# Assignment 4 Report

## G01145163 Hung Mao Chen

Outline:

Implement a hybrid system using ALS solution and item-item CF

ALS part in hybrid system

Recommended movie to me by ALS model

Item-item CF in hybrid system

The challenge for building up pivot table

Back to the item-item CF

Recommended movie to me by Hybrid Model

Conclusion

----------------------------------------------------------------------------------------------------------------

Implement a hybrid system using ALS solution and item-item CF

To implement a hybrid system, I built up CF with ALS and item-based CF as classes separately. The source file of hybrid system, which is also the entry of the source code, is in the file `/src/main.py`.

1. **from** als\_recommender **import** ALSRecommder
2. **from** itemCF **import** ItemCF

In the head of file, `ALSRecommender` represents ALS solution which can be found in the file `/src/als\_recommender.py`, and `ItemCF` represents item-based CF which can be found in file `/src/itemCF.py`.

ALS part in hybrid system

For the part of ALS solution, it can be separated to two parts: make cross-validation over 80% of the dataset to decide the tuned parameters, and the other part is to use the tuned ALS model to recommend some movies to me.

For the cross-validation part,

1. als\_recommender = ALSRecommder(spark, movies\_path, ratings\_path)
2. #als\_recommender.tune\_test([0.1], [x for x in range(6, 20, 2)])

I built up the ALS model with ALS class, here I would move to part of ALS class to explain the detail of implementation.

The major function provided in ALS class is `tune\_test`:

1. **def** tune\_test(self, regParam, ranks):
2. train, validation, test = self.ratings\_df.randomSplit([0.6, 0.2, 0.2])
3. min\_err = float('inf')
4. best\_rank = -1
5. best\_reg = 0
6. final\_model = None
7. **for** rank **in** ranks:
8. **for** reg **in** regParam:
9. als = self.model.setMaxIter(10).setRegParam(reg).setRank(rank)
10. model = als.fit(train)
11. predictions = model.transform(validation)
12. evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")
13. rmse = evaluator.evaluate(predictions)
14. **if** rmse < min\_err:
15. min\_err = rmse
16. best\_rank = rank
17. best\_reg = reg
18. final\_model = model
19. # get best model "final\_model"
20. **print**("Best model RMSE on validation set with rank: " + str(rank) + " is: " + str(rmse))
21. self.model = final\_model
22. # test model
23. preds = self.model.transform(test)
24. rmse\_eval = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")
25. mse\_eval = RegressionEvaluator(metricName="mse", labelCol="rating", predictionCol="prediction")
26. mae\_eval = RegressionEvaluator(metricName="mae", labelCol="rating", predictionCol="prediction")
28. rmse = rmse\_eval.evaluate(preds)
29. mse = mse\_eval.evaluate(preds)
31. with open(“/home/hchen28/tuned\_result.txt”, “a+”) as result:
32. **result.write**("Test set (RMSE/MSE/MAE): " + str(rmse) + "/" + str(mse))
33. result.close()

Most of all, I tested the rank from 6 to 20 with step of 2. In line 14-18, I compared the RMSE error to find the best parameter for the model. At the end, I found the best value of rank for the ALS model would be 16. With the code above, I computed RMSE and MSE on the test set with my best ALS model. The result can be found in `/output/tuned\_result.txt`.

Test set (RMSE/MSE): 0.8075382905785043/0.6521180907504529

Then I use this best ALS model trained by the Movie Lens dataset to try to make some recommendation for myself.

First, I created a utility matrix with userId 0 based on the Movie Lens 20M dataset.

1. # 9: Sudden Death (1995)
2. # 23: Assassins (1995)
3. # 66: Lawnmower Man 2: Beyond Cyberspace (1996)
4. # 145: Bad Boys (1995)
5. # 153: Batman Forever (1995)
6. my\_rated\_df = spark.createDataFrame(
7. [
8. (0, 9, 4),
9. (0, 23, 2),
10. (0, 66, 2),
11. (0, 145, 4),
12. (0, 153, 3)
13. ],
14. ['userId', 'movieId', 'rating']
15. )
16. new\_ratings\_df = ratings\_df.union(my\_rated\_df)

Personally, I make some ratings on these five movies: Sudden Death, Assassins, Lawnmower Man 2: Beyond Cyberspace, Bad Boys (Wow, I love the latest movie of Bad Boys so much), Batman Forever. Then, I merge my rating records into the original dataset of ratings.

I used the best model I got previously to predict rating on the new dataset.

1. model = als.fit(new\_ratings\_df)
2. userRecs = model.recommendForAllUsers(10)
3. # movieId recommended by ALS
4. # { movieId: ALS score }
5. als\_recommend\_list = {}
6. # recommended movies for me
7. **for** i **in** userRecs.collect():
8. **if** i['userId'] == 0:
9. with open(als\_output\_path, "a+") as output:
10. output.write("With ALS Recommendation: \n")
11. **for** j **in** i['recommendations']:
12. output.write("Recommend movieId: " + str(j[0]) + " with score: " + str(j[1]) + " to me.\n")
13. als\_recommend\_list[j[0]] = j[1]
14. output.close()

In the second line, `recommendForAllUsers()` helped me to recommend top 10 movies for each users. Then I filter my own userId to list the recommended movies. Then I can get the result.

Recommended movie to me by ALS model

With ALS Recommendation:

Recommend movieId: 5271 with score: 10.924333572387695 to me.

Recommend movieId: 74263 with score: 9.590706825256348 to me.

Recommend movieId: 71017 with score: 9.150797843933105 to me.

Recommend movieId: 2954 with score: 8.941307067871094 to me.

Recommend movieId: 86947 with score: 8.911588668823242 to me.

Recommend movieId: 104583 with score: 8.319062232971191 to me.

Recommend movieId: 50942 with score: 8.305917739868164 to me.

Recommend movieId: 48045 with score: 8.248882293701172 to me.

Recommend movieId: 32088 with score: 8.155112266540527 to me.

Recommend movieId: 27009 with score: 8.083059310913086 to me.

The result file also can be found in `/output/als\_movie\_recommendation.txt`. The results show movieId and the according scores. The higher the score is, I would be more likely to love the movie based on the ALS model.

This is the result with only ALS recommendation. Let’s take a look at the movie to see whether the result made sense or not.

|  |  |  |
| --- | --- | --- |
| Movieid: 5271 | 30 Years to Life (2001) | Comedy|Drama|Romance |
| Movieid: 74263 | Dangerous (1935) | Drama |
| Movieid: 71017 | Full Body Massage (1995) | Drama |
| Movieid: 2954 | Penitentiary (1979) | Drama |
| Movieid: 86947 | Van Diemen's Land (2009) | Drama |
| Movieid: 104583 | Logorama (2009) | Action|Animation|Crime |
| Movieid: 50942 | Wake Up, Ron Burgundy (2004) | Comedy |
| Movieid: 48045 | Fear City: A Family-Style Comedy (La cité de la peur) (1994) | Comedy |
| Movieid: 32088 | DNA (1997) | Action|Sci-Fi |
| Movieid: 27009 | Torrente, el brazo tonto de la ley (1998) | Comedy|Crime |

I consider this recommendation list from ALS model not bad. First, there are a lot of similarities in the genres of the recommended movies, such like Drama, Crime, Action. And what’s more important, these genres of movies actually are what I love.

Item-item CF in hybrid system

To mix the recommendation strategy of ALS solution and item-item CF. I decided to give scores to movies in both ALS model and item-item CF, then got the product of the two scores on each movie to decide the order of the recommended movies. One thing is very important for item-item CF. To evaluate the similarity between movies, we don’t use the attributes (genres) for movies themselves, but also use the utility matrix such like user-user CF. The formula of item similarity between movie M and movie N would be

(The count of the people who both like M and N) /

Sqrt( (The count of the people who like M) \* (The count of the people who like N) )

More implementation detail would be explored in later. Before that, I would introduce a challenge I ran into when I tried to build up a pivot table for user ratings table.

The challenge for building up pivot table

What’s pivot table? Let’s take a look at the ratings dataset, each of the ratings record included a user’s rating on a single movie. I wish to create pivot table, which would merge the records with same userId, and ratings on all movies. However, we already know that user won’t make ratings on each movie (well, they don’t have such time to do it), it should be a sparse matrix, and there would be many null values in the columns. If we tried to pivot in dataframe for pandas, it would be easy; however, the maximum number of pivot column is 10000, and we have near to 130000 movies which also means 130000 pivot columns! Therefore, I found it impossible to build up pivot table on such dataset. I deduce the method to measure item similarity is not to build up pivot table. Finally, I came up with an easier way.

Back to the item-item CF

Just like ALS recommender, I also built up the model of itemcf in `main.py` with ItemCF class. Let’s take a look at the implementation detail of ItemCF class. The major function provided in the class is `movieSimilarity()`:

1. **def** movieSimilarity(self, id1, id2):
2. # calculate the cosine similarity between movies
3. # if rating is larger than 3, it means like
4. # like id1
5. ratings1 = self.ratings\_df.filter((self.ratings\_df["movieId"] == id1) & (self.ratings\_df["rating"] >= 3.0))
6. counter1 = ratings1.count()
7. # like id2
8. ratings2 = self.ratings\_df.filter((self.ratings\_df["movieId"] == id2) & (self.ratings\_df["rating"] >= 3.0))
9. counter2 = ratings2.count()
10. # like both
11. merged\_ratings = ratings1.select("userId").intersect(ratings2.select("userId"))
12. counter3 = merged\_ratings.count()
13. **if** counter2 == 0 **or** counter1 == 0:
14. **return** 0.0
15. **return** (counter3) / math.sqrt(counter1 \* counter2)

This function can measure the cosine similarity between movies by their movieId. Take a look at the formula again,

(The count of the people who both like M and N) /

Sqrt( (The count of the people who like M) \* (The count of the people who like N) )

`counter1` in line 6 represents the count of the people who like M. `counter2` in line 9 represents the count of the people who like N. `counter3` in line 12 represents the count of the people who both like M and N. Here is another problem, we only have ratings but how do we check whether a people like the movie or not? I consider this problem should be handled by our own definition. Take myself for example, I consider the user with ratings on movie larger than three would like the movie. Therefore, I use the condition of `rating >= 3.0` to filter the rating records in line 5 and line 8.

I already have a recommended movie list with ALS model including 10 movies, and each of them have a score given by ALS model. With item-item CF, I tried to measure the similarities between my rated movies and the recommended movies by ALS model, then give another score by the method.

1. # item-based CF
2. itemcf = ItemCF(spark, movies\_path, ratings\_path)
3. ratedId = [9, 23, 66, 145, 153]
4. # testId collects the recommended movieId by ALS model
5. testId = [key **for** key **in** als\_recommend\_list.keys()]
6. **for** rated **in** ratedId:
7. **for** tid **in** testId:
8. sim = itemcf.movieSimilarity(rated, tid)
9. als\_recommend\_list[tid] = als\_recommend\_list[tid] \* sim

In line 8, I call the movieSimilarity() method of ItemCf class to measure the similarity as a score. Then I multiply the score given by ALS model by the score of the similarity. This is how my hybrid recommender system works.

Recommended movie to me by Hybrid Model

With hybrid Recommendation:

Recommend movieId: 32088with score: 7.309709816638938e-10 to me.

Recommend movieId: 5271with score: 0.0 to me.

Recommend movieId: 74263with score: 0.0 to me.

Recommend movieId: 71017with score: 0.0 to me.

Recommend movieId: 2954with score: 0.0 to me.

Recommend movieId: 86947with score: 0.0 to me.

Recommend movieId: 104583with score: 0.0 to me.

Recommend movieId: 50942with score: 0.0 to me.

Recommend movieId: 48045with score: 0.0 to me.

Recommend movieId: 27009with score: 0.0 to me.

Out of expectation, I got score on only one movie: DNA (1997). I considered this result not to be accurate enough compared to the ALS model.

Conclusion

The reason for the score 0.0 in item-item CF would be caused by the fact that there was no anyone rating for the movie or no anyone likes the movie enough. Therefore, such mistake might be caused by the threshold I set for liking movie was too high. To handle the problem, I think I can lower down the threshold in future. Another problem was that I failed to compute the Mean Average Precision (MAP). However, I have a Pseudo Code (which can be computed in ALS class) in my brain.

1. # userRecs = self.model.recommendForAllUsers(10)
2. # assume userid to be 1 and find recommended movie list for him in userRecs
3. # get the recommended movie list with sorted scores in userRecs
4. # get the movie list with ratings by user 1 and also sorted by ratings
5. # Focus only on the movies which are rated by the user 1 on both lists
6. # assume both have movie 1, movie 2, movie 3
7. # first list: [m3, m2, m1], second list: [m2, m3, m1]
8. # Both lists above are sorted by scores and ratings
9. # The MAP would be 1/3 because only m1 matches the prediction rank order

What’s the most important, I would only focus on the movie which have been rated by the user. Because only the rated movies, we can check whether predicted rank is right or wrong. Hope I would be able to figure out the MAP in future.