Group 12 Final Report
Recommendation System on Million Song Dataset

Big Data Analysis (BYGB 7990 002)

Yidi Kang
Ruiqing Lyu
Peiying Wu

1. **Executive Summary**. (0.5-1 page)

With the rapid development of mobile networks and digital multimedia technologies, digital music has become the mainstream consumer content sought by many young people. According to Nielsen's Music 360 2014 study, 93% of the U.S. population listens to music, spending more than 25 hours each week jamming out to their favorite tunes [1]. Many successful music apps such as Spotify, Netease music and QQ music use recommendation system as their core methodology. Nevertheless, personalized music recommendations have also become a research hotspot in the recommendation field. In this project, Our team conducted a study to predict which songs will be recommended to end users. We used Million song dataset, which is a collection of audio features and metadata for a million contemporary popular music tracks. We have designed, developed and implemented two contented-based recommendation systems based on our selected features. Since the size of this dataset is big, we first used python to build the model and apply it on spark for all dataset. We used our test dataset to generate a predictive result and use it to make a comparison with the original test dataset. Finally we calculated precision, recall and coverage score to evaluated our recommendation system,the precision score is 0.2614 which means that 26% recommendation in our list actually happened in the actual test result.

2. **Business Problem** (0.5-1 page)

Nowadays, the massive and huge heterogeneous music data generated by it undoubtedly exceeds the basic needs and affordability of the audience, which in turn leads to user information fatigue. Therefore, how to use music personalized recommendation to help users quickly and accurately obtain music tracks of interest in the vast music library is becoming more and more important. This project is conducted to predict which song will be recommended to a user by relying on the Million Song Dataset.

By building the recommendation system, the online media industry have the potential to change the way music apps communicate with end users and to allow companies to maximize their Revenue on Investment by investigating the information they can gather from end users including music preference and purchases.

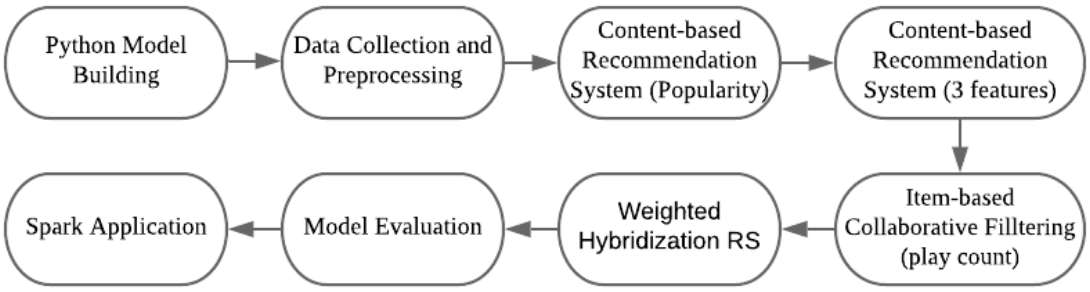
3. **Dataset Description** (1 page)

In this project, we use train_triplets as our main dataset. This dataset contains a full music listening history of more than 1 million users, the data size is 2.8GB, and we divided this dataset into 70% training set and 30% testing set. Main features for this dataset are user information, song information and play count which record how many times this song been played.

We also used two addition dataset, Track_metadata.db has 728 kb data size, it contains metadata about each track, main features are a song title, song hotness, and duration. The other one is Unique_tracks.txt, this dataset gets the names of song, including song id, song title, track id, etc.

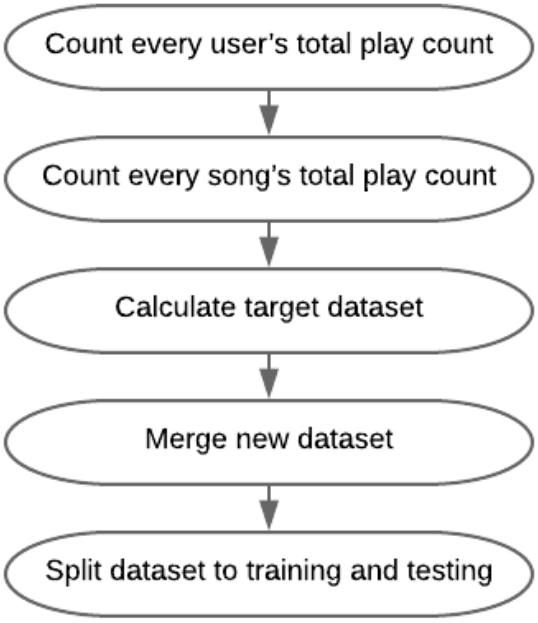
Dataset Name	Size	Source	Function	Description
train_triplets.txt	2.8GB	The Echo Nest Taste Profile Subset	Core dataset	A full music listening history of a litter over 1M users "User_id", "Song_id", "Play count"
Track_metadata.db	728.692 KB	MSD Additional Files	Relative information	Containing most metadata about each track "Title", "hottness", "duration" etc.
Unique_tracks.txt	80MB	MSD Additional Files	Get the names of songs	List all track id "Track_id", "Song_id", "artist_name", "song_title"

4. **System Design** (0.5 page)



The project is focusing on the recommendation system which is an unfamiliar area for us to practice, so we decide to use Python for building the model first and then apply the model on Spark. For saving time and enabling Python to run effectively, we eliminated the data set first (the algorithm will be discussed in data preprocessing). We developed two content-based recommendation systems which are based on popularity and 3 features that we picked and one item-based collaborative filtering that based on the play count to generate a hybridization recommendation system. The system used the training data set that we have split. We weighted the three similarity scores and reach our final recommendations for the players based on the rank. To evaluate our final model, we use the testing data set and generate the precision score, recall score and coverage score. Finally, we use the full listening history data set and apply the whole model on Spark to execute our recommendation system.

5. **Data Preprocessing** (1- 1.5 pages)



Python is an application not suitable for handling the huge dataset such as our full listening history data. Since Spark is an unfamiliar analytics engine with us, we still decide to build our model on Python first. Under this circumstance, we have to eliminate our dataset. The original dataset has three columns: user id, song id, and play count.

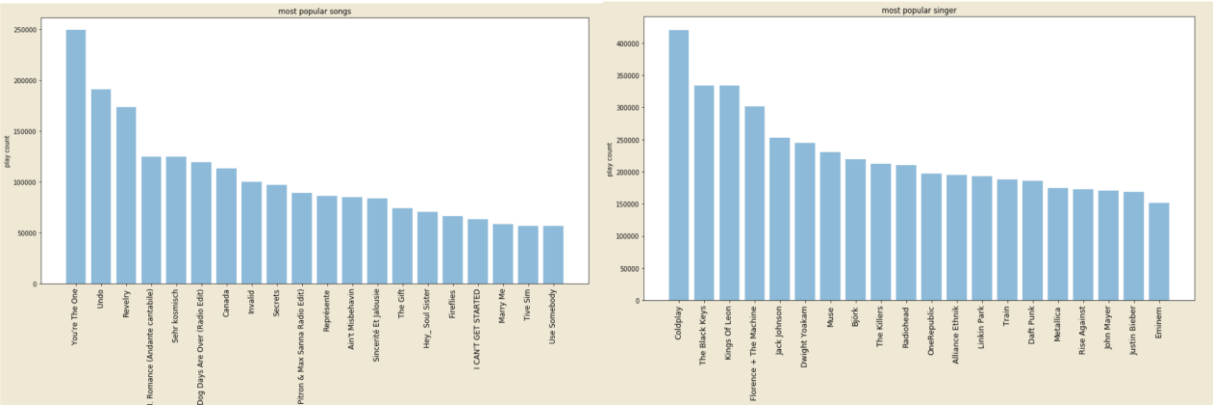
	user	song	play_count
0	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOAKIMP12A8C130995	1
1	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOAPDEY12A81C210A9	1
2	b80344d063b5ccb3212f76538f3d9e43d87dca9e	SOBBMDR12A8C13253B	2

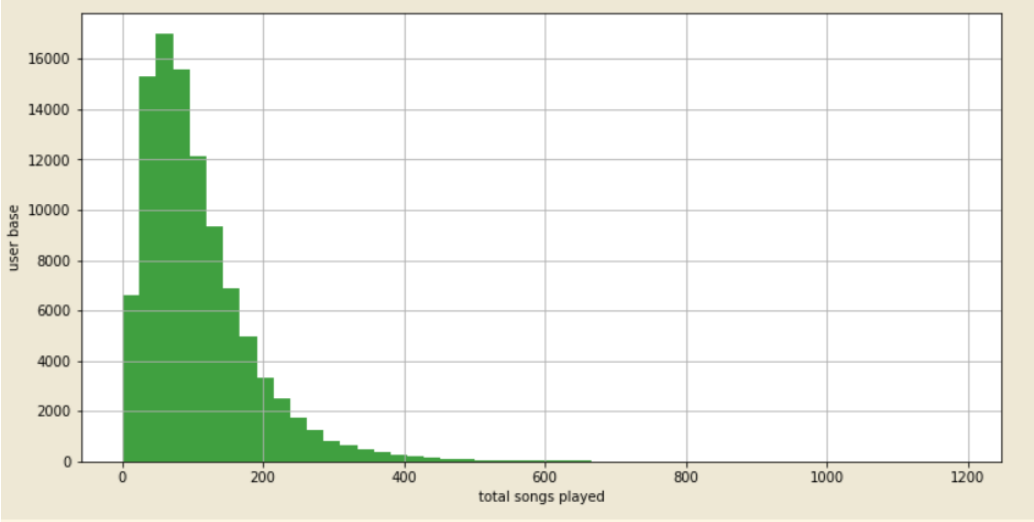
The algorithm based on the consideration that if 20% of the players played 80% of the total play count, we can train our model on this 20% players. Firstly, we count every user's total play count and every song's total play count. After calculating these two data, we found the 80% of the total play account is played by the top 40% players, which means these players can be our representatives of the whole listening history to train the model. We reserved the top 40% of players and their listening history and deleted other data for the moment. The deleted dataset is in the controllable range and will be our sub_dataset used in Python.

```
print ((float(play_count_df.head(n=100000).play_count.sum())/total_play_count)*100)
print (float(song_count_df.head(n=30000).play_count.sum())/total_play_count*100)
```

```
40.8807280500655
78.39315366645269
```

Next, we merged the sub_dataset with the track_metadata to obtain information about artists and songs. The data is an SQL dataset, so we need to transform it into a table. After that, we merged it with the sub_dataset using left inner join with the constraint of 'song' = 'song_id'. We can use descriptive analysis on our merged_dataset now to get some insights into the data we've got, such as the what are the most popular songs, who are the most popular singers and the distribution of the number of songs a player listened.





The last thing for the data preprocessing is to split our data into training and testing dataset, which is used to evaluate our model. To evaluate the performance of our recommendation system, we randomly took out some users' listening history and put in our model to predict the blank. Then the prediction result and the actual result will be compared to generate some evaluation scores.

6. **Algorithm Development** (1-2 pages)

Baseline Approach

Our baseline approach is the recommendation system based on popularity. It is very simple to understand that if a song is fond of most people, it won't be worse for us to recommend it to more people. The problem is this approach is too simple to personalized every user's taste, so we developed a content-based recommendation system based on the following steps: (1) Find the songs that user k have listened. (2) Fill in the cooccurrence matrix. (3) Calculate the similarity scores of other songs based on the songs have listened. The algorithm for calculating the similarity score that we used is the Jaccard Index, which is the intersection over union that is calculated by the code shown below.. (4) Rank the similarity scores and recommend songs which have higher scores.

```
for j in range(0,len(user_songs)):
        #Get unique listeners (users) of song (item) j
        users_j = user_songs_users[j]
        #Calculate intersection of listeners of songs i and j
        users_intersection = users_i.intersection(users_j)
        #Calculate cooccurence_matrix[i,j] as Jaccard Index
        if len(users_intersection) != 0:
                #Calculate union of listeners of songs i and j
                users_union = users_i.union(users_j)
                cooccurence_matrix[j,i] =
                float(len(users_intersection))/float(len(users_union))
```

Here is the example result of the recommendations of a user:

	user_id	song	score	rank
0	8c62b595d55003438fd0125b2df4ada6e0e9cd61	Undo	0.128998	1
1	8c62b595d55003438fd0125b2df4ada6e0e9cd61	You're The One	0.124972	2
2	8c62b595d55003438fd0125b2df4ada6e0e9cd61	Revelry	0.124398	3
3	8c62b595d55003438fd0125b2df4ada6e0e9cd61	Représente	0.120557	4
4	8c62b595d55003438fd0125b2df4ada6e0e9cd61	OMG	0.116242	5
5	8c62b595d55003438fd0125b2df4ada6e0e9cd61	Pursuit Of Happiness (nightmare)	0.115652	6
6	8c62b595d55003438fd0125b2df4ada6e0e9cd61	Invalid	0.114378	7
7	8c62b595d55003438fd0125b2df4ada6e0e9cd61	Dog Days Are Over (Radio Edit)	0.113265	8
8	8c62b595d55003438fd0125b2df4ada6e0e9cd61	Canada	0.111994	9
9	8c62b595d55003438fd0125b2df4ada6e0e9cd61	Sehr kosmisch	0.110925	10

Advanced Approach

Up to now, we haven't build a recommendation system that is personalized customization, so we decided to add more features. Since for listening to the songs, the song

itself and its artists are the two aspects that influence most, we chose the features including artists' hotness, artists' familiarity, and songs' duration. The index of hotness and familiarity are in the range of 0 to 1, but the duration is counted as total seconds, so we need to normalize the duration first. For now, we simply add the three scores and calculate the distance between the sum_score with the song that a user listened most. Use this distance as our second similarity score and make the top recommendation list.

	Unnamed: 0	user_id	song	score	rank	distance
0	0	8c62b595d55003438fd0125b2df4ada6e0e9cd61	SOAUWYT12A81C206F1	0.128998	1	0.007405
1	1	8c62b595d55003438fd0125b2df4ada6e0e9cd61	SOBONKR12A58A7A7E0	0.124972	2	0.052919
2	2	8c62b595d55003438fd0125b2df4ada6e0e9cd61	SOSXLTC12AF72A7F54	0.124398	3	0.157302
3	3	8c62b595d55003438fd0125b2df4ada6e0e9cd61	SOOFYTN12A6D4F9B35	0.120557	4	0.093340
4	4	8c62b595d55003438fd0125b2df4ada6e0e9cd61	SOUSMXX12AB0185C24	0.116242	5	0.095966
5	5	8c62b595d55003438fd0125b2df4ada6e0e9cd61	SOPPROJ12AB0184E18	0.115652	6	0.043069
6	6	8c62b595d55003438fd0125b2df4ada6e0e9cd61	SOUFTBI12AB0183F65	0.114378	7	0.097313
7	7	8c62b595d55003438fd0125b2df4ada6e0e9cd61	SOAXGDH12A8C13F8A1	0.113265	8	0.021458
8	8	8c62b595d55003438fd0125b2df4ada6e0e9cd61	SOPUCYA12A8C13A694	0.111994	9	0.061978
9	9	8c62b595d55003438fd0125b2df4ada6e0e9cd61	SOFRQTD12A81C233C0	0.110925	10	0.071809

To improve our model, we developed item-based collaborative filtering based on the play count. We mapped listeners' play count of all songs and assigned '0' to songs that the user has not listened. After filling in the occurrence matrix, we used Pearson correlation to calculate the similarity score of each song.

We tested a few weights to combines the rank scores of three lists, what we have for now is item-base weights 0.5, baseline approach weights 0.3, and the weight of the features is 0.2, the final weighted hybridization formula is shown below:

Weighted Hybridization: 0.48 Item-Base × 0.3 baseline × 0.22 Content Base

7. Results and evaluations

We used our test dataset to generate a predictive result and use it to make a comparison with the original test dataset. Precision, recall and coverage scores are calculated to evaluate the performance of the recommendation system. Finally, we have a precision score of 0.2614, the recall score of 0.0705 and a coverage score of 0.0846. The precision score means that 26% recommendation in our list actually happened in the actual test result. The recall score indicated 7% of listening history is included in the final recommendation list. And the coverage score means that 8% of songs our recommendations system is able to ever recommend.

```
precision = hit / (1.0 * rec_count)
recall = hit / (1.0 * test_count)
coverage = len(all_rec_movies) / (1.0 * self.movie_count)
print('precisioin=%.4f\trecall=%.4f\tcoverage=%.4f' % (precision, recall, coverage))
```

```
TrainSet = 9426933
TestSet = 1347625Total song number = 30000

Evaluation Result:
precision = 0.2614   recall = 0.0705   coverage = 0.0846
```

To improve our model, we believe we more relevant features should be input in the models to have a more comprehensive recommendation system that can cover songs and artists' relative information. Also, adjusting the weights of the models under different conditions should be an option of improvements. Moreover, besides content based algorithm and collaborative filtering, more recommendation strategies could be implemented.

7. Spark implementation

Till now, we used "kaggle_visible_evaluation_triplets.txt" dataset in our model fitting in Python, this is a subset of "1M_user_playcounts.txt", which is 2.8G. However, we still want to try the whole dataset, because no matter for item-based CF or content based recommendation, more user's data may give us bigger co-occurrence matrix or more accurate information of songs popularity.

So we used Spark to implement item-based collaborative filtering and content based popularity score recommendation on 1M users.

Before Spark, we preprocessed the whole dataset in Python, to generate unique ids in integer type for each song and each user, and the generated ids will be integrated with 1M user dataset in Spark so that the integrated data can be used in ALS model.

Then we can begin the pySpark part. Firstly, we split dataset to training and testing, we fit ALS model with splited training dataset and compute the RMSE via test data. And for each user, our model will generate 10 recommendation to them.

```
>>> userRecs.show()
+--------------------+--------------------+
|              userId|     recommendations|
+--------------------+--------------------+
|026b3e52b951d2f09...|[[20979, 607.6144...|
|c0bf917692fe8affa...|[[83301, 79.98482...|
|07e8f8ec7f72d4e6c...|[[20979, 43.07166...|
|910988119b9b69d02...|[[16261, 189.3777...|
|d249f3c8d25213b0c...|[[29141, 186.9752...|
|8e8281652a61d23ff...|[[10678, 433.3765...|
|b51eeda3c09e2426f...|[[9631, 509.54065...|
|b777c0fbb3d4b6b67...|[[30205, 111.2623...|
|8eb68ed47b15e3824...|[[9631, 1064.5238...|
|d9c3499de67f276d7...|[[30205, 218.9244...|
|c3a94037f5176147a...|[[23657, 168.4577...|
|b99dd800be568c8c7...|[[66036, 319.4322...|
|df1917b424c78bd6e...|[[23657, 350.5113...|
|8be3b3ccecc499441...|[[44026, 166.8813...|
|c107d52b36b0141d6...|[[92027, 456.9249...|
|24b712829acbdc97c...|[[25009, 550.5843...|
|02e5bfba6931178d9...|[[23657, 81.98762...|
|3e2195e7ae4d011f5...|[[23657, 848.9922...|
|c9372b4a4fa6ad355...|[[66036, 138.0413...|
|f3d8a0b53143b053a...|[[44026, 118.0536...|
+--------------------+--------------------+
only showing top 20 rows
```

For content based popularity score, we just computed all songs sum of play counts by users, and generate a list of the 100 most popular songs.

```
>>> pop.sort('sum(playcounts)',ascending=False).show(10)
+------------------+---------------+
|            songId|sum(playcounts)|
+------------------+---------------+
|SOBONKR12A58A7A7E0|        35432.0|
|SOAUWYT12A81C206F1|        33179.0|
|SOSXLTC12AF72A7F54|        24359.0|
|SOFRQTD12A81C233C0|        19454.0|
|SOEGIYH12A6D4FC0E3|        17115.0|
|SOAXGDH12A8C13F8A1|        14279.0|
|SONYKOW12AB01849C9|        12392.0|
|SOVDSJC12A58A7A271|        11610.0|
|SOUFTBI12AB0183F65|        10794.0|
|SOHTKMO12AB01843B0|        10515.0|
+------------------+---------------+
only showing top 10 rows
```

8. **Conclusion and Lessons Learned** (1 page)

First of all, knowing your data well is vital for model building, further optimization and result evaluation. Just as any real-life problems, we start at a position that we have a lot of data we can use but do not know how to. we spend a long time on dataset exploring and getting to know about each feature, which we think really worth it because knowing data well can give you a clearer mind when optimizing models.

Secondly, the data we can use are massive and huge, such as “1M_user_playcounts” dataset (2.8G), and “Metadata” from h5, that means we cannot perform general Exploratory Data Analysis on them. What we did instead is trying to discover the data distribution by using summary formulas into subset data “kaggle_visible_evaluation_triplets”. We tried several ways to have a glance of how the users and songs distributed in the whole dataset, which led us to the conclusion that 80% of the songs are played by 20% users, so we decided that, in Python we build our model using this 20% users’ data, and we managed to eliminate our training data to a scale that Python can handle.

Besides, In reviewing the whole project, what we always wanted to do was to optimize our model and the algorithms inside, to improve the model performance on test data and also let it making more sense. The baseline of our project is content based recommendation system based on popularity and Jaccard Index, which means we will recommend popular songs to users without considering their personalized prefer. Then we optimized our model with an item based collaborative filtering, we took users' play count to songs as ratings, and generate personalized recommendation for each individual. But If our model's main body is item based collaborative filtering, we are going to meet with serious "Cold Start" problem: for the songs and users that have few records in our data, we cannot generate enough recommendation and the accuracy of prediction will be much lower. Thus, we added another model: content based recommendation on 3 indexes which measures the similarity between songs, and will not be limited by users' rating. In this way, even though we do not have enough data to form a co-occurrence matrix in item based CF to generate recommendation, we still can make up the missing part by only focusing on the few records the user already have. By combining all these models together, we managed to give personalized recommendation to individuals as well as improving model performance on new users in prediction, and took weighted vote of final indexes so that we have a more flexible model.

However, during the process, we put majority of the time in developing and testing our model, which is not efficient for us and computationally. In future projects, we are going to using smaller training set and prototype model when developing and testing.

References

1. Everyone listens to music, but how we listen is changing. [online] Available at: http://www.nielsen.com/us/en/insights/news/2015/everyone-listens-to-music-but-howwe-listen-is-changing.html [Accessed 10 Oct. 2017].
2. https://towardsdatascience.com/how-to-build-a-simple-song-recommender-296fcbc8c85