Music Recommendation System

Group Name: RUC

Group Members: Tongpeng Zhang(tz136),Ao Guo(ag1311),Junjie Feng(jf736)

1.Abstract

With rapid growth online music markets,automatic music recommendation has become an increasingly relevant problem in recent years, since a lot of music is now sold and consumed digitally. In most current approaches, content-based recommendation methods play an important role. Indeed, the basic process performed by a content-based recommender consists in matching up the attributes of a user profile in which preferences and interests are stored, with the attributes of a content object (item), in order to recommend to the user new interesting items.However, people have their own unique tastes in music. This report proposes a method to calculate a personalized distance measure to recommend user based on the counts of music. Thus, we use another method called Collaborative filtering recommendation, and relying on ALS algorithm of spark. Meanwhile we optimize some method in the algorithm to reduce time.

We also provide a parallel implementation of the accelerated ALS algorithm in the Apache Spark distributed data processing environment, and an efficient line search technique as part of the ALS implementation that requires only one pass over the data on distributed datasets. And we compare our new algorithm with the original one, the acceleration factor increases as greater numerical precision is required in the solution.

2.Introduction

In recent years, the music industry has shifted more and more towards digital distribution through online music stores and streaming services such as iTunes, Spotify, Grooveshark and Google Play. As a result, automatic music recommendation has become an increasingly relevant problem: it allows listeners to discover new music that matches their tastes, and enables online music stores to target their wares to the right audience.

The abundance of information available on the Web and in Digital Libraries, in combination with their dynamic and heterogeneous nature, has determined a rapidly increasing difficulty in finding what we want when we need it and in a manner which best meets our requirements.

As a consequence, the role of user modeling and personalized information access is becoming crucial: users need a personalized support in sifting through large amounts of available information, according to their interests and tastes.

Many recommender systems rely on usage patterns: the combinations of items that users have consumed or rated provide information about the user's preference, and how the items relate to each other. This is the collaborative filtering approach. Another approach is to predict user preferences from item content and metadata.

2.1 Content-based music recommendation

Music can be recommended based on available metadata: information such as the artist, album and year of release is usually known. Unfortunately this will lead to predictable recommendations. For example, recommending songs by artists that the user is known to enjoy is not particularly useful.

2.2 Collaborative filtering

Collaborative filtering methods can be neighborhood-based or model-based. The former methods rely on a similarity measure between users or items: they recommend items consumed by other users with similar preferences, or similar items to the ones that the user has already consumed. Model based methods on the other hand attempt to model latent characteristics of the users and items, which are usually represented as vectors of latent factors. Latent factor models have been very popular ever since their effectiveness was demonstrated for movie recommendation in the Netflix Prize.

2.3 Dataset

we get this data package from open sources website. The whole file includes 141000 users,1.6 millions artists, and 24.2 millions music played records.

Our report is organised as follows. First, we present the differences between our project with other similar music recommendation methods in Section 3. Second, we introduce what we have done, our contributions for our algorithm, and project in Section 4. Third, we give details what our contribution is, and how we implemented them, how we thinked about them in Section 5. Fourth, we show the results & analysis of our optimized method, meanwhile, we also compared it with the original one in Section 6. Last, conclusions are drawn in Section 7.

3.A previous works section

Most current approaches, content-based recommendation methods play an important role. Indeed, the basic process performed by a content-based recommender consists in matching up the attributes of a user profile in which preferences and interests are stored, with the attributes of a content object (item), in order to recommend to the user new interesting items.However, people have their own unique tastes in music. We change that to using ALS algorithm of spark for Collaborative filtering recommendation method.Previous work needs get the exact rating of the music, Our algorithm only needs get how many times that user have listened of the artists. And another difference is, previous work input the data with the whole file, and our system optimize this, our algorithm also accessed to whole data file under MAX_Integer, but system will split data to small size, and do parallel computation. To one same artist who has different ID, previous work Solves independently, because of this, the precision of system is bad. Our algorithm will check all same artist with different ID, and put to a new same ID. We also choose implicit feedback instead of explicit feedback.

4. Contribution section

Tongpeng Zhang:

What I did for our project, first, I read a bunch of recommendation algorithms, and compared these methods to find the fittest one to be the basic algorithm of our project. Found collaborative filtering recommendation is the better one, this method could use other user's information who has common preference with client to do recommendation, we don't need to analyze (index) content, can capture more subtle things. I also decided to use implicit feedback instead of explicit feedback.

Ao Guo:

In this project, my contribution is implement the detailed algorithm and try to optimize it. The first thing is to figure out the philosophy of Alternative Least Square algorithm and dig out the low-level implementation in Spark mllib. Then I set up Spark running environment with building source code and integrate necessary depended libraries in Intellij IDEA IDE. To get a better performance and recommendation accuracy, it is important to build the program in a good manner, tune all the relevant parameters and cache intermediate RDDs when necessary. The optimization get good performance improvement in the end.

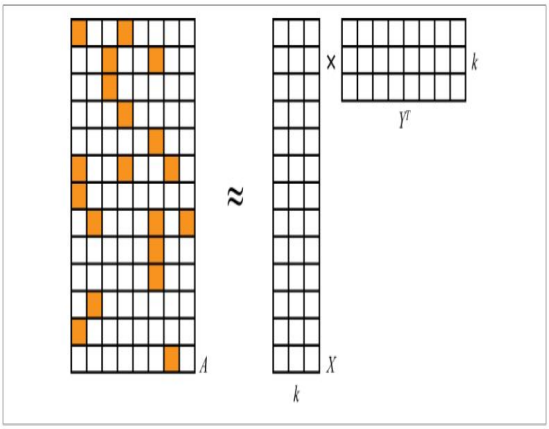
Junjie Feng:

My contribution to our project is mainly on the clustering the artist alias IDs into one unique ID in order to improve the recommendation precision. First, I find the raw dataset from the Audioscrobbler public data, and study on it and find out the relationship between each file.

I also make the test on different ALS model parameters and compare all the result. Finally. I find out the best set of ALS parameters, which could get the best recommendation precision.

5.An implementation details section

Audioscrobbler received around 2 million song submissions per day. This data set contains profiles for around 150,000 real people. The dataset lists the artists each person listens to, and a counter indicating how many times each user played each artist.



Alternative Least Square (ALS) is an algorithm which turn one matrix into the product of two smaller matrices. Different from gradient descending, the algorithm fix one of the matrix as known coefficient and modify the other. This process will be done alternatively every iteration. The error is calculated using Root-Mean-Square Error:

$$RMSE = \sqrt{\frac{1}{n}\sum_{u,v}|(p_{u,v} - r_{u,v})^2|}$$

In this way the loss function becomes:

$$(u_i, v_j) = \min_{u,v} \sum_{(i,j)\in K} (p_{i,j} - r_{i,j})^2 + \lambda(||u_i||^2 + ||v_j||^2)$$

We will get the final approximate matrix when loss function is optimized.

5.1 Tuning the training parameters

Spark's mllib has its built-in ALS algorithm API "*ALS.trainImplicit()*"to do recommendation:

```
def trainImplicit(ratings: RDD[Rating], rank: Int,
  iterations: Int, lambda: Double, alpha: Double)
    : MatrixFactorizationModel = {
    trainImplicit(ratings, rank, iterations, lambda, -1, alpha)
  }
```

In this function, there are five parameters involved:

- RDD[Rating]
- rank
- Iterations
- lambda
- alpha

RDD[Rating] is the dataset that we input. The data has been preprocessed to improve recommendation accuracy, which we will mention in the following.

Rank represents the rank value of the decomposed two matrices. As we know, the rating of each user on each song is determined by a lot of factors, which become the latent variables in ALS. To be more specific, the rating of a song is affected by a lot of factors such as the music style, length, languages and any other related features. However, we only got single general rating of each user on each listened song, that means we need to tune the rank parameter, make a good assumption on the number of latent variables. We need to know that higher rank will

bring bigger decomposed matrices, and bring more computation cost.

Iteration means the number of times we train our model. Traditionally we may calculate the error in each iteration and terminate the training when error reaches the target that acceptable. However, in Spark ALS they use a parameter to indicate the termination condition under which we stop the training. More iterations may bring a better recommendation but it can also bring more computation cost.

Lambda is the coefficient of regulation term. The reason why we need this term is that we need some mechanism to do penalty to the MSRE. In this way we can adjust the weight of the model (previous term) so that we can get rid of overfitting. The recommended value of lambda is 0.01.

Alpha determines the base line of our acceptance. Originally the dataset is a sparse matrix with plenty of zeros in it. As training goes, we will get a denser joined matrix with many zeros replaced by value that closer to 0. We need to set a base line of confidence to determine whether the rating is strong enough.

There is also a parameter -1, which implies the number of block reflecting the parallelism. Due to the limitation of PC we failed to test it. -1 is default value.

5.2 Data preprocessing that improves accuracy

According to our analysis on dataset, single artist may have multiple aliases due to the data quality. This situation greatly affects the final output of recommendation. In another word, the rating would be dispersed into different "items" that supposed to be entirety. Luckily, we have another file indicating this mapping between artists and their multiple aliases:

```
val trainData = rawUserArtistData.map{
  line =>
    val Array(userId, artistId, count) = line.split(' ').map(_.toInt)
    val finalArtistID = bArtistAlias.value.getOrElse(artistId, artistId)
    Rating(userId, finalArtistID, count)
}.cache() // (*)
```

Above is how we combine all the aliases belonged to single artist and become an entry. This becomes the final training dataset we use.

5.3 Caching the frequent intermediate data

As the same code fragment shows above, we can easily see that the final training dataset will be access for many time to calculate the error and compute when alternative least square calculated. In this case, we cache the processed RDD (here is the training data matrix) in memory so that we can avoid data transferring every iteration. That brings great improvement to the performance and we will analyse it in the following sections.

6.A results section with analysis

We have test our recommendation system on the Audioscrobbler's dataset. We download this dataset

from its webpage. The dataset includes more than 24 million playback records from 2002 to 2005. It contains approximately 141,000 unique users and 1.6 million unique artists. Of course, each artist is recorded with ID. Note that the same artist may correspond to many different names that have different ID.

6.1 Clustering the artist ID

In the raw dataset, each line of playback records contains one user ID, one artist alias ID and the

number of playing count which represents how many times this user has listened to this artist's music pieces. However, many musicians have several alias names, and each alias name corresponds to an artist alias ID. This is a common case when a musician has published multiple albums or has cooperated with other musicians.

In our recommendation system, we make use an additional dataset. In this dataset, each line contains one alias name ID and one unique ID of that musician. Before starting the ALS training model, we cluster the several artist alias ID into one unique artist ID. The play count of the new pair (user ID, unique artist ID), is the sum of the play count of each pair (user ID, alias ID).

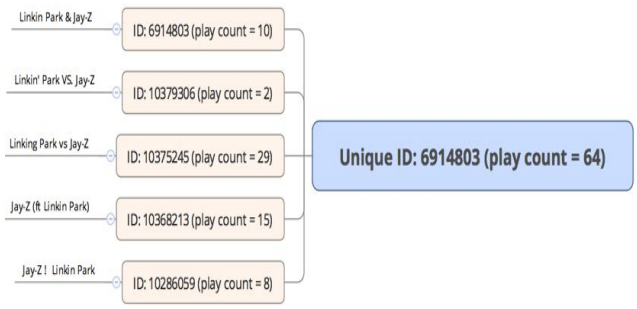


figure 6.1

Figure 6.1 shows that, Jay-Z and Linkin Park has collaborated to publish one or several albums. There are five different alias ID and alias name. For user (ID=209732), he has listened to these artists for different times. Finally, in our ALS input training data, there will be only one unique artist ID (6914803), which represents the the combination of Jay-Z and Linkin Park, and only one pair (user ID, artist ID, play count).

```
Without merger,rank=10,λ=0.012,iteration=5
(2093760,1300642,0.026266728)
(2093760,1308254,0.023869438)
(2093760,1114670,0.022670428)
(2093760,1046450,0.022248158)
```

figure 6.2

```
With merge,rank=10,λ=0.012,iteration=5
(2093760,1300642,0.026453131)
(2093760,2814,0.0262199577)
(2093760,1037970,0.025403633)
(2093760,1001819,0.0251077078)
```

figure 6.3

```
Without merge,rank=12,λ=0.012,iteration=5
(2093760,2814,0.0332615202)
(2093760,1811,0.0333170304)
(2093760,1001819,0.0334348967)
(2093760,1300642,0.0344533654)
```

figure 6.4

```
Without merge,rank=10,λ=0.008,iteration=5
(2093760,1300642,0.028083230)
(2093760,1811,0.02631811)
(2093760,1114670,0.027046450)
(2093760,1046450,0.026856572)
```

figure 6.5

```
With merge,rank=10,λ=0.010,iteration=5
(2093760,1001819,0.031332741)
(2093760,2814,0.0311111766)
(2093760,4605,0.0307145103)
(2093760,1811,0.0301325664)
```

figure 6.6

```
With merge,rank=10,λ=0.012,iteration=10
(2093760,1015537,0.028112695)
(2093760,1244723,0.027193579)
(2093760,5209,0.027075033716)
(2093760,5209,0.027075033716)
```

figure 6.7

```
With merger,rank=12,λ=0.010,iteration=10
(2093760,1300642,0.036339613)
(2093760,2814,0.0351139756)
(2093760,1811,0.0331339797)
(2093760,4605,0.0337264865)
```

figure 6.8

Figure 6.2 and figure 6.3 shows the comparison of the recommendation result of user (ID=2093760). Each pair of data represents the recommended artist and the preference rating of this artist. The result only contains the top four preference rating artists for the specific user. As the result shows, after cluster the alias artist IDs into one unique artist ID, the preference rating grows significantly.

For the artist ID 1300642, it corresponds to 3 alias IDs. The rating grows a little because our system could only merge 10 clusters into 1 cluster.

For the rest artist IDs (2814, 1037970, 1001819), they are even not being recommended to the user before. There are two potential reasons for this result.

First, we searched the raw dataset and found out that these artist IDs each has more than 20 alias IDs. After the merger, several playback records clustered into one records. The total play counts in the new pairs (user ID, artist ID, play count) are changed to a much bigger value. As we explained in section 4, the more times the musician has been selected, the higher probability that it may be recommended to other users who has not listen to this musicians' pieces.

Second, if the previous recommended musician has an alias ID 1308254, and the user (ID=2093760) has listened to his pieces which has another alias ID (1308255). After the merger, the system will consider the user has listened to this musician's pieces before, so the system will not recommend this musician anymore. That's a potential reason that why some previously recommended musicians are not recommended after merger.

6.2 Adjusting the ALS model parameters

The core algorithm in our recommendation system is ALS algorithm, which is used to decompose the utility matrix into two long and thin matrices. As we explained before, we called the ALS function in spark's machine learning library. For the implicit training model, there are five parameters in the ALS model, which are rank, iterations, lambda, blocks and alpha. We test the recommendation precision with different parameter values.

6.2.1 Rank Number

The rank number not only represents the number of implicit feature to use, but also represents the size of two low-rank matrices, which are decomposed from utility matrix. Assume that we have X users and Y artists, then the size of utility matrix is X*Y. The size of two low-rank matrices are X*rank and rank*Y.

Compare the result showing in figure 6.2 and figure 6.4, the higher the rank, the better the recommendation precision we get. The bigger the rank number, the more the implicit features we use. From the system view, the system could evaluate the users' taste and artist's characteristic from more angles. So when we run the system with rank number 12, we get the result with much better recommendation precision than the run the system with rank number 10 does.

6.2.2 Regularization Factor

The λ parameter is the regularization factor. We test with several values and then find out that. With λ equals to 0.01, we could get the best result at all.

The reason why we get these result is because when the λ grows bigger or smaller, the ALS model would overfitting the data. Both overfitting and underfitting can lead to poor model performance. But by far the most common problem in applied machine learning is overfitting. When the system overfitting the train data, we could get good performance on the training data, but poor generliazation to other data. When the λ equals to 0.01, we could get the result with best recommendation precision. As the figure 6.6 shows, when the λ equals to 0.01, we could get the result with best recommendation precision.

6.2.3 Iteration Number

By run the ALS training model iteratively, we could get more precise low-rank matrices, and more precise recommendation.

As the figure 6.7 shows, the more times the ALS runs, the better the performance we could get. However, as the trade-off for better performance, the running time also grow linearly with the number of iteration.

6.3 Cache the data

Section 6.1 and 6.2 is what we achieved to improve the recommendation precision. In this section, we did the most

Since the training data will be accessed many times during the whole process, we decide to cache the training data in order to speedup the running time.

7.A conclusion

Our music recommendation system is a classic collaborative filter system, which could recommend musicians to user based on the similarity measure between users and musicians. Instead of using feature of musician to determine their similarity, we focus on the similarity of the musicians the individual user would like select.

In our program, first it would load the raw data as RDD structure in memory, then it would merge the records that includes the same (user ID, artist ID) pairs. Next, it would use ALS implicit training model to decompose the utility matrix and calculate the rating preference of each musician for the specific user. Finally, the top rating musicians would be recommended to the user.As the figure 6.8 shows, when we combine all the improvement together, the result of recommendation precision is best of all.

In the real life, users are unwilling to give the rating to every music they have listened. As a result, music platform or website could collect much less rating than playback records. Since the playback records are easy to collect, we could get a huge dataset to train our model.

8.References

Our project use the Audioscrobbler public dataset. Audioscrobbler is (http://www.last.fm/en/) the first music recommendation system. Http://www.last.fm/zh/ is the first web streaming audio site, founded in 2002. Audioscrobbler for "scrobbling" provides a development API, the main record listener listened to the writer's songs. The site uses this information to create a powerful amount of music recommendation system. This system reaches millions of users, because third-party App and Web sites can provide listening data to the recommendation engine.

Previously, we learned the example on scala official website shows how to use the scala machine learning ALS algorithm. Then we built our own program to do the recommendation.

[1]Overfitting and Underfitting With Machine Learning Algorithms: http://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

[2]Optimize the ALS algorithm in Spark: http://www.ithao123.cn/content-8762538.html

[3]Spark Document: https://zhidao.baidu.com/question/1756944151030325348.html

[4]Matrix Factorization Techniques for Recommender Systems, Journal, Computer, Volume 42 Issue 8, August 2009 ,Pages 30-37 , IEEE Computer Society Press.

Audioscrobbler Dataset: http://www-etud.iro.umontreal.ca/~bergstrj/audioscrobbler_data.html

[5]Spark official Site: http://spark.apache.org/docs/latest/quick-start.html

[6]Spark MLlib Collaborative filter: http://blog.javachen.com/2015/04/17/spark-mllib-collaborative-filtering.html

[7]Content-based Recommender Systems: State of the Art and Trends: http://facweb.cs.depaul.edu/mobasher/classes/ect584/Papers/ContentBasedRS.pdf

[8] Collaborative Filtering Recommender Systems: http://files.grouplens.org/papers/FnT%20CF%20Recsys%20Survey.pdf