Music Recommendations based on Implicit Feedback and Social Circles: The Last FM Data Set

Abhishek Majumdar[1], Aravind Kumar K[1] and Sriram Manohar[1]

***Abstract*— The goal of recommender systems is to make personalized product recommendations based on users taste. In this paper we perform an exploratory analysis on the LastFM data set. Based on the data set properties we use collaborative filtering , latent factor models and propose community detection using clique percolation to give personalized artist recommendations to the user. We circumvent the implicit network feature and give reasonable recommendations to the user.**

I. INTRODUCTION

One of the most exciting results of music consumption transferring into the digital online marketplace is the explosion of user data that is generated as a result. Thanks to access to user's social circles, tagging behaviour and viewing history through the LastFM API- a number of music recommender system algorithms have been developed using the LastFM dataset.The data publicly available through this free API serves as the inspiration for this project.

In this project we study the network properties of the graph generated from the LastFM Dataset and try to come up with personalised artist recommendations for users. After an initial exploratorty analysis of the data set we observe that the network is a highly dense connection of users with list of artists, friends and tags as features. We apply and evaluate techniques such as Neighbourhood Modelling, Community Detection and Latent Factor Models on this data set and compare them with relevant baselines.

II. DATASET AND GOALS

A. Collection

The freely available and wide range of last fm APIs[5] were used to collect data, starting with a single user as the seed and going up to 5 layers deep. The first important idea was to study the network of friends in order to observe the similarity in listening behaviour among friends. The second important information needed from the dataset was the implicit behaviour of users, which included their tagging activity and top playcounts for each artist. The last.fm APIs artist.getTags, User.getTopArtists, user.getNeighbours and User.getFriends were used to collect the dataset.

B. Basic Statistics

Table 1 demonstrated the basic statistics of the dataset that was collected.

*This work was not supported by any organization

[1]Department of Computer Science and Engineering, University of California- San Diego abmajumd,akumark,srmanoha at ucsd.edu

TABLE I
DEFINING THE DATASET

Number of users	1689
Number of artists	12110
Number of unique tags	8984
Number of user, artist, tag tuples	170249
Number of bi-directional friendships	12527
Number of user, artist, playcount tuples	82911

C. Preliminary Data Analysis

To further understand the properties of our dataset, the various relationships such as user-user, user-tags, user-artist, artist-tags were analyzed.

The friendship network with 12527 bidirectional relationships(14.833 friends per user on an average) is dense as can be seen from Figure 1. In order to analyze the overlapping community structure in this network, clique percolation technique was used. A community is the maximal union of k-cliques that can be reached from each other through a series of adjacent k-cliques. We observed that the maximum value of k was 12 before we ended up with zero communities. This opens up the possibility of studying the musical tastes within a close knit community of friends.

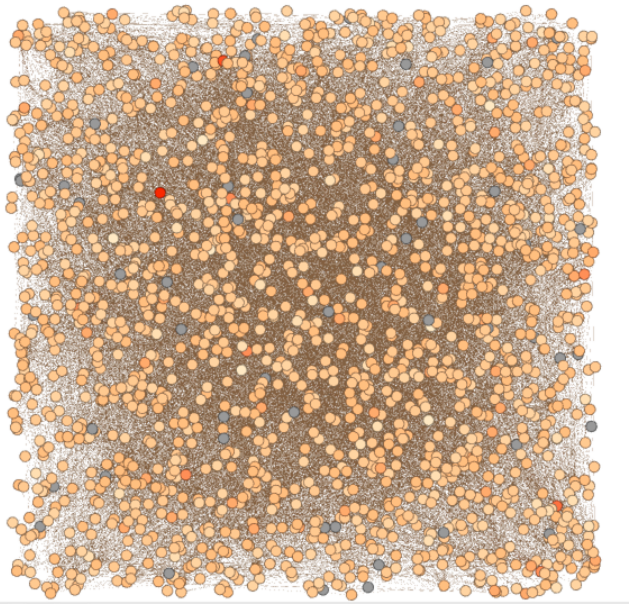

Fig. 1. The dense friendship network in the dataset

The frequency of tags that have repeated atleast twice is

plotted in sorted order in figure 2. It was observed that the tags have a longtail distribution which was expected. There were a few tags which repeated a lot of times in the dataset namely ('rock', 6697), ('pop', 5143), ('alternative', 4750), ('electronic', 4380). Also, some interesting properties that we found were that alternate rock occurs with rock 74 percent of the time, indie rock occurs with rock 72 percent of the time, rap occurs with hip-hop almost 95 percent of the time. Such simple overlaps show which tags are least informative. Essentially, this opens up the possibility of clustering tags while building tag based recommender systems.

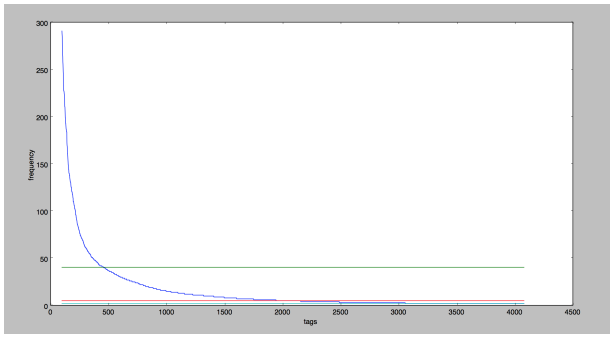

Fig. 2. Frequency of tags vs the Tags repeated at least once
Mean=40.5063694268
Median = 5.0
Mode = 2

Plotting the sum of normalized playcount of each artist which is the sum of each user's playcount for a particular artist divided by his total playcount as shown in Figure 3, gives a more unbiased idea of who the popular artists are. It turns out that Britney Spears, Lady Gaga and The Beatles are at the top of this study.

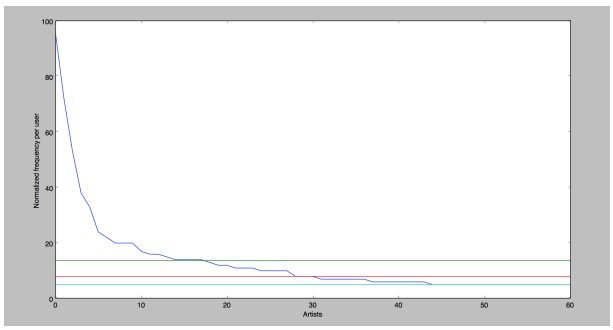

Fig. 3. Normalized frequency per user vs Artists
Mean=13.4590163934
Median = 8.0
Mode = 5

D. Data Pruning

To remove the effect of the longtail of tags and artists, the tags that repeated less than 6 times were removed and every artist who received tags less than 2 were removed. This reduced the number of user-artist-playcount tuples to 60134 from 82911 and the number of user-artist-tag tuples to 71686 from 170249

E. Goals

From the basic analysis of the dataset, the goal was to develop models that would exploit:

- The implicit feedback from the users which is given by the playcount for each artist,
- The rich collection of tags given the user-artist-tags relationship.
- The dense network of friendship relationships between users.

This motivated our literature survey and the 3 models that were designed as described in the following sections

III. LITERATURE SURVEY

For implicit feedback datasets such as the last.fm dataset which does not have ratings etc., a number of algorithms have been used that suit this specific requirement. The literature surveyed included collaborative filtering algorithms proposed and implemented on the lastfm and movielens datasets. CF was found to be the most popular technique that was used to filter information. In memory-based CF the similarity between two users or items are calculated based on Pearson Correlation Coefficient similarity, cosine similarity and Jaccard similarity. Then, according to the nearest neighbors opinions the most interesting items for the target user are recommended. Tso-sutter's work focussed on tag based recommendation[2] while there were future works[4] that fused tag based models with social network analysis. Our interest in understanding the effect of social network in predictive analysis motivated us to pursue a model based on [4].

There were very few papers that used the playcount information to build a model. Pacula's model[1] converted the playcount information into a rating scale and was based on matrix factorization algorithms. To sum up, in this project, we would like to explore the models presented in[1],[4] and their performance on our dataset.

Also, we have implemented a clique percolation based model to see the effect of social networks on recommender systems and study their performance.

IV. LATENT FACTOR MODELS

The key idea behind latent factor models is to project the users and items into a smaller dimensional space (such lower dimensional projections are called factors), thereby clustering similar users and items. Subsequently, the interest (similarity) of a user to an unrated item is measured and the most similar item(s) is(are) recommended to the user.

One of the most efficient ways to build such systems is based on Matrix Factorization. Matrix factorization is usually used in high-quality explicit feedback datasets, where users make their preferences known by directly rating subsets of available items on a fixed scale.

However , in the LastFM dataset there is no option for the user to rate artists or playlists. We hence use an implicit feedback mechanism to generate ratings for a given user-artist combination. For each user we know the number of

times the user played songs by a given artist, but we do not have direct ratings for that artist. As a result, in order to use the matrix factorization approaches described earlier, ratings must be estimated from play count information. We define play frequency freq for a given user i and artist j to be the users play count for that artist normalized by users total plays:

$$freq_{i,j} = \frac{count(i,j)}{\sum_{j'} count(i,j')} \tag{1}$$

We also adopt the notation $freq_{k'}(i)$ to denote the frequency of the k-th most listened to artist for user i. As Figure 3 shows, play frequencies have a clear power law distribution. A rating for an artist with rank k is computed as a linear function of the frequency percentile.

$$r_i, j = 4.(1 - \sum_{k'=1}^{k-1} freq_{k'}(i)) \tag{2}$$

A. Baseline

1) Baseline 1: We use the α model as the first baseline. This model computes the gloabal average of the ratings which was reported to be 2.97189617732. This gave a mean-squared error of 0.74330392221 on the test set.

2) Baseline 2: We use the β_u and β_i model as the second baseline. This algorithm is described in [3]. The parameters β_u and β_i are estimated using a decoupling method, which requires less complexity but are less accurate. We choose λ as 3.25 which is a parameter applied to estimate β_u and β_i respectively.

Table 2 shows the results of the baseline models.

TABLE II
BASELINE MODEL

Name of the Model	Train Error	Test Error
α	-	0.74330392221
β_u β_i with $\lambda = 3.0$	0.55443212919	0.67668413290
β_u β_i with λ = 5.0	0.59846212789	0.69331476312

This shows that by choosing proper parameters of λ we can improve test errors. Our experiments showed that the best value of λ was between 1 and 5. We have however shown results for $\lambda = 3.0$ and 5.0 as they give as reasonable parameters to which we can compare our model.

The predicted rating is

$$rating_{u,i} = \alpha + \beta_u + \beta_i \tag{3}$$

B. MATRIX FACTORISATION ALGORITHM

1) Theory: The Matrix factorization algorithm we use is inspired by Singular Value Decomposition of the user-product rating matrix $M_{i,j}$ defined as

$$M_{i,j} = r_{i,j} \tag{4}$$

where $r_{i,j}$ is the observed rating of product j by user i (Koren et al., 2009). The matrix M is usually sparse, with the Netflix Challenge matrix having less than 2% of all possible ratings available combinations(Funk, 2006). For m users and n products, a typical model decomposes M into two matrices U and P of dimensions f m and f n, respectively, such that M is approximated by the product U and transpose of P.

$$r_{i,j} = u_i^T p_j \tag{5}$$

Parameter estimation is accomplished by minimizing the regularized square error between observed and estimated ratings

$$min_{U,P} \sum_{r_{i,j}} (r_{i,j} - u_i^T p_j)^2 + \lambda(||u_i^2|| + ||p_j^2||) \tag{6}$$

Stochastic gradient descent can be used to estimate the parameters.The error term $e_{i,j}$ is defined as below:-

$$e_{i,j} = r_{i,j} - u_i^T p_j \tag{7}$$

The algorithm then uses this error to modify its parameters by a magnitude proportional to γ using gradient descent.

$$u_i \leftarrow u_i + \gamma.(e_{i,j}.p_j - \lambda.u_i) \tag{8}$$

$$p_j \leftarrow p_j + \gamma.(e_{i,j}.u_i - \lambda.p_j) \tag{9}$$

Thus through minimizing equation 6 and updating equations 7 and 8 for finding the optimal values of the User and Artist Matrix we can get the optimum user and artist features.

2) Implicit Problem and the Algorithm used: One of the most significant differences between explicit and implicit feedback data sets is the distribution of ratings. In an explicit setting, the average user only rates a reasonably small subset of products, and the ratings are not heavily skewed towards one end or the other. Our data set was skewed towards the right as shown in the figure 4.

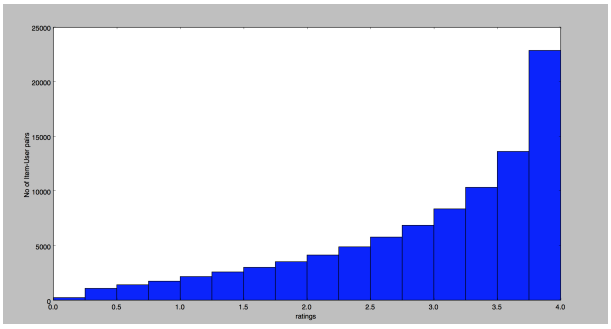

Fig. 4. Rating Distribution of the Data Set - No of User-Items versus rating

This demonstrates that ratings can be uncertain. We also see here that most of the users translate to a score between 3 and 4. To model this uncertainty, confidence coefficients is introduced.[6]

The confidence score is a linear function of the rating and is of the form:-

$$c_{i,j} = 1 + \alpha.r_{i,j} \tag{10}$$

where α is determined through cross-validation. The equations hence become:-

$$u_i \leftarrow u_i + \gamma.(c_{i,j}.e_{i,j}.p_j - \lambda.u_i) \quad (11)$$

$$p_j \leftarrow p_j + \gamma.(c_{i,j}.e_{i,j}.u_i - \lambda.p_j) \quad (12)$$

We now use the stochastic gradient descent algorithm to do parameter estimation.

3) Results: As we see in figure 5 we could not arrive at the elbow point for testing after 280 iterations. The training error at the end of 280 iterations with 32 features was 0.37441886112441169 while the test error was 0.57069697841812323. We stopped at 280 iterations because the difference in train error between subsequent iterations was small.

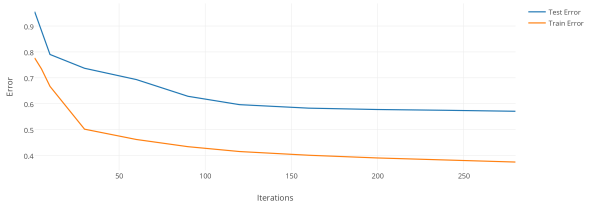

Fig. 5. Training and Test Error versus number of Iterations

Figure 6 is a plot between number of features and error.

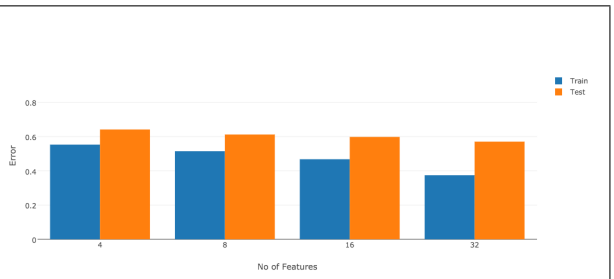

Fig. 6. Error versus number of features

Adding more features gave us better RMSE values. We had experimented this model with 4 , 8 ,16 and 32 features respectively. The test error after 280 iterations for 4 features was 0.6416538452231 and that reduced to 0.57069697841812323 with 32 features. Hence we conclude more the features better is the model. Due to the fact that the model is computationally expensive we could not try it on larger number of features. The γ value chosen is 0.0012, and λ value is 0.8. This is chosen randomly based on trial and error.

V. TAG BASED CF INTEGRATING SOCIAL NETWORK INFORMATION

In this method, we use a new metric proposed in [4] which combines the implicit and explicit relationships among users viz., the tagging behavior and the friendship formed respectively. The objective is to improve the accuracy of recommendation based on tagging behavior and friendship network of a particular user. Friends generally share common tastes and it is likely for them to give a good idea of what the user would like.

The similarity metric proposed in [4] gets the user similarity based on item-tag pair and friendship. This is then combined with item-item similarity with the user's friends and return the top N recommendations of a particular user.

A. Item-Tag similarity

A measure of common tags by users gives would be inappropriate since two different users could give the same tag to totally different items. Also, measuring common items alone could again give rise to a situation where the tags assigned to the common items by the users are different. Here a metric is used to measure only the common tags assigned to the same item by users.

$$TSim_{u,v} = \frac{\sum_{I_u \cap I_i} \frac{|T_{uv_i}|^2}{|T_{u_i}| \times |T_{v_i}|}}{Max(|I_u|, |I_v|)} \quad (13)$$

B. Friendship based similarity

For a user u in the system, AvgFu (the average of the item-tag based similarities of all the friends of u) is calculated. The item-tag similarity measure is then amplified if the friend user v is very similar to u.

$$FSim_{u,v} = Tsim^{\frac{1}{1+FSim_{u,v}+AvgFu}} \quad (14)$$

C. Overall similarity

$$Sim_{u,v} = \alpha \times TSim_{u,v} + (1-\alpha) \times Fsim_{u,v} \quad (15)$$

D. Recommendation Algorithm

Here, the friends of a given user u are taken. The interest level of a user u to a particular item that is selected by friend v depends on two main components: the similarity of v to u and the similarity of item i selected by v to the items tagged by u. In order to find the similarity of items in view of tags that are assigned to them by users, the weighted Jaccard similarity is used.

$$SimItem_{i,j} = \frac{\sum_{V_i \cap V_j} Min(V_i(t).Fq, v_j(t).Fq)}{\sum_{V_i \cap V_j} Max(V_i(t).Fq, v_j(t).Fq) + \sum_{V_i \cup V_j - V_i \cap V_j} Max(V_i(t).Fq, v_j(t).Fq)} \quad (16)$$

Here vi is denoted to the vector of item i that has pairs of tag name and frequency of that tag. The values calculated above - Overall Similarity and Item Similarity are then input to the algorithm described in the paper. The algorithm is as follows:

Step1:
For v as neighbor of u:
    For item i in vs item list
        For item j in us item list
            ItemInterests[i].ADD(j, $SimItem_{i,j}$*$Sim_{u,v}$);
            NIAvgs.ADD(i, ItemInterests [i].AVG());

Step2:
For gi as NIAvgs.Groupby(i)
TopNs.ADD(gi.Key, gi.Max());
Step3:
TopNs.Sort();
RETURN TopNs.Select (Item);

First, multiplications of item similarity of item list of user u and her/his friends item lists and user similarity of u and v are stored in an array of lists, called ItemInterests. In the next step the average value of duplicated items is calculated and stored in NIAvgs. The algorithm finds the maximum value of each item and returns a sorted list of items based on their final values.

E. Results

The baseline model used for comparison was a global artist recommender that recommended the most popular top-N artists.

We used the all but 1 protocol similar to what Tso-Sutter used in his paper[1] to calculate and compare the recall values.

As can be seen in figure 7, this model only marginally performed better then the baseline for certain values, but most of the pattern was rather random and no proper trend could be established as the Tag Similarity (Tsim) between the users were mostly zeroes and the matrix, was rather sparse to compute any predictor effectively. For this given dataset, this model is not ideal and could perform better with a more dense network matrix.

A further refined model can be extracted from this by incorporating the Group Similarity between the users and their groups, but this extension is highly computational and was beyond our time frame to perform a group similarity on this network. Also, a lastfm API to extract group information given a particular user was not available. This could have probably improved the results.

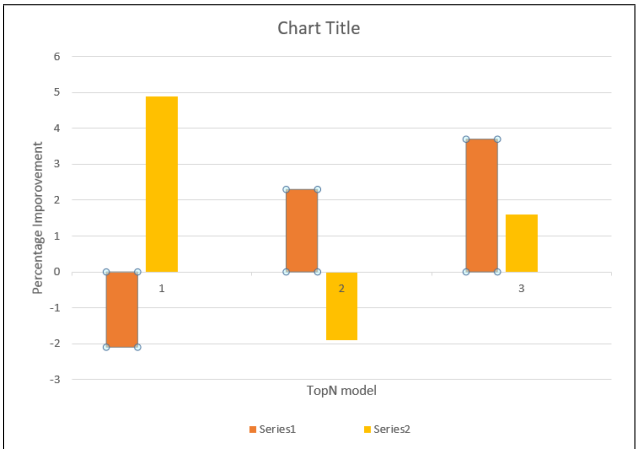

Fig. 7. Percentage improvement in recall values over the baseline model

VI. CLIQUE PERCOLATION BASED MODEL

As the network was extremely dense, a clique percolation analysis was performed to see if there was a pattern in which users in a given clique behaved. The network was broken down into smaller Subgraphs which formed independent K-Cliques networks. For our analysis we observed for clique size (K) varying from 2-10.

The designed method of recommendation was subdivided into 2 approaches:

- TopN Frequency based Item suggestion
- Top-N Normalized Frequency based item suggestion

A. Baseline

Artists with the top N playcounts were recommended to every user.

B. TopN Frequency based Item suggestion

In this approach we analyzed Each Sub-Network for a given K-Clique by splitting the user in Training set(70% users) and Test set (30% users) and counting the Frequency of the Artists in the Training Set and recommending the Top 3 Artists to the Users in that observed clique to the Test Set. The Performance of this model was gauged by the Recall of Artists for each user.

Drawback : Since we added the frequency of artists by each user directly , there is a possibility of the ranking to be biased to an artist because of certain users who view the artist excessively or too rarely. To get rid of this bias we added the weighted frequency by each user. This method takes negates the bias by taking the viewing activity of each user and Normalizing the frequency.

C. Top-N Normalized Frequency based item suggestion

In this method, we analyze the viewing frequency of each user and normalize the overall frequency of an artist by the given users maximum view. We then perform the similar process of taking the Normalized frequency count and ranking the artist in the given clique.

D. Results

The top 3 Artists acquired by each of the models is suggested to the users in the clique and the Recall is considered as the performance metric for this process.

As can be seen from figure 8, the Top-N Normalized Frequency based item suggestion model performed marginally better than the TopN Frequency based Item suggestion model as can be seen from the blue and green lines respectively. Also, it can be seen that it performs much better than the baseline represented by the red line. The graph gives a good intuition on the results that were developed. It shows that as the clique size increases, we are seeing an improvement in the precision values that are obtained up to a certain point. The drawback of these models is that it can be a good predictor only if the user has a densely connected network of friends although the model still predicts better than the baseline.

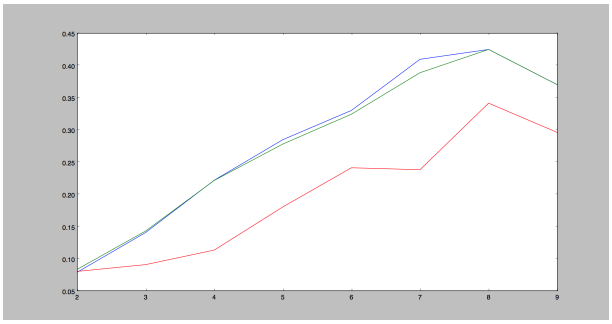

Fig. 8. Percentage improvement in recall values over the baseline model, Red- baseline, Blue- Model C, Green- Model B

VII. CONCLUSIONS

In Section 4, 5, and 6 we have developed 3 models that are models independent of each other. One is based on implicit playback count of every user, one does collaborative filtering analysis on the tags assigned to artists by different users and also combining it with friendship relationships. The other is based on friendship network in which recommendations are made only based on clique behavior. The results, drawbacks and scope of improvement of the different models have been assessed and evaluated in their respective sections. However,there is no clear baseline to evaluate the performance of these models against each other. The results that have been obtained are mostly consistent with the ongoing research.[1],[4].

REFERENCES

[1] Maciej Pacula. A matrix factorization algorithm for music recommendation using implicit user feedback.

[2] K.H.L. Tso-Sutter, L.B. Marinho, and L.Schmidt-Thieme. Tag-aware Recommender Systems by Fusion of Collaborative Filtering Algorithms, Proceedings of the 2008 ACM symposium on Applied computing, ACM, USA, 2008, pp. 1995-1999

[3] Recommender Systems Handbook. Kantor, Paul B ; Ricci, Francesco ; Rokach, Lior ; Shapira, Bracha

[4] Huizhi Liang, Yue Xu, Yuefeng Li, Richi Nayak, Collaborative Filtering Recommender Systems Using Tag Information, 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology

[5] Last FM API http://www.last.fm/api/

[6] Yifan Hu, Yehuda Koren, Chris Volinsky. 2008. Collaborative Filtering for Implicit feedback Datasets. Proc. IEEE International Conference on Data Mining (ICDM 08), IEEE CS Press, pp. 263- 272