IEOR 4571 Project 1

Zhiyi Guo (zg2350), Yujie Wang (yw3442)
Congcheng Yan (cy2550), Banruo Xie (bx2168)

Introduction

The project below builds two collaborative filtering models, item-based method and model-based method, that used some open sourced packages and methods to implement our objective to recommend movies based on movie user score database. The most important metrics are generating similarity matrix for item-based method and rating matrix for model-based method in order to compare effectiveness in practical case by calculating their accurateness.

For business purpose, this project is primarily designed for internal company use. After comparing different methods' usefulness in database, discussion will be further made for final decision of letting which models be put into real world application to recommend movies for users. The model with more accuracy and usefulness will be selected. Our business rule is to using recommendation to improve customer satisfaction, user engagement and increase the number of short-term users to long-term user. Therefore, the better recommendation model can help us increase users and improve revenue.

Data Preprocessing

We are trying to optimize Pearson coefficient by trying to get better MSE and NDCG in our metrics. We sacrifice the movie genre, timeslot to simplify our model. As for our model, it will automatically prefer popular items since it have more raw data recorded. The database is randomly selected with around 2 million ratings and 0.4 million tag applications across 27 thousand movies. After rearranging the dataset to 3 dimensions for `userId`, `movieId` and `rating`, then we group by `userID` to use top 10 percent of user who have rate the movie most. Similarly, we use 10 percent of `movieID` that are rated most by users. The reason is that the detailed rating of user and movie can generate more accurate model below. For the first sample, we randomly select 1000 movies and 10000 users.

Collaborative Filtering

Neighborhood-based (item-based)

<u>Algorithm</u>

1. Determine a set of peers for every user
2. Calculate the similarity of each user within the peer set, comparing users on items that they have both rated
3. Use prediction function to fill in a user's missing rating--weighted average mean-centered rating from the users in the peer set

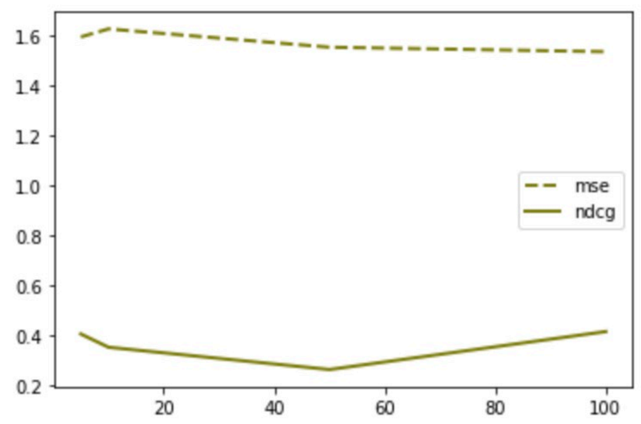
- In the function `update_pearson`, to determine the nearest neighbor for user, we use Pearson's correlation coefficient to find user's peer set. Pearson's correlation is the covariance of the two variables divided by the product of their standard deviations.
  Input: original dataframe
  Output: similarity matrix of all item in original dataframe
- In the function `predict`, it predicts ratings for each item and user in original dataframe.
  Input: original dataframe, top k similar movies for each movie
  Output: a dataframe with all predicted ratings
- In the function `recommend`, it uses predicted dataframe to recommend movies to a specific user.
  Input: predicted dataframe, how many movies recommend to a user, userid
  Output: a list of userid, movieid, predicted rating

Evaluation

We choose MSE as our primary accuracy metric and NDCG as our secondary accuracy metric. MSE stands for mean square error, which measures the average squared difference between estimated value and actual value. NDCG stands for normalized discounted cumulative gain, which measures ranking quality. In other words, we expect MSE to be as small as possible and NDCG as large as possible.

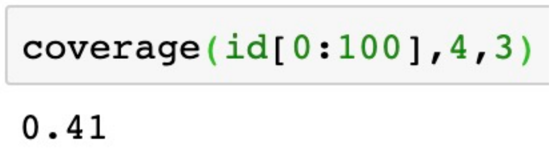
- In the function `cross_validation`, it uses cross validation to tune k (neighborhood size).
  Input: cross validation fold, top k similar movies for each movie
  Output: an evaluation list with mean MSE and mean NDCG score
- In the function `dcg_score` and `ndcg_score`, we use online reference code to compute ndcg score for the model (https://gist.github.com/mblondel/7337391).
- In the function `tune_k`, it use cross validation to tune hyperparameter k (neighborhood size).
  Input: a list of k that we want to test
  Output: a metric, each row stands for each k, the first element is mean mse, the second element is mean ndcg
- In the function `coverage`, it calculates coverage. In other words, it chooses some number of user to calculate the coverage percentage if the recommend rating is greater than the threshold rating, then we see it as success.
  Input: user, threshold, number of item need to be successful
  Output: a probability

The below picture shows the relationship between MSE and NDCG for different k (neighborhood size).



As the k increase from 5 to 100, MSE is highest at 10, and then decrease, for NDCG, it decreases and is lowest at 50 and then increase. Base on the formula and definition of MSE and NDCG, the best situation is the model have smaller MSE and bigger NDCG, which is our objective. From the graph we can have that as k goes larger, the result is closer to our objective. While the calculation and running time also increase as k increase, so we have to choose some trade-off, 'elbow method' is suggested. In this case, k should be 100.

The below graph shows user coverage, which is 0.41 as we expected.



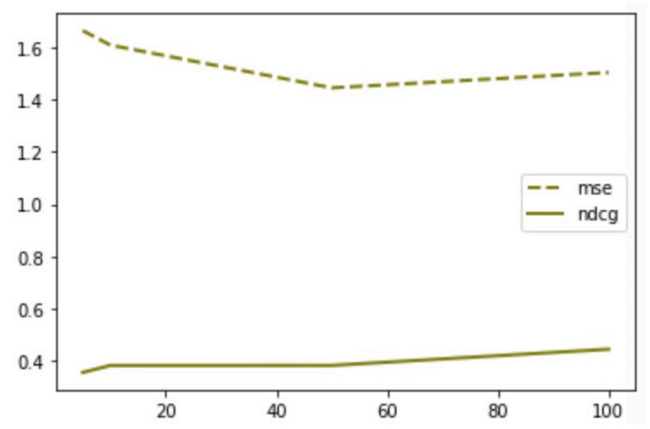
```
coverage(id[0:100],4,3)

0.41
```

Q: What other design choices might you consider in order to have a more accurate or more useful model?
A: For item-based method, we can use Jaccard similarity. Jaccard similarity between two users is the number of items that they share in common, normalized by the total number of items between two users together. It is more appropriate for binary or unary ratings, where items are either in the set, or not. We can also calculate the Euclidean distance between two points, where points are the user's data. It can be converted to similarity by subtracting it from zero.

<u>Larger Dataset</u>
For the second sample, we choose 1500 movies and 15000 users. The below picture shows the relationship between MSE and NDCG for different k (neighborhood size) in the second sample. Compare two sample, we can get the following results.

1. As the sample size increases, the overall accuracy increases for larger k (we notice that MSE decreases a lot and NDCG increases a lot). In this case, k should also be 100.

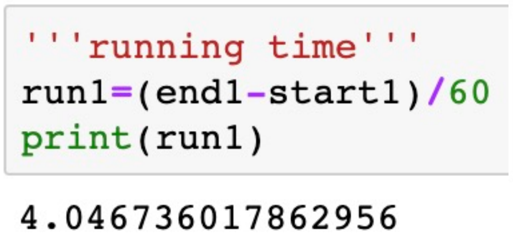


2. The distribution of accuracy is similar to smaller size data, but more accurate. We can notice that the second recommendation result has overall higher ratings than the first one.

```
[(54, 3361, 3.719271035284994),
 (54, 1197, 4.0),
 (54, 1286, 3.4609756097560975),
 (54, 2391, 3.72443294866932),
 (54, 2890, 3.603629845395474)]
```

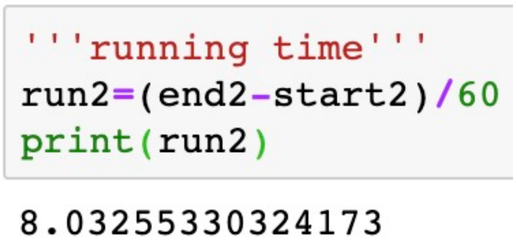
```
[(54, 1200, 4.081000443927544),
 (54, 318, 4.407446539904147),
 (54, 3471, 3.8499206983891354),
 (54, 292, 3.1783216783216783),
 (54, 1197, 4.133459255261737)]
```

3. The run-time increases as sample size increase. The smaller sample takes 4 min and larger sample takes 8 min. It is not linear relationship, the increase of running time is much bigger.



```
'''running time'''
run1=(end1-start1)/60
print(run1)
```

```
4.046736017862956
```



```
'''running time'''
run2=(end2-start2)/60
print(run2)
```

```
8.03255330324173
```

Model-based (ALS)

<u>Algorithm</u>

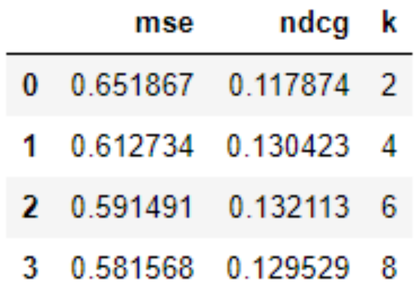
1. Cleaning the datasets for future use
2. Using alternating least squares (ALS) algorithm in pyspark factorizes the given user matrix associate with `movieID` and rating that factors into to matrix with `userID` matrix and `movieID` matrix. In Spark, the ALS use L2 regularization, which is better than L1

regularization and It minimizes two loss functions alternatively: it first keeps user matrix unchanged and runs gradient descent with item matrix; then it keeps item matrix unchanged and runs gradient descent with user matrix. For the scalability of ALS method in Spark, ALS runs its gradient descent in parallel across multiple partitions of the underlying training data from a large set of machines.

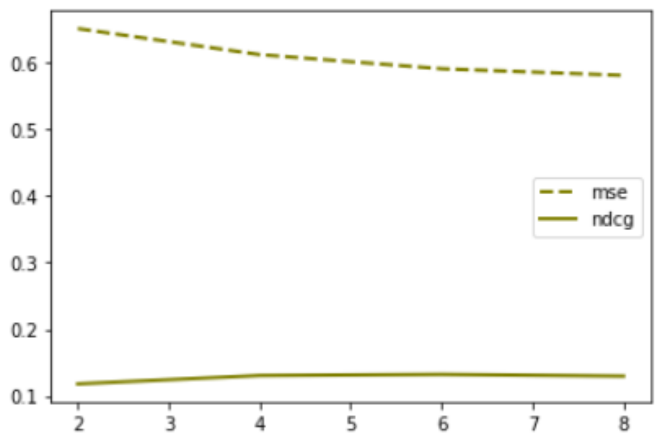
- In the function `getMse`, we predict ratings for each item and user in original data frame.
  Input: original data frame, degree, iteration, param
  Output: trained model, MSE and other data for further use
- In the function `ndcgAt`, we get NDCG from ranking.
  Input: an RDD of (predicted ranking, ground truth) set pair
  Output: the coefficient of NDCG
- In the function `recommend`, it uses predicted data to recommend movies to a specific user.
  Input: predicted data, userid, how many movies recommend to a user
  Output: a list of userid, movieid, predicted rating
- In the function `tune`, it uses cross validation to tune hyperparameter k (degree).
  Input: a list of k that we want to test
  Output: a metric, each row stands for each k, the first element is mean MSE, the second element is mean NDCG

<u>Evaluation</u>
Similarly, as the item based method, we choose MSE as our primary accuracy metric and NDCG as our secondary accuracy metric.

The below picture shows the relationship between MSE and NDCG for different k (degree).



	mse	ndcg	k
0	0.651867	0.117874	2
1	0.612734	0.130423	4
2	0.591491	0.132113	6
3	0.581568	0.129529	8

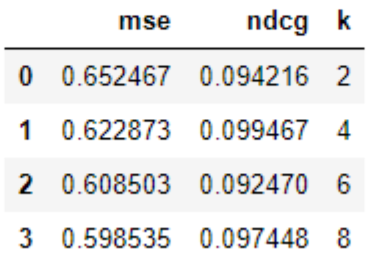


As the k increase from 2 to 8, MSE is lowest at 8. For NDCG, it increases and gets steady after 4. Base on the formula and definition of MSE and NDCG, the best situation is the model have smaller MSE and bigger NDCG, which is our objective. From the graph we can have that as k goes larger, the result is closer to our objective. While the calculation and running time also increase as k increase, so we have to choose some trade-off, 'elbow method' is suggested. In this case, the best degree is 6.

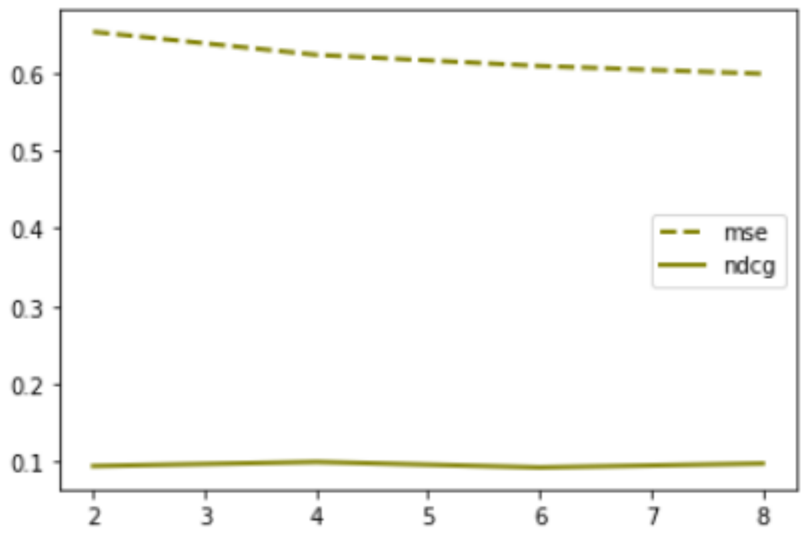
Larger Dataset

For the second sample, we choose 1500 movies and 15000 users. The below picture shows the relationship between MSE and NDCG for different k (degree) in the second sample. Compare two sample, we can get the following results.

1. As the sample size increases, the overall accuracy decreases for larger k. It is possibly because we use more data, and the way we choose data from dense to sparse. Now, the data is sparser than the first dataset. So, the result gets a little worse.



	mse	ndcg	k
0	0.652467	0.094216	2
1	0.622873	0.099467	4
2	0.608503	0.092470	6
3	0.598535	0.097448	8



2. The run-time increases as sample size increase. The smaller sample takes 74s and larger sample takes 88s. It is not linear relationship, the increase of running time is much smaller.

```
runningtime1 = getMse(data.reset_index(drop = True),6,10,0.01)[-1]
runningtime1
```

```
74.8125512599945
```

```
runningtime2 = getMse(data.reset_index(drop = True),8,10,0.01)[-1]
runningtime2
```

```
88.39221334457397
```

3. The coverage of ALS is larger than that of item-based (0.86>0.41).

```
id=list(df_result.columns)
coverage2(id[0:100],4,3)
```

```
0.86
```

Conclusion

To clarify, our objective is to compare two model and decide put which one in real. Based all results we get above, it is clear that model-based (ALS) is better because it has a smaller MSE, higher coverage, and less running time. Based on our objective, we think it is comfortable to put these solutions into production since a better recommendation model can help the company attract more users (short-term and long-term) and then increase revenue.

If I am a manager at work, I don't think I need to understand the overall staff behind the code. In other words, the write-up is enough for me. Although the model is technical, but the result (graph and explanation) is easy to follow. Thus, I think I can make a business decision based on this write-up.

As for potential watch outs, since Spark and MLlib do not have built-in function for item-based algorithm, we used python base packages and Sklearn package for the whole implementation. For convenience and time efficiency, we used online open source NDCG method instead of using NDCG from Spark. The NDCG function that we used for item-based algorithm has a slight difference in definition with that used for model-based ALS algorithm. Thus, the NDCG score for item-based runs a little higher than that for model-based in general. In this case, we should not compare both NDCG scores for different methods. Accordingly, to eliminate bias in metrics, we decided not to choose NDCG scores as a criterion when comparing models. Instead, we only used NDCG score for item-based model to tune our hyper parameter—neighbor size k. For future improvement, we will use the exactly the same evaluating metrics across different methods to compare methods.