Music Recommender System Based on Collaborative filtering Final Report

Zili Ma, Yang Zheng

Abstract : In this project, we listed the materials and methods which were adopted in our final project. The dataset selected is called Yahoo! Music User Ratings of Songs with Artist, Album, and Genre Meta Information, v. 1.0, We will implement both user-based and item-based collaborative filtering (CF) algorithms in our final project. The recommendation system was implemented on both Spark Java and Scala API.

Keywords: User-based collaborative filtering, Item-based collaborative filtering

1. Introduction

1.1. Motivation

There are three major filtering algorithms to make recommendations. Content-based filtering is to learn what kinds of contents a user likes and then match the contents of a current article with a “content prototype” that we believe describes well what the user likes [3]. Collaborative filtering (user-based filtering) assumes that if users who are similar to the current user like some items, the current user might also like it [3]. A hybrid recommender combines the two, probably also involving knowledge-based and demographic techniques.Because CF algorithm relies on the past user behavior, and it is a data source with much higher information richness, CF is more popular in music recommender system, and as the result, can produce more accurate prediction than that of content-based recommender.

Since customer's choice of music are becoming increasingly dependent on the music recommender system. The value of a music collection would drastically increase, if it can provide an efficient way to explore hidden “treasures” inside users’ music collections. The goal of our recommendation system is to better assist users in finding the right songs based on their music rating histories, or in returning the similar songs when they are looking for without having to enter a very concise query to the recommender system.

In this final report, we first introduce the background and goals of our project, offering a briefly overview about CF algorithm and the dataset we used. Then we describe our motivations, goal tasks and expectations in this project. After that, in the literature review part, we discussed recommendation systems research in general, including a short critique review on several models and algorithms that has been reported aiming to solving the well-known drawbacks of CF algorithm in some music recommendation applications. Following that, we presented our current system design, talked about the challenges we face, the solutions we have come up with, and how we plan to build the system. Subsequently, we discussed the evaluation metrics to ensure the quality of the results as well as the performance of our recommendation system. Finally, we laid out our plans and milestones in a semester timeline table along with our current progress.

1.2. Dataset description

The dataset used for this analysis is Yahoo! Music User Ratings of Songs with Artist, Album, and Genre Meta Information, v. 1.0 from Yahoo dataset. In the dataset, Yahoo music users were asked to rate a song with 5 levels, (a) never, (b) very infrequently, (c) infrequently, (d) often and (e) very often. The dataset contains over 717 million ratings of 136 thousand songs given by 1.8 million users of Yahoo! Music services. The data was collected between 2002 and 2006. Each song in the dataset is accompanied by artist, album, and genre attributes. The users, songs, artists, and albums are represented by randomly assigned numeric id's so that no identifying information is revealed. The mapping from genre id's to genre, as well as the genre hierarchy, is given. There are 2 sets in this dataset. Part one is 1.4 Gbytes and part 2 is 1.1 Gbytes. The ratings include both date and one-minute resolution timestamps, allowing refined temporal analysis. Each item and each user has at least 20 ratings in the whole dataset. The available ratings were split into train, validation and test sets such that the last 6 ratings of each user were placed in the test set and the preceding 4 ratings were used in the validation set. All earlier ratings (at least 10) comprise the train set. Table 1 details the total number of ratings in the train, validation and test sets.

1.3. Project tasks and achievements

In the context of this project we will implement a similar system based on CF algorithm. We believe we can use a different form of decision module from the system described above. The items recommended to a user are those preferred by users with similar music preference. Since we already have the user data initially, we

will not face the widely popular cold start problem. but we still can rely on item-item filtering to recommend documents to the user when the user issues search queries. It is also useful to avoid the mismatch problem caused by new users' limited preference data. For the given data set, we used cosine distance to measure the similarity of each items for doing the item-based CF recommendation by iterating through all music track pairs and computing a similarity metric for each pair. The implementation was completed both on the Spark Java and scala API. For better performance, we tuned the parameters by submitting our Java version of CF model to AWS ECM platform and trained the model with the whole dataset.

2. Literature Review

2.1. Pure Collabarative filtering algorithm

2.1.1. User-based collabarative filtering algorithm

In [1], Collaborative filtering is defined purely in terms of user-user similarity measures. The pure collaborative filtering based on user-user similarity measure, which make two basic assumptions. The first is that the users with a common interest will have similar item preferences and second that the users with similar preferences share the same interest. For example, two friends with similar preferences for books are very likely to read what the other has already read. In this approach we infer an individual's interest based on other similar users. The actual content of the items these similar users prefer does not have any significance. However, this approach requires a large amount of user preferences data to be available. However, for the pure CF algorithm, there are some well-known drawbacks:

1. CF recommenders perform badly when the user has an atypical taste, because there are no many neighbours near the user. It is known as gray sheep problem.

2. CF recommenders tend to recommend users with the most popular music items, and ignore artists with relatively lower popularity. This is because the nature of teh input of the CF algorithm totally relies on the subject feedback from the users, like rating or implicit responses like download or listening times. This would cause another problem called

popularity bias, the possibility of recommending unknown items in the long tails will be very low, which is bad for a novel artist.

3. “Cold start” problem, which appears for both elements of a recommender: users and items, make new users and new items very hard to categorise and to be recommended. It is the typical problem for CF recommender. The paper only mentioned one method to solve the cold start problem by using the implicit data of users. But there are various techniques have been developed.

2.1.2. Item-based ollabarative filtering algorithm

The item-based filtering is defined as a separate branch from collaborative filtering. Item based filtering is to learn the kind of content a user likes. To this user the system recommends the item that has high similarity with a item prototype that the user likes. The idea is to inspect the content of the item in the global item set and compare it to the active user's preferences without any consideration of other users and their similarity to the active user. This is usually based on the existing user-item preferences or ratings. In this method, Items that tend to be rated the same by similar users will be classed as similar under this approach. Once we have these similarities, we can represent a user in terms of the items they have interacted with and fnd items that are similar to these known items, which we can then recommend to the user. Again, a set of items similar to the known items is used to generate a combined score to estimate for an unknown item.

2.2. Hybrid music recommendation based on social tags

To solve the problem of applying contain-based caused by the limited content that each music item has, some groups start to mine the data like social tags. They are freely chosen keywords without the predefined vocabulary. So, social tags are an effective way to explain the feeling brought by a particular music track. It also offer us a valuable dataset to understand and categorise music tracks. Ji proposed a method to add the dimension of tags into the recommend system [2].

Instead of user-item matrix, three matrices describing the relations of user-tags, item-tags, user-item which is a binary matrix denoting whether the user has tagger on the specific item, were introduced into the framework. Furtherly, Levy applied latent semantic analysis and successfully reduced the noise in the dataset that collected from

last.fm data source [3]. Although social-tags provide us a solution for adding more content information to measure the similarity between items and users, which could be helpful to solve the cold start problem in a typical common CF-based recommender. It is still helpless to those items in the long tail region. Because tags are also concentrated in those highly popular music tracks. Therefore, it cannot promote the recommendation chance of those unknow items. Moreover, the highly repeatedly used word in the tag make the sparsity of matrix more serious.

2.3. Matrix factorisation

Since Spark's recommendation models currently only include an implementation of matrix factorization, we will focus our attention on this class of models. Data sparsity is an inherent property of the dataset. The low coverage of the interaction between users and items require us to use dimension reduction techniques to solve the problem, which are knowns as matrix factorisation. Matrix factorisation technique. such as singular value decomposition (SVD), Non-negative matrix factorization (NMF), or principal component analysis(PCA) are useful when the user-item matrix is sparse, which is very common in most recommender system. The main idea of these technique is to reduce the dimensionality of the user matrix by generating two result matrix U and V. For instance, SVD method computes matrices (n × k) U and (k × m) V for a given number of dimensions k. one for users of size n × k and one for items of size k × m. These are known as factor matrices. If we multiply these two factor matrices, we would reconstruct an approximate version of the original ratings matrix. Note that while the original ratings matrix is typically very sparse, each factor matrix is dense, as shown in the following diagram:

According to Yehuda and Rober's review focusing on matrix factorisation techniques [4], the prediction of those missing ratings values in the user-item matrix can be summarized as the process to solve the . Briefly, the result U matrix stands for a dimension-reduced user matrix, in which a group of vectors representing each user' music preference were compressed into a k-dimensional space composed by completely uninterpretable dimensions. Similarly, music items in V matrix, as a group of vectors, are scattered into the same k-dimensional space after the process of UV decomposition. By given U and V matrix with features of each user and music item, we can rebuild the user-item matrix and predict the user' preferences by calculating the dot product of the particular user and item vector. Secondly, it discusses many typical optimization methods for tuning the performance of the learning algorithm. including the technique like the regularization, bias control and temporal dynamic modeling.

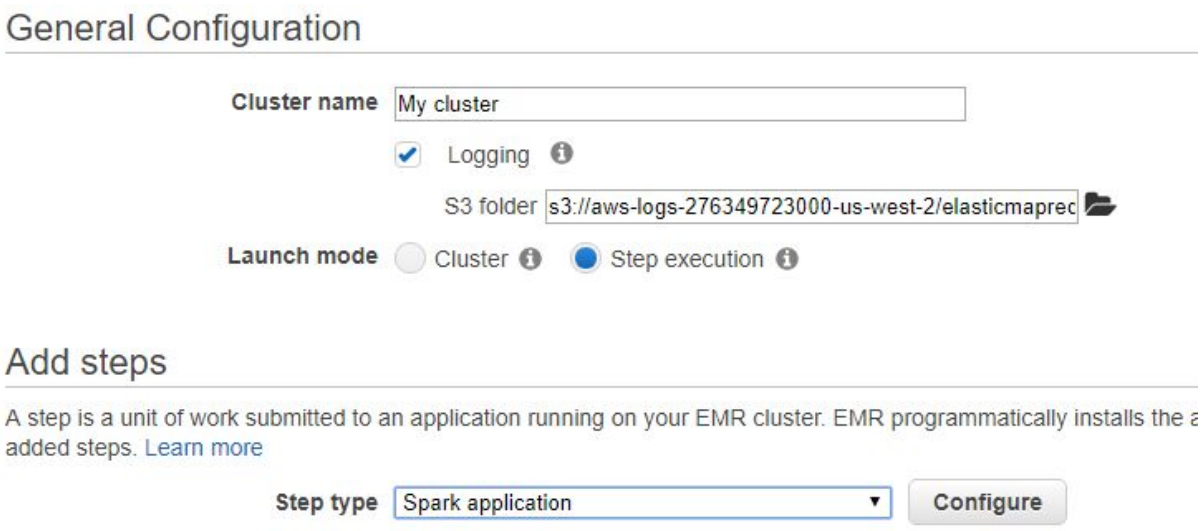
2.4. Alternating least squares (ALS)

Alternating Least Squares (ALS) is an optimization technique to solve matrix factorization problems; this technique is powerful, achieves good performance, and has proven to be relatively easy to implement in a parallel fashion. Hence, it is well suited for platforms such as Spark. ALS works by iteratively solving a series of least squares regression problems. In each iteration, one of the user- or item-factor matrices is treated as fixed, while the other one is updated using the fixed factor and the rating data. Then, the factor matrix that was solved for is, in turn, treated as fixed, while the other one is updated. This process continues until the model has converged [5].

3. System Setup for project development

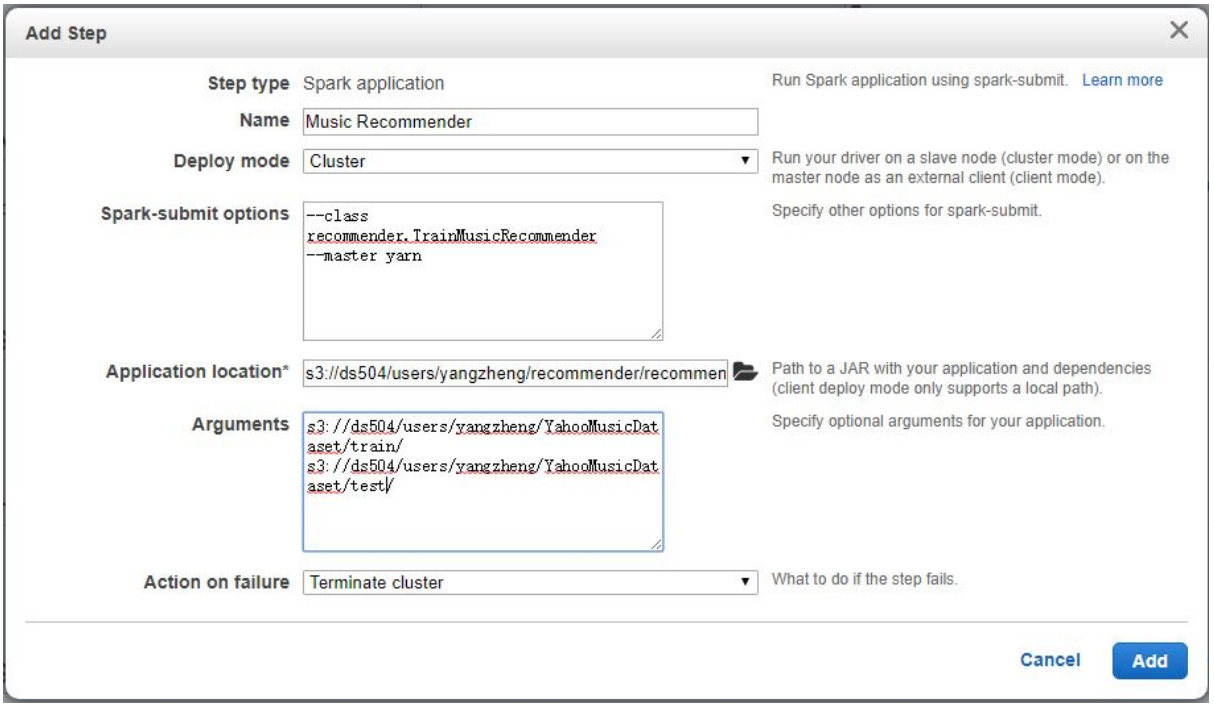
We develop this project as a maven project in Eclipse. So, in order to comile and package the code into jar file. You have to import it into Eclipse as an maven project. We use Amazon AWS to train our model. And then save the model in S3 file system and downloaded it to our local machine to do the recommendation. In this section I will describe how to run the spark application on AWS to train the model.

1) Use eclipse with maven plugin to compile and package the source code into a jar file.
2) Upload the jar file to Amazon S3.
3).Go to EMR and click “Create Cluster”, you will see the following screen



4) Set the Lunch mode in the General Configuration section as step excecution
5) Set the step type as Spark Application

6) Click Configure button beside the Step tye, you will see the following



7) Set the Deploy mode as Cluster

8) Set the Spark-submit options to be:

--class recommender.TrainMusicRecommender

--master yarn

9) Set Application location as the path to your jar file

10) Set Arguments as

<the path to the training data>

<the path to the testing data>

11) Click Add button

12) The click create cluster

The trained model will be output to:

s3://ds504/users/yangzheng/recommenderModel

Then you can download that model to do recommendation on your local machine

4. Manual for users

In this section, I will describe how to use the downloaded model to predict the recommendation.

Place the downloaded model folder “recommenderModel”(DO NOT CHANGE THE NAME) in the root folder of the project. Run the MusicRecommender class in Eclipse. In the Eclipse console you can see the following information:

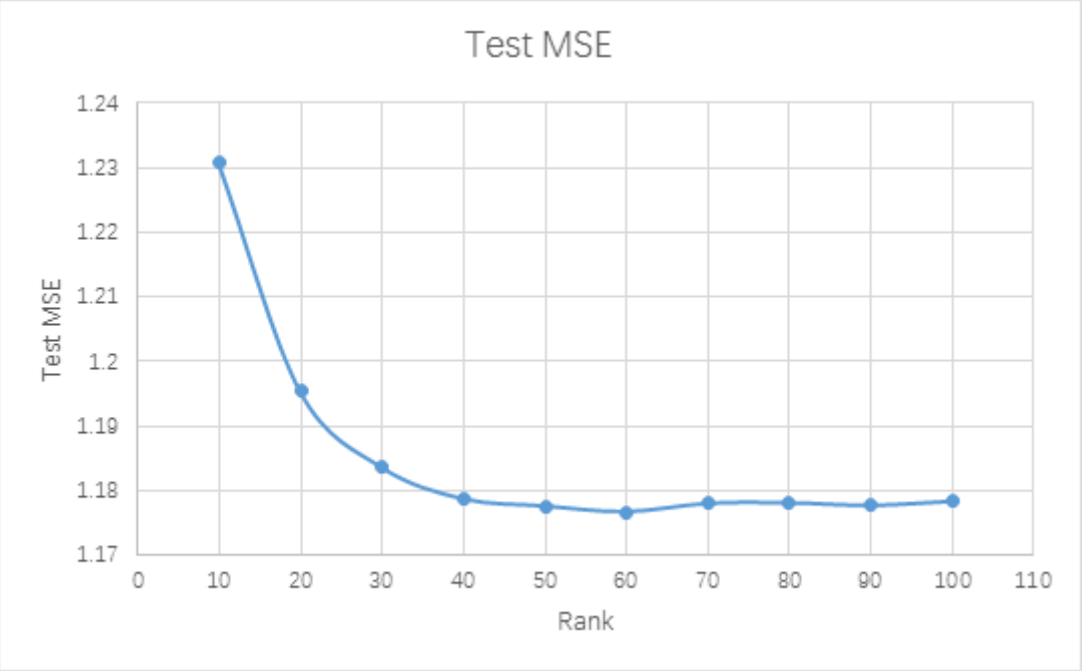
```
===Top 20 similar songs for Song 113 with corresponding ID and cosin
Song ID: 113    Cosine Similarity: 1.000000
Song ID: 78370  Cosine Similarity: 0.960862
Song ID: 131425 Cosine Similarity: 0.960394
Song ID: 129318 Cosine Similarity: 0.956410
Song ID: 117681 Cosine Similarity: 0.956109
Song ID: 62898  Cosine Similarity: 0.955344
Song ID: 108396 Cosine Similarity: 0.953851
Song ID: 5019   Cosine Similarity: 0.950176
Song ID: 57452  Cosine Similarity: 0.948416
Song ID: 19523  Cosine Similarity: 0.946241
Song ID: 102357 Cosine Similarity: 0.945880
Song ID: 113541 Cosine Similarity: 0.943659
Song ID: 132751 Cosine Similarity: 0.942756
Song ID: 116909 Cosine Similarity: 0.942532
Song ID: 11052  Cosine Similarity: 0.941974
Song ID: 6254   Cosine Similarity: 0.940187
Song ID: 28144  Cosine Similarity: 0.939146
Song ID: 129874 Cosine Similarity: 0.938081
Song ID: 50468  Cosine Similarity: 0.937952
Song ID: 128465 Cosine Similarity: 0.936405

===Top 10 songs recommended to user 200013===
Song ID: 38708
Song ID: 76577
Song ID: 107314
Song ID: 133077
Song ID: 61178
Song ID: 22514
Song ID: 10041
Song ID: 104688
Song ID: 86115
Song ID: 14902
```

The trained recommender model for small dataset is provided in the source code we submit. .

5. Results

5.1. Parameter tuning

In order to train the best model, we have to decide the best rank parameter which is the number latent factors. We tried different rank values on a small validation dataset. Below is a plot of different rank values vs. the corresponding test MSE.



The lowest test MSE comes when the rank value is 60.

5.2. Recommendation result based on user-user sililarity

Given a user id we can recommend songs to that user. Below is the recommendation to user with the ID 200115

```
===Top 10 songs recommended to user 200115==
Song ID: 90025
Song ID: 98520
Song ID: 108307
Song ID: 81173
Song ID: 22282
Song ID: 126780
Song ID: 61007
Song ID: 2253
Song ID: 2569
Song ID: 131928
```

These recommendation is based on a small training dataset about 1 GB, because the recommender model by the full training data is still under training on Amazon AWS by the time the report is written.

5.3. Recommendation result based on item-item silimilarity

Given a song we can find similar songs to it based on their cosine similarity. Below is the top 20 similar songs to the song with song ID 368 and corresponding cosine similarity.

```
===Top 20 similar songs for Song 368 with corresponding ID and cosine similarity===
Song ID: 368    Cosine Similarity: 1.000000
Song ID: 59400  Cosine Similarity: 0.923172
Song ID: 74121  Cosine Similarity: 0.922918
Song ID: 126460 Cosine Similarity: 0.920683
Song ID: 44065  Cosine Similarity: 0.918735
Song ID: 131398 Cosine Similarity: 0.912423
Song ID: 36874  Cosine Similarity: 0.911555
Song ID: 5659   Cosine Similarity: 0.909928
Song ID: 29030  Cosine Similarity: 0.904657
Song ID: 44725  Cosine Similarity: 0.903740
Song ID: 59777  Cosine Similarity: 0.903676
Song ID: 132570 Cosine Similarity: 0.899098
Song ID: 4341   Cosine Similarity: 0.898776
Song ID: 133392 Cosine Similarity: 0.897825
Song ID: 66270  Cosine Similarity: 0.896001
Song ID: 69481  Cosine Similarity: 0.895994
Song ID: 64617  Cosine Similarity: 0.895892
Song ID: 31463  Cosine Similarity: 0.895857
Song ID: 86721  Cosine Similarity: 0.895849
Song ID: 26057  Cosine Similarity: 0.895382
```

6. Challenges and lessons learned

6.1. Evaluating of recommendation models besides MSE

During the project proceeding, we met the challenges that how to evaluate and compare the performance of our user-based and item-based CF recommendation model. Because the evaluation metrics for measuring of user-based CF model's predictive capability or accuracy is Mean Squared Error (MSE). It is a direct measure of the reconstruction error of the user-item rating matrix, while the recommendation results in item-based CF model are totally based on the similarity in V matrix which described the factor features of items and unrelated with the resulted U matrix which described the factor features of users.

6.2. Mean average precision at K

Mean average precision at K (MAPK) is the mean of the average precision at K metric across all instances in the dataset. APK is a metric commonly used in information retrieval. APK is a measure of the average relevance scores of a set of the top-K documents presented in response to a query. For each query instance, MAPK will compare the set of top-K results with the set of actual relevant documents. However, in this project, we only demonstrated the MAPK comparison results on particular user points instead of on the whole dataset, because the algorithm time cost for calculating MAPK for m users, n items and k top recommendation items are O(mnk) which is very expensive.

6.3. Inspecting the recommendation result and similar items

Another challenge we met in the project is how to inspecting the recommendation items, because the dataset only contains music track IDs instead of music name. So we have to use other data source with explicit meaning, for example, MovieLens dataset with movie names, as our input of our script for building the models and to help us to inspect whether the model can function right and return the correct similar items.

References

[1] Resnick P, Varian H R. Recommender systems[J]. Communications of the ACM, 1997, 40(3): 56-58.

[2] Ji A T, Yeon C, Kim H N, et al. Collaborative tagging in recommender systems[J]. AI 2007: Advances in Artificial Intelligence, 2007: 377-386.

[3] Levy M, Sandler M. A semantic space for music derived from social tags[J]. Austrian Computer Society, 2007, 1: 12.

[4] Koren Y, Bell R, Volinsky C. Matrix factorization techniques for recommender systems[J]. Computer, 2009, 42(8).

[5] Pentreath N. Machine Learning with Spark[M]. Packt Publishing Ltd, 2015.