Yahoo Music Recommendation System

Yuanjie Shi
Kaggle team:Alone

***Abstract*— This report is talking about how to recommend a new track to the user according to the training data, including all rating history of all users, track id, album id, artist id, genre id and all their affiliations.**

I. INTRODUCTION

Nowadays, music plays a very crucial part in our life. Some people literally eat, live and breathe music. But as we know the music data around us is very large. The number of songs available exceeds the listening capacity of single individual. People sometimes feel difficult to choose from millions of songs. Moreover, music service providers need an efficient way to manage songs and help their costumers to discover music by giving quality recommendation. People listen to a variety of songs on various website like Pandora, EMusic, Spotify, etc., and all of them have their recommendation system or in other way I can say they recommend songs to the user based on some parameters.

Well, that is an interesting thing. I tried to find out how only a few songs are selected and recommended from such a large data. So for this project, I designed a music recommendation system.

II. DATA PROCESSING

Yahoo Music Dataset is very sparse and contains many parameters like user id, track data, album,artist, genre and rating information. I need to process it first.

A. Data Set Description

The whole data set is published by Yahoo on the website. The music data is stored in hierarchy structure. Theres no user records for the tracks. The data set only contains the scores for the album, artist, type, etc. So I have to match the track which Ill consider to recommend and the album the artist it belongs to and I have to make our own rules to rate these tracks.

- trainIterm2.txt - the training set
- testIterm2.txt - the test set
- sample_submission.csv - a sample submission file in the correct format
- trackData2.txt - Track information
- albumData2.txt - Album information
- artistData2.txt - Artist listing
- genreData2.txt - Genre listing formatted as: $\langle 'GenreId' \rangle$

*This work is supported by Prof. Rensheng Wang

B. Data Integration

In the final project, I have 20,000 customers and six songs. From the given six tracks, I have to select three tracks labeled as "1" and recommend to each customer, and I label the remaining three songs as "0". Firstly, I try to integrate the data files, The first data file "testItem2.txt" shows the userID and the given six tracks, the second data file "trackData2.txt" shows the hierarchy structure of each track. From these data files, I designed Python file "Geth.py" to get hierarchy structure file "Ranking2.txt", the form of the content is shown as below: UserID—trackID—Album—Artist—Genre1—Genre2—. . .

For training data:

```
[[[199808, 35],
  [248969, 90],
  [2663, 90],
  [28341, 90],
  [42563, 90],
  [59092, 90],
  [64052, 90],
  [69022, 90],
  [77710, 90],
```

Fig. 1. Rating records: user id, number of records, track id, score

```
[56, 48507, 236393, 173467, 98154, 48505],
[57, 176764, 236794, 61215, 34486, 18161],
[58, 215407, 18147, 17453, 35389],
[59, 0, 0, 61215],
[60, 68134, 263882, 214765, 131552, 173467, 48505],
[61, 88463, 5232, 131552, 61215],
```

Fig. 2. Track id, album id, artist id, genre id

For testing data:

```
[[199810, 6, 208019, 74139, 9903, 242681, 18515, 105760],
 [199812, 6, 276940, 142408, 130023, 29189, 223706, 211361],
 [199813, 6, 188441, 20968, 21571, 79640, 184173, 111874],
 [199814, 6, 122375, 189043, 122429, 52519, 232332, 262193],
 [199815, 6, 64345, 118841, 275682, 30062, 258473, 129866],
 [199816, 6, 274758, 102153, 183464, 23616, 81699, 46627],
```

Fig. 3. User id, number of tracks, track id

III. METHODS IMPLEMENTATION

Using what I have learned from the EE627 course, I try to define my own scoring rule and assign different weights to album scores, artist scores and genre scores so I can get our first group of scores. I also tried the matrix factorization and ensemble all the solutions I previously got to calculate the coefficients which I can use to get the true result.

A. Matrix Factorization

I can construct a sparse matrix to represent the rating history first, then through Matrix Factorization to find the user-item matrix and item-track matrix. The values I get in new user-track matrix corresponding to the empty values in original matrix are what I want for the recommender system.

Its hard to construct a huge sparse matrix using normal package, much less sparse matrix computation. I choose the ALS (alternating least squares) model in Spark to solve it. For this part, I only use rating records history for training, every track, album, artist and genre is treated as a product and every raring record is treated as a user-product pair. Then I use embedded predict method to work out the empty user-product (testing data) pair I want. But with rank=15, number of iterations =20, I can only get a score 0.67749.

Code (intact code for this part is in ALS_modle.ipynb ALS_model_predictor.ipynb):

```
rank = 15
numIterations = 20
model = ALS.train(ratings, rank, numIterations)
```

Fig. 4. Matrix Factorization code part 1

```
ratings = data.map(lambda l: l.split(' ')).map(lambda l: Rating(l[0], l[1], l[2]))
ratings.take(10)
```

```
[Rating(user=199808, product=248969, rating=90.0),
 Rating(user=199808, product=2663, rating=90.0),
 Rating(user=199808, product=28341, rating=90.0),
 Rating(user=199808, product=42563, rating=90.0),
 Rating(user=199808, product=59092, rating=90.0),
 Rating(user=199808, product=64052, rating=90.0),
 Rating(user=199808, product=69022, rating=90.0),
 Rating(user=199808, product=77710, rating=90.0),
 Rating(user=199808, product=79500, rating=90.0),
 Rating(user=199808, product=82317, rating=90.0)]
```

Fig. 5. Matrix Factorization code part 2

```
predictions = model.predictAll(testdata).map(lambda r: ((r[0], r[1]), r[2]))
predictions.count()
```

```
119974
```

```
ratesAndPreds = ratings_test.map(lambda r: ((r[0], r[1]), r[2])).join(predictions)
ratesAndPreds.take(10)
```

```
[((199813, 184173), (0.0, 52.959590916119375)),
 ((199814, 232332), (0.0, 87.70558097654228)),
 ((199815, 118841), (0.0, 34.04502134605689)),
 ((199816, 23616), (0.0, 56.55580089840702)),
 ((199818, 283698), (0.0, 34.40088700464496)),
 ((199819, 36793), (0.0, 94.22779566674552)),
 ((199819, 247549), (0.0, -60.7849861300207)),
 ((199823, 170625), (0.0, 67.76466104885257)),
 ((199823, 254625), (0.0, 15.354385060092135)),
 ((199826, 179298), (0.0, 121.11961734750045))]
```

Fig. 6. Matrix Factorization code part 3

B. Ensemble Learning

Through math derivation, I can merge several solutions for this task, getting a better result. I just need to work out the weights for each submission and combine all of them to get the final score for the recommender. The score is 0.81273.

Code (intact code for this part is in model_ensemble.ipynb):

Weight and generating new rating list:

```
# score for two model
c_1 = 0.81183
c_2 = 0.68225
c_mat = np.transpose(mat([c_1,c_2]))
c_mat
```

```
matrix([[0.81183],
        [0.68225]])
```

```
# ST * y
k = 120000
STy = k* (2*c_mat - 1)
STy
```

```
matrix([[74839.2],
        [43740. ]])
```

```
STS = np.transpose(S) * S
STS_inv = STS.I
STS_inv
```

```
matrix([[ 9.04444524e-06, -2.53606245e-06],
        [-2.53606245e-06,  9.04444524e-06]])
```

Fig. 7. Ensemble Learning code part 1

```
W = STS_inv * STy
W
```

```
matrix([[0.56595168],
        [0.20580715]])
```

```
new_ratings = S * W
new_ratings
```

```
matrix([[-0.77175883],
        [ 0.36014452],
        [-0.36014452],
        ...,
        [ 0.36014452],
        [-0.36014452],
        [ 0.77175883]])
```

Fig. 8. Ensemble Learning code part 2

C. Adjust the weight of each parts scores

In my last method, among the score for albums, artists, genres, I try to figure out which part is more important for a user to decide if he or her will try a new track.

For Preliminary Tuning, I assign the weight 0.9 for the album score firstly, 0.7 for the artist and 0.4 for the genre score. I get the accuracy for 68%. for the first solution so I must change the weight assignment. I continue to decrease the weight of the album scores and the artist scores. While I try the weight 1, 0.01, 0.00007 for each part, I get the accuracy for 0.865 and make a huge progress. However, I can not make any progress after tuning the weights.

So, to make the fair rule, I reconsider the whole definition of the weights. I find that I can not give a track a better grade just because it has more genres than other songs and I should make the rule fair to the tracks which dont have any genre. So while I compute the genre scores, I take the average values and I calculate the average score again after getting the final scores. As for the tracks which dont have genre, I just take the average value of the album score and artist score. Therefore, the rule has become more reasonable and I get the score for 0.87129.

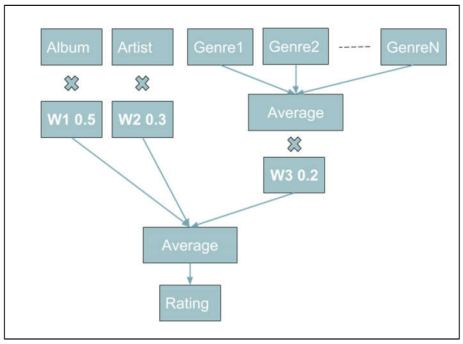

Fig. 9. Album Artist and Genre with Weight Algorithm

IV. Results

After applying all the methods I learned from the course, I get different results as follows:

TABLE I
RESULTS OF EACH METHODS

Method	Score
Matrix Factorization	0.67749
Ensemble Learning	0.81273
Tuning Weights	0.87129

So based on our implementation, I find that Matrix Factorization can help me get more scores, however, it can not help with getting better accuracy. In Ensemble Learning, some of my solutions are really similar results so the accuracy just stays the same. In the last method, which matters most is the album scores and the genres scores take only a little weight in the whole calculation.

V. Future Work

A. Matrix Factorization

Because sparse matrix factorization really consumes computational load, its hard to get the model with more rank (items between user and product) by only one regular laptop. I will try use several laptops for much more ranks on this task. And as shown above, there will be some missing values after prediction, I just fill zero for them for now. there should be some more accurate way to deal with them I will try in the future

B. Ensemble Learning

At present, I only tried combining two solutions and the accuracy has not improved observably. I will try different number and different solutions in the future to get more accurate result.

C. Tuning Weights

In future, Id like to get more new better solutions by tuning the weights and Ill ensemble it with our previous solutions to make better predictions.