values to fall into range of -1 to 1.

5.5.4 Apriori algorithm

Each song is associated with set of various genres. Thus from available datasets we mapped the genres associated with each user in jams. There were vast number of genres hence we used apriori algorithm to find out frequent genres. So from around 3000 genres, we were able to reduce it to around 60 important genres. We tested data multiple times with apriori to find out minimum support we can use. If we increase the support beyond 10%, then we were missing on important genres like classical, rock etc. Hence we decided to use 10% support.

5.6 Collaborative filtering

Collaborative filtering is a prediction technique that uses user to user relation. We have user data of almost 2 Million records with each user playlist history, data is in the format of userId, song title, Artist name and few other attributes that are irrelevant to our project. The main challenge is to map 0.2 Million songs with 2 Million users and selecting an attribute that defines the relation between users. To achieve this we selected song genre as defining factor between two users.

Our data contains many genre for each song so we have applied apriori algorithm on these genres and selected top 67 genre forming an important attribute to differentiate user. First step, has involved with getting the user genre count for each user by mapping the songs he listen to the 0.2 Million songs data. This will give us a set of values for each user with the count for each of 67 attributes of all the songs he listen to. There is a chance that user may listen to multiple songs this will give us an advantage of understanding more about user taste. In our case we have almost 70k users with matching songs. These intermediate results are stored in mongoDB, which reduced the work of running multiple times.

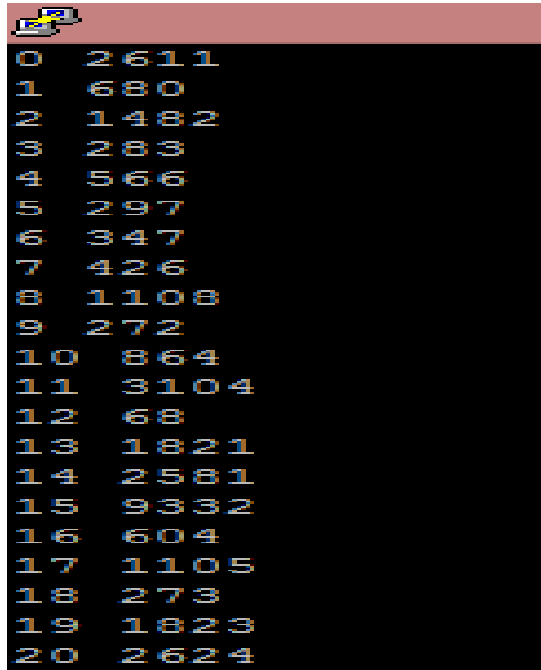
Now with the user's data of genres the main task is to relate the users and form clusters. To achieve this we used to correlation to find the distance between each user. As we have set of points that describes the user interest the best suitable correlation technique for this is correlation coefficient as it calculates the mean of the points, standard deviation and applies a formula that checks for each of the point in the set.

Correlation coefficient formula

$$r_{A,B} = \frac{\sum_{i=1}^N (a_i - \bar{A})(b_i - \bar{B})}{N \sigma_A \sigma_B}$$

For k-means we have started with 500 random points from 70k users and considered each point as centroid of the cluster. Now, we have calculated correlation between each user and these 500 centroids, the centroid with maximum correlation value is considered as the user's cluster. K-Means is used here because all we have is set of random points from which we can find the centroid and calculate the distance between the centroid and the user, which will be helpful to understand how close the user to this point is. This helped us to group all the users with the same

interest, these clusters are helpful to understand what kind of songs a particular user group listens to.



0	2611
1	680
2	1482
3	283
4	566
5	297
6	347
7	426
8	1108
9	272
10	864
11	3104
12	68
13	1821
14	2581
15	9332
16	604
17	1105
18	273
19	1823
20	2624

With the clusters in hand the next step is to find out the top songs in each of these clusters that users listen. According to our model we are recommending these top songs to an user apart of the songs that are matching with his play list. These tops songs are calculated based on number of times the song has been listened by different user, with a minimum count of 5 times for each songs. The accuracy for this model is defined as number of songs that matches the top songs to total number of songs the user listens.

5.7 Content Based Filtering

Content based filtering is a prediction technique that tries to understand individual user's preferences. From user's history, the filtering system tries to build a prediction model. Hence it is supervised learning algorithm. Accuracy is measured by checking if user preferences match with the data. We have user data of almost 2 Million records with each user playlist history. A playlist has a Jam_Id. One song can be heard by user multiple times, hence it will have multiple Jam_Ids. But we if song Artist Name and Title Name are same in 2 jams, then the song has same features. A song has features like artist_familiarity, artist_hottnesss, danceability, energy, loudness etc. These are numeric values and are normalized in data preprocessing. There are few other attributes that are irrelevant to our project.

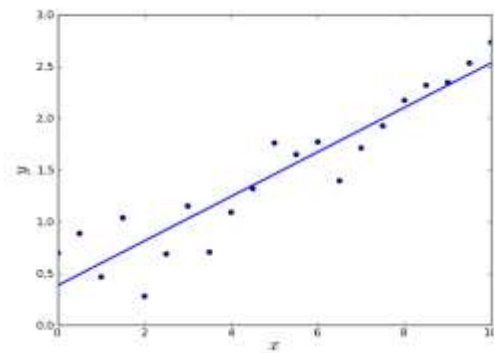
Content based filtering needs a preference system. There are users and items. Items have features which define them and users have ratings for individual item. From a user's history, we have to create a rating system. From the rating information for a song, we have to create a preference vector for individual user. This preference vector can be used in future for rating songs not heard by the user.. Hence, first step involved with getting the user to songs mapping for 0.1 million users and 0.2 million songs. Ratings is the number of time a user listen to a song

which is the item in our system. In our case we have almost 70k users with matching songs. These intermediate results are stored in mongoDB, which reduced the work of running multiple times.

We decided to create a linear regression model from user to song matrix. In linear regression, we try to find best model in form of linear affine function of following form: $Y = A \cdot X + B$, where X is feature vector.



A is user preference vector and B is regularization parameter. Y is the predicted rating for the song. Y' is the real rating. Hence, error in prediction is $(Y' - Y)$ Eg. In following example, X is a unidimensional feature vector and Y is prediction.



Our preference vector for individual users are all set as $[1,1,1,1,...1]$ belonging to R^n . Minimizing the Root Mean Square error is a convex optimization problem. We can use gradient descent method for minimizing.

Problem Formulation is like following:

$r(i,j) = 1$ if user j has rated song i (0, otherwise).

$y(i,j)$ = rating by user j on movie i (if defined)

Θ_j = Preference vector for user j .

x_i = Feature vector for song i

For user j , predicted rating for song $i = \Theta_j \cdot x_i$

M_j = no of movies rated by user.

λ = regularization parameter

Minimize $\frac{1}{2} \sum_{j=1}^M M_j \cdot (\sum_{i=1}^n x_i \cdot \Theta_j - y(i,j))^2 + \lambda$

After around 10 iterations, running gradient descent on this minimization, our solution converges. By this point, we have found out our preference vector for individual user.

For every $r(i,j) = 0$, where a user i has not no rating for song j , we predict his rating as $X^T \cdot y(i,j)$.



6. HOW TO EVALUATE

6.1 Cluster Analysis

First we have started with cluster evaluation to understand if the formed clusters are correct. For this we have calculated inter-cluster and intra-cluster correlation values. In Inter-cluster we have randomly choose users from two different clusters and calculated correlation values if this correlation value is less the clusters are not similar. The same process is repeated for Intra-cluster but the users are randomly choose from the same cluster and calculated the correlation value, if the value comes out maximum then the users are strongly correlated and they are much similar. For our clusters we have seen values in negative for Inter-clusters correlation and values as high as 1.0 for Intra cluster correlation. This shows that give the attributes our clusters are strongly not correlated.

```
Intracluster_similarity 0.177020069476
Intercluster_similarity -0.137003844749
user@cu-cs-vm:~/Documents/datamining/pro
Intracluster_similarity 0.305350400009
Intercluster_similarity 0.195830231241
user@cu-cs-vm:~/Documents/datamining/pro
Intracluster_similarity 0.435551269267
Intercluster_similarity 0.191236866699
```

6.2 Collaborative filtering

For testing we have divided the total users into 80% and 20% for training data and testing data. Based on 80% of the training data our model recommends the songs to other 20% of the users based on which cluster they belongs to. We are suggesting the top songs in the cluster because we strongly believe if the users of same interest are listening to some set of songs then these songs can be liked by the testing user. Our testing model uses the coverage method where accuracy is calculated as number of covered values to the total number of values in the area.

The testing starts with randomly picking up an user from the 20% users data and calculating his correlation value with all the available clusters and select the cluster with maximum correlation value. Once the cluster is determined we collect all the user songs from jam database that matches 0.2 Million song database. The top songs for the cluster are calculated as follows

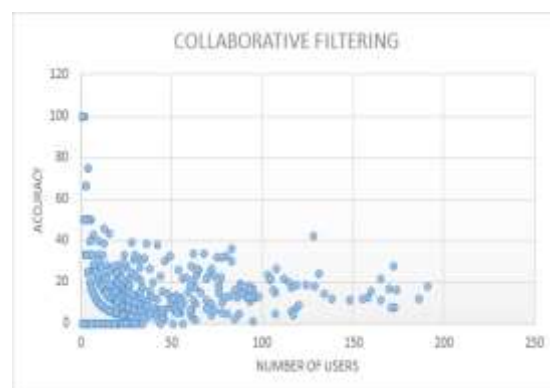
1. We collect all the songs that the users in the cluster listen to from Jam database.
2. These user songs are matched with 0.2 Million songs data and everytime we see the same song it's count value is incremented.
3. Now we have songs with their count there are songs with zero count as we are considering only 20% of the total songs songs, in this step we eliminate these songs followed by ordering the songs based on the count value.
4. In the final step the testing user songs are mapped with all of these top songs and get the count of total matched songs.

5. Based on the number of songs matched to the total songs we calculate the accuracy as,

Number of songs matched/Total number of user songs

With this testing model we are getting an average accuracy of 30% with a maximum accuracy of 100% and minimum accuracy of 0%. The testing is considered with different number of clusters first we start with 500 clusters followed with 600, 700 and even 1000 clusters. The accuracy levels increases as the clusters are more defined to particular genre of users.

The reasons sometimes the accuracy drops to 0% are the limitation of the song genre's as explained before we are applying 10% support for Apriori to decide number of genres to be considered for the user classification. If the minimum support is reduced further then we may have enough genres that can lead to get better correlation value for clusters. Another problem can be the user song selection is very sparse and the genres might not be matching with our set of genres we are considering and as we are considering only 20% of the total songs the accuracy can drastically increase in case if we run our model on the entire data. 100% accuracy defines that the users are strongly correlated to the cluster and all their songs are matching with the top songs.



The above figure shows the accuracy value for 1000 users with number of songs on the x-axis and accuracy on the y-axis. This graphs explains that users with less songs has more accuracy and users with more songs has less accuracy the reason for this are as we are considering only 20% of the songs data set if we consider full dataset then there is more chance to get more top songs for each cluster that can match with the user songs. Another reason can be the number of genre's if this value increases then we can expect more close relation between users.

6.3 Content based filtering

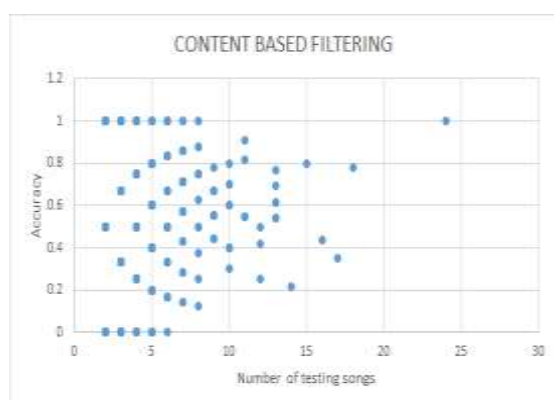
In this problem, our prediction is based on user's history. So, if user has no history, we can not create prediction model for him. While calculating user preferences, we found out that, out of 70,000 users, approximately 10,000 users heard more than 5 different songs. For these 10,000 users, we divided their songs ratings in 2 parts, 80% for training data and 20% for testing data. For usability, we stored them in MongoDB. While running the prediction model, we used training data and built prediction

parameters for all 10,000 users. It looks like this. While testing the model, we used predicted rating and matched it with real rating from testing data. As there are no negative ratings, we used sensitivity as our accuracy.

TPR (True Positive Rate) = True Positive/Positive.

With this setting, we calculated mean TPR for 10,000 users. It came around **40%**.

Our Root Mean Square error for all user was around 1.9. We tested with multiple Theta and Lambda values and reduced it to around 0.9 and our mean TPR shot up to **55%**.



The above figure shows the accuracy value for 10,000 users with number of songs on the x-axis and accuracy on the y-axis. In this graph, number of songs are the songs from testing alone. Hence, the maximum songs we saw is 24. For this user, there are 120 songs but 80% of it, that is 96 songs are used for training set. This graphs explains that on average, if a user has more songs in its history, chances are the accuracy of his predictions is better. In this setup, we are only using 0.2 million songs out of 1 million songs but based on the trend that a user with higher song history has better accuracy, we think our accuracy will be very high with 1 million songs.

7. CONCLUSION

We implemented the proposed system successfully. We built an efficient songs recommendation system with that takes into consideration the unique interests of the user. The system is built on the foundation that each user has varied interests and cannot be compared to other users.

8. FUTURE ENHANCEMENTS

1. Extend it to hybrid recommendation system. Currently the collaborative and content based filtering is independent of each other. We can make use of user cluster and song prediction model combined to recommend the song more accurately.
2. Increase the accuracy. Current datasets lack the data about users. We would like to explore data from songs to find better correlation between users.
3. Use more data mining techniques: We used simple data mining techniques at each stage of the project. We want to test using different techniques to see if they give any better results. For example, instead of

k-means for clustering we can use some hierarchical clustering method.

4. We have used linear regression which may not always yield best results. We would like to explore higher order regressions as well as artificial neural network for content based prediction

9. CONTRIBUTIONS

No	Tasks	Contributors
1	Collection of data	Upendra, Dheeraj, Nachiket
2	Environment setup	Upendra
3	Evaluation of data	Upendra, Dheeraj, Nachiket
4	System design	Upendra, Dheeraj, Nachiket
5	Missing attributes	Upendra
6	Normalization	Dheeraj
7	Apriori	Nachiket
8	Collaborative filtering	Dheeraj
9	Content based filtering	Nachiket
10	Testing	Upendra, Dheeraj, Nachiket
11	Presentation	Upendra, Dheeraj, Nachiket
12	Report	Upendra, Dheeraj, Nachiket

10. REFERENCES

- [1] Pasquale Lops, Marco de Gemmis, Giovanni Semeraro, "Content-based Recommender Systems: State of the Art and Trends" (2007).

- [2] Aaron van den Oord, Sander Dieleman, Benjamin Schrauwen, "*Deep content-based music recommendation*".
- [3] J. Ben Schafer, "*The Application of Data-Mining to Recommender Systems*".
- [4] Wikipedia,
https://en.wikipedia.org/wiki/Recommender_system
- [5] <http://blog.stevakrause.org/2006/01/pandora-and-lastfm-nature-vs-nurture-in.html>
- [6] *Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The Million Song Dataset. In Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011), 2011.*
- [7] *[7]Andreas Jansson, Colin Raffel, and Tillman Weyde. "This is my Jam -- Data Dump", in 16th International Society for Music Information Retrieval Conference Late Breaking and Demo Papers, 2015.*
- [8] <https://www.youtube.com/watch?v=UvCgCLC9pCc>