CS221 Project Final Report
Community Detection based on Music Listening Habits

Agrim Gupta, Karan Rai, Nehal Bhandari

1 Introduction

Music plays a big role in our lives and forms a billion dollar industry. A lot of research goes into music recommendation and prediction especially with so many internet radio services coming up like Spotify, Pandora etc. Even though community detection, in general, is a well tackled problem in social and information flow networks, an interesting and fairly unsolved problem is that of identifying communities within the music listening population. We want to tackle this problem as it can provide various insights into listening habits of people. As this is a unsupervised learning problem, we also build a recommendation system to suggest songs to our users based on the clustering. This helps us in evaluating our clustering model.

2 Task Definition

Given song listening history for several users along with song meta-data, we want to cluster users according to their tastes in various genres of music as well as song characteristics. A particular user can belong to multiple such clusters. Making use of these communities, we would then recommend songs to suit user's tastes based on what other users listen to in his/her cluster. The input data for our system will be user listening history i.e. user-song-playcounts as well as song characteristics. The output of our clustering algorithm will be cluster assignments for each of the users. Using this output, we then apply our recommendation system which outputs a set of suggested songs for each user.

Since this is unsupervised learning, we do not have any direct evaluation metrics like accuracy, precision, recall etc. Hence, we define our own evaluation metrics. We evaluate our recommendation based on how many songs, artists and genres we are correctly able to predict with respect to the user's already listened songs. Further, we also claim that our algorithm is much more scalable than traditional recommendation systems like collaborative filtering. So, we also evaluate running time of our recommendation system.

3 Related Work

There has been an increased interest in the discovery and analysis of community structure in networks [1] in recent years. [2] discusses methods like spectral graph partitioning, hierarchical clustering etc used to form communities in networks such that every vertex belongs to one community. In contrast to traditional partition clustering, overlapping clustering [3] allows items to belong to multiple clusters.

Besides Community Detection, a lot has been done in the field of Recommender Systems (RS). Collaborative Filtering [6] methods are the most commonly used techniques in recommendation systems which rely on collection and analysis of user's preferences and predicting what users will like based on their similarity to other users. Many algorithms have been used in measuring user similarity or item similarity in recommender systems. For example, the k-nearest neighbor (k-NN) approach [7] and the Pearson Correlation as first implemented by [8]

We aim to identify overlapping communities within music listening populations and leverage this information to get insights on what are the characteristic features of songs in each of these different communities. A Recommendation System on top of these results will help us evaluate how relevant these communities are.

4 Infrastructure

4.1 Dataset

We started off with the Last.FM data set that provides user listening history along with Last.FM IDs of songs that can be used to gather meta-data for different songs. But due to site redesigns, new developer API keys were not available at the time of data collection. Finally, we decided to go with The Million Song Dataset (MSD) [5] which is publicly available from the joint collaboration between LabROSA and the Echo Nest [4]. The Echo Nest Taste Profile Subset in MSD contains (User ID, Song ID, Playcount) tuples.

For clustering purposes, we further queried additional features for each individual song via the Echo Nest API, which is freely available. Due to access limit of 20 API calls per minute (for non commercial users) we decided to limit our dataset to 15000 users. One limitation in the API is unavailability of genres for each song. However, as an approximation, we attribute genres for artist of a particular song to song genres. For every unique song we scraped the ArtistID and 10 other relevant song parameters: key, energy, liveness, tempo, speechiness, acousticness, instrumentalness, loudness, valence and danceability, provided by the EchoNest API. Subsequently, we used these ArtistIDs to scrape genre information. In the end we have a dataset with 60123 unique songs and 14881 unique users (because for some users, none of the songs had artistID information available from the API).

4.2 Features

Since the number of distinct genres obtained was large (1211), we used 100 most popular genres based on frequency of occurrence. Also, genres that occur less frequently won't help in our clustering, in fact, they might worsen it by adding noise. Genre feature template is 'ith genre is present' which basically results in a 100 dimensional vector, such that if i^{th} genre is present in the song, the i^{th} row is set to 1. For each of other song parameters, values are uniformly divided to 10 bins (12 for key, where values are already discrete), and a feature vector of length 10 with one-hot values is used. For example: if energy has value 0.66, it lies in 7th bin and 7th row is set to 1 and rest to 0. This is repeated for all the song parameters to give a 102 dimentional feature vector. Combining these with genre features, we have a total of 202 features.

Of the 10 song parameters (102 features), we wanted to see if there is enough variability in the values of these parameters across all songs. Figure 1. depicts this variation. We see that for features like liveness, tempo, speechiness, acousticness, instrumentalness and loudness, the variation is not that significant and for most songs, these features take on similar values.

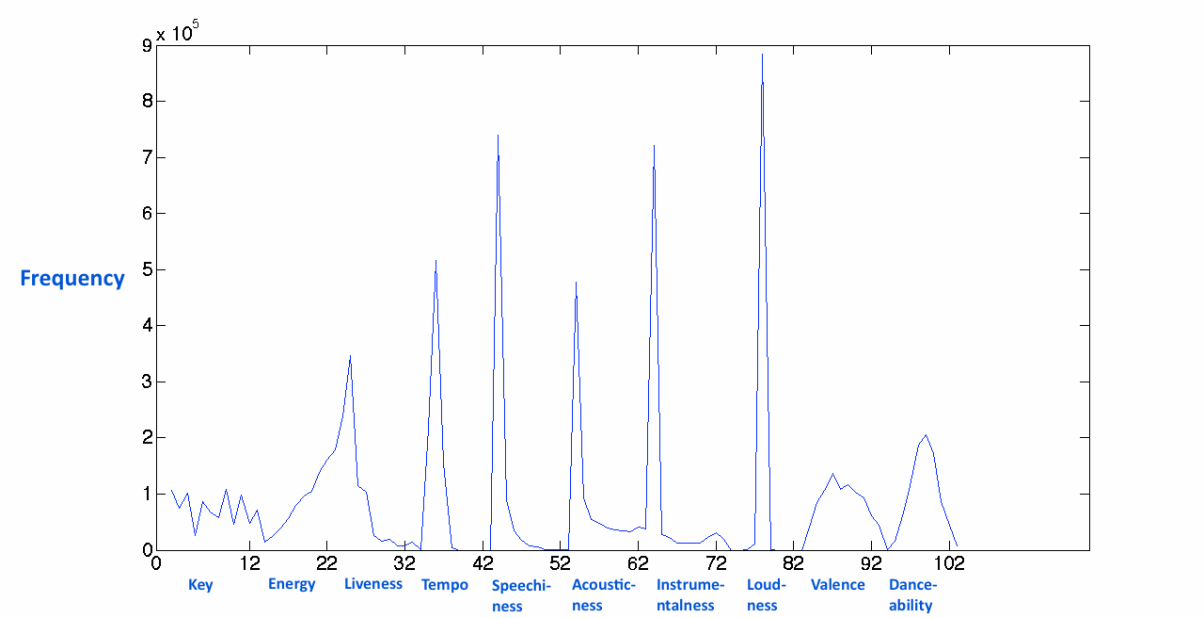

Figure 1: Song Feature Bin Frequency

Therefore, in the final execution and evaluation we only consider genre, key, energy, valence and danceability as features, giving a total of 142 features (including genre features). Once we have this 142 length feature vector for all distinct songs, we compute user features by:

$$Feature(UserX) = \sum_{S \in Songs(UserX)} PlayCount(UserX, S) * Feature(S)$$

5 Methodology

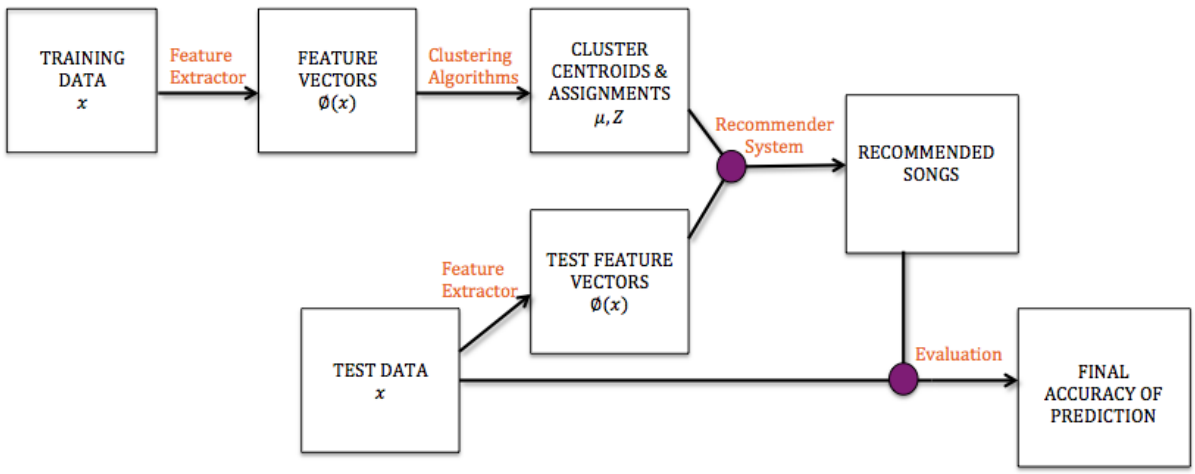

Figure 2: Work Flow

The flow chart for our system is shown in Figure 2. We used SKLearn library for implementing some of the clustering algorithms. [14]

5.1 K-Means

With feature vectors as described above, we ran K-Means algorithm for varying cluster counts. We observed formation of one large cluster always, which would contain majority (80-90%) of the users because with huge play counts (leading to large coordinate values), they do not get assigned to other meaningful clusters. To overcome this we normalized the feature vector for each user, after which we get a reasonable distribution of users across clusters.

Determining the number of clusters in a data set, is a frequent problem in data clustering. We used Silhouette coefficients to determine the value of k. Silhouette coefficients near +1 indicate that the sample is far away from the neighboring clusters. A value of 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters and negative values indicate that those samples might have been assigned to the wrong cluster. Plot on the left side of Figure 3has the silhouette values for every user. The dotted line represents the average value of the silhouette coefficient. From the graph on the right side in Figure 3 we can see that for $k = 18$ the average value of k is among the highest. Ideally, we would want that the silhouette values are positive. But we can observe from the graph on the right that for many users this is not the case. We attribute this to the susceptibility of k-means to outliers. Also, we looked at the top features (specifically genres) for each cluster and noted that the features clustered together made sense. We can see in Table 1 a sample of 4 clusters for $k = 18$. We observe that similar genres are clustered together.

0	1	2	3
groove metal	pop	r&b	gangster rap
post-grunge	pop rock	urban contemporary	alternative hip hop
rap rock	indie rock	pop christmas	pop rap
Energy 9	Energy 9	Energy 9	Energy 7
Valance 3	Valance 7	Valance 3	Valance 3

Table 1: Top Features for k = 18 for 4 clusters

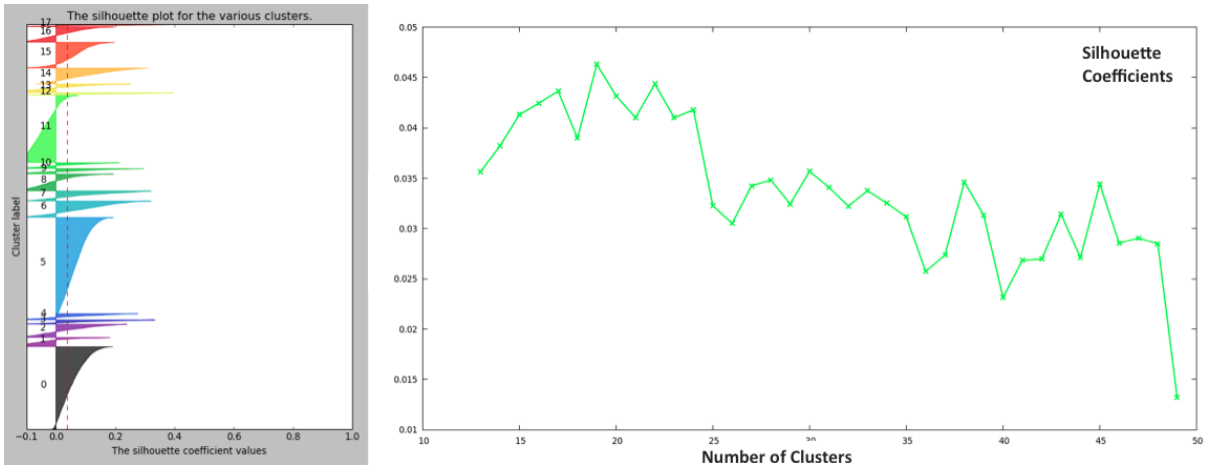

Figure 3: K-means Clustering Analysis

5.2 Gaussian Mixture Models

The main drawback of K-Means algorithm is that one user can only belong to one cluster. But in reality, one user could potentially listen to many different types of songs that might not sit well in the same cluster. Hence, we want to classify single users into multiple clusters. GMM allows for the possibility to do this. The basic difference in GMM algorithm is that instead of assigning each user to a fixed cluster we maintain probabilities of a user belonging to each cluster, based on gaussian distribution parameters. According to literature, number of clusters for GMM is often initialized with K of K-Means clustering. Since we obtained some success with $k = 15$ in K-Means, so we started with that assumption here as well.

GMM is an EM algorithm where we assume that the data points are generated from K multivariate Gaussians. We initialize the Gaussian distribution parameters (mean, covariance) randomly. Then, in the expectation step, we find probabilities of each data point belong to each of the K clusters. In the maximization step, we use these probabilities to re-estimate the means and covariances for the Gaussian models. We continue doing these 2 steps until the Gaussian distributions converge. We use the same feature vector for GMM as we had in K-Means. We also plotted the Bayesian Information Criterion for various values of number of clusters (shown below). From this we infer the optimal cluster count of $k = 9$ for 1000 users. But we tested on various other values of k as well for our evaluation because good clustering doesn't always imply good recommendation.

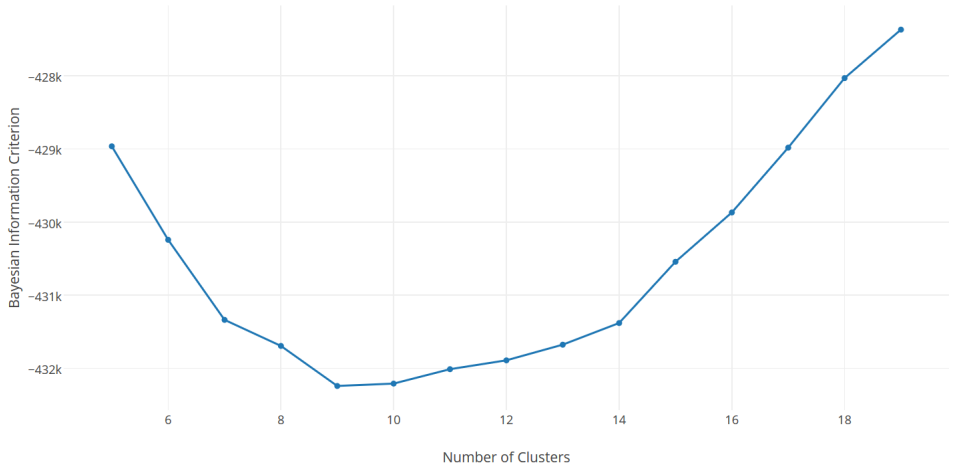

Figure 4: Bayesian Information Criteria

5.3 BIRCH: Balanced Iterative Reducing and Clustering using Hierarchies

BIRCH is an unsupervised learning algorithm used to perform hierarchical clustering [13]. K-means works best when the clusters are linearly separable (by hyperplanes) and the clusters are compact. Whereas, hierarchical clustering methods are best suited for applications where clusters are more spread out and have a tree like structure. For eg. let's observe the top genres in cluster 1 in Table 1. Now, users belonging to this cluster may prefer pop rock or indie rock in particular and pop in general. It is also likely that since songs have multiple genres, the users clustered into this cluster might like specifically pop rock but because the song also belonged

to pop category they were clustered with users who don't have such specific tastes. To capture this behavior we felt that hierarchical clustering based methods like BIRCH would be better suited for our task.

BIRCH algorithm maintains Clustering Feature Trees (CF Trees). Clustering Feature is a triple containing information about a cluster, specifically, number of data points, linear sum and squared sum of points. A CF Tree is simply a height balanced tree with branching factor B and threshold T. Threshold represents the maximum allowed diameter of a cluster i.e. how big a cluster is (technically, it will be the distance between farthest points of the cluster). Threshold tells us when to branch and when we can make do with the current tree node itself. Leaf nodes of this tree represent our final cluster which themselves will be made up of several subclusters because of the hierarchy.

BIRCH algorithm is an incremental clustering algorithm. It is performed in 2 main phases. In the first phase, we ingest all the data points and construct an in-memory CF Tree. The CF tree is created by iteratively inserting new points into appropriate leaf nodes such that the threshold is not violated. In the second phase, we use a general clustering algorithm that is used in each leaf node to generate the subclusters. There are also optional phases where we condense the tree to either make it small or to refine it.

The advantages of this algorithm are two-fold. On a low level, it decreases complexity and thus is more scalable. On a high level, it allows for enough flexibility as we can select any clustering algorithm for the subclustering. The hierarchical nature of this algorithm complements our problem because the genres often have this structure. Also this algorithm is not susceptible to outliers.

6 Recommendation Systems

Now that we have a good clustering over the users, we wanted to be able to exploit our network for providing recommendation for songs to various users. But we cannot have a cold start (as we weren't developing a reinforcement learning model) so we need some initial listening data about the user.

6.1 Traditional Recommendation Systems

In order to contrast the efficacy of our system with traditional methods for recommendation we decided to implement User based collaborative filtering and Biased Matrix Factorization Models. These methods were supported by Librec[10]. The algorithms were implemented with the default parameters from Librec, which can be found in their user guide [9].

6.1.1 User Based Collaborative Filtering

Collaborative Filtering (CF) Methods can be grouped into two types: User Based and Item Based. Due to large number of songs it was not computationally feasible for us to run Item based CF. Hence, we decided to go with User Based CF.

User based CF is based on the idea that people who agreed in their evaluation of certain items in the past are likely to agree again in the future. Therefore, CF predicts a test users's interest

on a test item based on rating information from similar user profiles. Since, we don't have explicit rating of the song by user we approximate the user rating by the number of play counts of the particular song.

The algorithm takes as input (UserID,SongID,PlayCount) tuples and first creates a user-user similarity matrix. Popular methods to for similarity measure are Pearson's correlation coefficient and Cosine similarity. We have used the default implementation which uses Pearson's correlation coefficient. Consequently, the predicted play count $\hat{x}_{k,m}$ of the test song m by test user k is computed as:

$$\hat{x}_{k,m} = \bar{u}_k + \frac{\sum_{u_a \in S_u(u_k)} s_u(u_k, u_a)(x_{a,m} - \bar{u}_a)}{\sum_{u_a \in S_u(u_k)} s_u(u_k, u_a)}$$

where $\bar{u}_k$ and $\bar{u}_a$ denote the average rating made by users k and a,respectively and $s_u(u_k, u_a)$ is the similarity between users k and a.[11]. We then recommend the top 50 songs based on these predicted play counts.

6.1.2 Biased Matrix Factorization Models

This technique is an improvement over CF. There are two approaches to CF: neighborhood methods and latent factor methods. Neighborhood methods are good at detecting very localized relationships but poor at detecting a users overall user preferences. In contrast, Latent Factor models are best at estimating the overall preferences of the user but poor at detecting strong associations among small set of closely related items. Due to this complementary nature of the two approaches [12], we can integrate them to provide better recommendations. For eg. neighborhood models are very good at capturing "People who like Nirvana also like Pearl Jam" whereas Latent Factor models can capture the overall preference of the user that he likes the genre grunge. [12] introduces a new neighborhood based model which focuses on optimizing a global cost function unlike CF which uses K-Nearest Neighbors. Also it extends the latent factor model by allowing implicit feedback into the model. This improves system performance as both steps complement each other and can be combined together. After, predicting the rating using this model we recommend the top 50 songs based on these predicted play counts.

6.2 Proposed Recommendation System

Using the test set and cluster configurations, we assign the remaining (test) users to their appropriate clusters. Then we find the features for the test users and recommend songs on the basis of top songs for the clusters closest to that user (different for GMM/K-Means/Birch).

In KMeans, we have 2 ways of recommendations -
1. **KMeans (Counts)** - Recommending songs that have been listened to most times by users of that cluster.
2. **KMeans (Users)** - Recommending songs that have been listened to by most users of that cluster.

In GMM, we would extract features for the new user and calculate the probabilities of it occurring in any of the K clusters and then recommend songs randomly from the clusters based on those probabilities. We tried 2 techniques to decided top songs for a cluster. -
1. **GMM(Hard)** - For each user, we put them into the top cluster based on highest probability of occurrence.

2. **GMM(Soft)** - For each cluster, we choose the Top M users (based on probabilities of occurring in that particular cluster). M was chosen to be N/K where N is the total number of users so as to maximize inclusion of all users.

7 Evaluation

7.1 Evaluation Metrics

Initially, we compared recommended songs with the actual (top) songs listened to by that user and evaluated our 3 recommender schemes accordingly. After performing these tests, we obtained quite low values of accuracy (around 3%) because it is a very hard task to predict the actual songs the user has listened to, out of the large number of songs that had been listened by users in that cluster. Moreover, the task is to recommend songs which user might like and not to predict songs that a user has already listened to. The ideal accuracy should be based on whether that user liked the recommended song or not. But that is very subjective.

To overcome this issue, we use 3 evaluation metrics -
1. Artist accuracy - How many artists that the user listens to, were we able to recommend
2. Genre accuracy - How many genres that the user listens to, were we able to recommend
3. Weighted Genre accuracy - How many genres that the user listens to, did we recommend, weighted by the number of times he listens to that genre.

7.2 Baseline

We used Majority Algorithm (top songs based on maximum play counts) as our baseline. We find out the songs which have been listened to the most number of times and recommend them to all users. In the clustering model, it basically means that we put all users into the same one cluster.

7.3 Oracle

Generally in unsupervised setting it difficult to evaluate your models as we don't have a ground truth. We use the following scheme for Oracle: we "cheat" and recommend songs (top 50) from user's own past playlist. This is assuming we have greater than 50 songs per user. In case of less than 50 songs in user's playlist, we recommend the entire playlist plus most popular songs from whole dataset. While accuracy here will be high, it won't be 100% since when a user's playlist is large, only 50 top songs are being recommended.

8 Results and Discussion

In general we observed that there is not necessarily a one to one correspondence between the silhouette based clustering efficiency and the accuracy of recommendation system we built on top of it. We observed using the silhouette coefficients that the optimal value of k was 18. However, as we can see from Table 2 depending on our evaluation metric the optimal value was either 16 or 18. This can be attributed to the fact that k-means is susceptible to outliers because it tries to optimize mean squared distance. This becomes evident from the silhouette plot for every user. Many users have a negative silhouette value which means they are potentially assigned to wrong clusters. Since our recommendation from k-means depends entirely on the songs from the assigned clusters this results in low accuracy values.

Algorithm	Clusters	Artist Accuracy	Genre Accuracy	Weighted Genre Accuracy
KMeans(Counts)	18	0.67	23.25	35.88
KMeans(Users)	16	0.81	23.62	35.99
GMM(Soft)	29	0.94	24.31	37.56
GMM(Hard)	16	0.76	23.52	36.37
BIRCH	87	1.73	45.76	69.87
BiasedMF	-	0.92	26.83	40.09
UserCF	-	2.98	21.12	39.94
Baseline	-	0.45	16.12	31.43
Oracle	-	79.38	57.17	87.31

Table 2: Accuracies for various algorithms

Algorithm	100 users	1000 users	15000 users
KMeans	1.06	3.89	76.51
GMM	7.65	73.26	842.41
BIRCH	0.89	1.78	3.77
BiasedMF	7.53	26.73	$\geq$3600
UserCF	169.85	1397	$\geq$3600

Table 3: Running time (in secs) of various algorithms

Further we can observe that the recommendation accuracy for GMM (hard) clustering is same as K-means. Contrary to our intuition GMM (soft) clustering did not perform significantly better than k-means. On further analysis we realized that a user may have overlapping taste in terms of genres but since the evaluation is based on genre recommendation it is essential to capture the structure within genres. This task was specifically suited for hierarchical clustering. As we can see BIRCH perform significantly better than both the clustering algorithms.

In general, our clustering algorithms perform better than the baseline of recommending the most popular songs and perform better (BIRCH) or worse (KMeans/GMM) than the state of the art CF based recommendation methods. However, this can be attributed to our approximation of user song rating by user song play count. This might not always be a good approximation as many times listening to music the user might be multitasking and not be focusing on the song itself. This would increase the play count but not necessarily imply that the user likes the song. One more advantage of clustering based algorithms is scalability. We were unable to run BiasedMF and User based CF for more than 1000 unique users. Both the algorithms need a user-user similarity matrix which becomes increasingly computationally intensive as we increase the number of unique users and songs. In contrast clustering based algorithms are scalable and faster. In particular, BIRCH makes large clustering problems computationally tractable and the I/O complexity is a little more than one scan of data [13].

9 Conclusion

We have shown that the identification of community structures within music networks is a useful tool in order to recommend music to users. Basic clustering algorithms, like Kmeans and GMM, perform only slightly worse compared to state of the art collaborative filtering algorithms. However, this may also be attributed to approximating song play count as ratings for collaborative filtering algorithms. We can conclude that hierarchical algorithms (BIRCH) perform better for music recommendation because they can exploit the inherent structure present in song meta-data (for example, genres). In addition to the recommendation accuracy, clustering based algorithms (especially BIRCH) are also more scalable. They are computationally less intensive then CF based methods and can be used to provide quick and efficient recommendations. Also, we observed that contrary to our initial belief that overlapping clusters would be best suited for this task, hierarchical based clustering method like BIRCH were able to improve the recommendation accuracy significantly.

In future we could use identification of community structures within music networks to evaluate the role of leading artists identify each community. Also, since now Last.fm API keys are available we could collect time series data and use it to study changes in user song tastes and community memberships over a period of time and leverage this information to improve our recommendations systems.

References

[1] Clauset, Aaron, Mark EJ Newman, and Cristopher Moore. "Finding community structure in very large networks." Physical review E 70.6 (2004): 066111.

[2] Newman, Mark EJ. "Detecting community structure in networks." The European Physical Journal B-Condensed Matter and Complex Systems 38.2 (2004): 321-330.

[3] Banerjee, Arindam, et al. "Model-based overlapping clustering." Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. ACM, 2005.

[4] Bertin-Mahieux, Thierry, et al. "The million song dataset." ISMIR 2011: Proceedings of the 12th International Society for Music Information Retrieval Conference, October 24-28, 2011, Miami, Florida. University of Miami, 2011.

[5] 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.

[6] Breese, John S., David Heckerman, and Carl Kadie. "Empirical analysis of predictive algorithms for collaborative filtering." Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc., 1998.

[7] Sarwar, Badrul, et al. Application of dimensionality reduction in recommender system-a case study. No. TR-00-043. Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[8] Allen, Robert B. "User models: theory, method, and practice." International Journal of man-machine Studies 32.5 (1990): 511-543.

[9] http://www.librec.net/tutorial.html

[10] Guo, Guibing, et al. "Librec: A java library for recommender systems." Posters, Demos, Late-breaking Results and Workshop Proceedings of the 23rd International Conference on User Modeling, Adaptation and Personalization. 2015.

[11] Wang, Jun, Arjen P. De Vries, and Marcel JT Reinders. "Unifying user-based and item-based collaborative filtering approaches by similarity fusion." Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 2006.

[12] Koren, Yehuda. "Factorization meets the neighborhood: a multifaceted collaborative filtering model." Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2008.

[13] Zhang, Tian, Raghu Ramakrishnan, and Miron Livny. "BIRCH: an efficient data clustering method for very large databases." ACM SIGMOD Record. Vol. 25. No. 2. ACM, 1996.

[14] Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E. "Scikit-learn: Machine Learning in Python" Journal of Machine Learning Research. Vol. 12. 2011.