Recommendations System Project 2

JI-DUNG, LO

1. User-Based Collaborative Filtering Algorithms

1.1 Implement the basic user-based collaborative filtering algorithms

Cosine similarity

	MAE				top k
	Test5.txt	Test10.txt	Test20.txt	Overall	
Method-1 cosine similarity	1.022133299987500	0.952833333333333	0.951384199864956	0.974963060252832	30
	0.941102913592597	0.900833333333333	0.888010031831774	0.908594647841077	160
	0.838314367887958	0.779333333333333	0.766374071573261	0.793178460022985	160
	0.925096911341753	0.937833333333333	0.934214333944246	0.932112953538007	1

Pearson Correlation

	MAE				top k
	Test5.txt	Test10.txt	Test20.txt	Overall	
Method-2 Pearson Correlation	0.912592222083281	0.870333333333333	0.842288029323816	0.872270563125924	45
	0.864574215330749	0.800666666666667	0.793189929584258	0.818461664751272	1
	0.877454045266975	0.816333333333333	0.806501398668853	0.832211459530455	160
	0.880330123796424	0.797666666666667	0.779203241053342	0.816943030701034	160

Pearson Correlation IUF

	MAE				top k
	Test5.txt	Test10.txt	Test20.txt	Overall	
Method-3 Pearson Correlation IUF	1.009503563836440	0.977333333333333	0.936915211729526	0.970694467246758	45
	0.879329748655746	0.827500000000000	0.809973955821356	0.837054670825809	1
	1.043016131049140	1.019500000000000	0.989582328542491	1.014488589722540	160
	0.966737526572465	0.908500000000000	0.896209125108517	0.922385486783779	160

Pearson Correlation Amplification

	MAE				top k
	Test5.txt	Test10.txt	Test20.txt	Overall	
Method-4 Pearson Correlation Amplification	0.864574215330749	0.800666666666667	0.793189929584258	0.818461664751272	1
	0.880455170689008	0.815166666666667	0.809105816533230	0.834017402725332	10
	0.877829185944729	0.816333333333333	0.806790778431562	0.832457724511574	160

Pearson Correlation IUF & Amplification

	MAE				top k
	Test5.txt	Test10.txt	Test20.txt	Overall	
Method-5 Pearson Correlation IUF&AMP	0.879329748655746	0.827500000000000	0.809973955821356	0.837054670825809	1
	0.943978992122046	0.886000000000000	0.864184431368766	0.895747824659333	10

2. Item-Based Collaborative Filtering Algorithm

	MAE				top k
	Test5.txt	Test10.txt	Test20.txt	Overall	
Method-6 item-based	0.862698511941978	0.801000000000000	0.791646570849812	0.817271384009194	

3. Implement your own algorithm

I first used Method 9 to predict the train data with train data and then used the predicted train data to predict test data. The result is surprisingly bad and the reason is that you never know the ground truth data so if you predict the ratings based on false ratings, then the result could surprisingly terrible.

Then I implemented Majority voting and the result is not too well since I found out that most of time the result is just test users' own average ratings and cannot correctly predict those very popular or unpopular movies.

Lastly, I implemented the weighted average method, which combined the 6 methods above and tried to find the best combination of weights that can yield the best MAE. To find the best combination of weights, I first thought the issue was actually just finding the overall minimum point in a 6-dimensional space, so I implemented the gradient descent which I learned in machine learning class with MSE because MAE is not differentiable. However, the time gradient descent took to find the best is too long because for every random initial random start, there are 200 iterations to search for the lowest point. So, I instead used a random approach to generate a set of weights and test their MAE, and repeated this for 200 to 300 epochs. After finding the potential minimum point's place by randomness, I then used gradient descent to find the minimum point around that place with a low learning rate.

Then I found in my split training data that there are actually several groups of weights that can produce the same or little difference in MAE. I put these to actual testing and found that some weights are actually more effective for test5 and test10, while some weights are more effective for test20, so I combined the best weight combinations of each test set to get an optimal MAE. Finally, I tested edge cases. If the predicted rating exceeds the range of 1 and 5, is it more effective to use the average rating of test users or the average rating of movies? The result is that using the average rating of test users will reduce the MAE of test5 and test10 a

little bit, while using the average rating of movies (the last row) will increase test5, test10, and overall MAEs, so the final best MAE is to use the average rating of test users to optimize test5 and test10. There was actually a mistake in the selection of Top k, because the split training data I tested would have the best results covering all train users, so I used 160, but when I took it to the actual test, I forgot to switch back to 200, and when I found out, all the number of submissions had been used up, so theoretically the MAE value could be lower.

Majority voting:

	MAE				top k
	Test5.txt	Test10.txt	Test20.txt	Overall	
Method-7 majority voting	0.941102913592597	0.900833333333333	0.888010031831774	0.908594647841077	[160,1,1,1,1]

Prediction train data with train data and then use it to predict test data:

	MAE				top k
	Test5.txt	Test10.txt	Test20.txt	Overall	
Method-9 (Predict train data with train data and then predict test data)	1.336126047267730	1.382666666666670	1.357094627182410	1.356509604334260	30

Weighted Average:

	MAE				top k
	Test5.txt	Test10.txt	Test20.txt	Overall	
Method8-weighted average	0.941102913592597	0.900833333333333	0.888010031831774	0.908594647841077	[160, 1, 1, 1, 1]
	0.797924221583094	0.740166666666667	0.724414005980515	0.752421605647677	[160,160,160,160,160]
	0.796173565086908	0.740000000000000	0.726343204398572	0.752626826465277	[160,160,160,160,160]
	0.796173565086908	0.751500000000000	0.735121057200733	0.759193892628468	[160,160,160,160,160]
	0.796173565086908	0.740000000000000	0.724414005980515	0.751805943194878	[160,160,160,160,160]
	0.798424409153432	0.742166666666667	0.725089225426835	0.753365621408636	[160,160,160,160,160]
	0.797674127797924	0.740000000000000	0.724606925822321	0.752380561484157	[160,160,160,160,160]
	0.795923471301738	0.739000000000000	0.726150284556767	0.752216384830077	[160,160,160,160,160]
	0.795923471301738	0.739000000000000	0.724414005980515	0.751477589886718	[160,160,160,160,160]
	0.798299362260848	0.742166666666667	0.724414005980515	0.753037268100476	[160,160,160,160,160]

Do you think your results are reasonable? How can you justify the results by analyzing the advantages and disadvantages of the algorithms?

At first, I thought my results for Pearson Correlation family are not reasonable because as the k goes up, the MAE drops. Then it turned out that it is because the way I computed the Pearson Correlation weight is wrong. In addition, my first implementation for cosine similarity is also wrong, so the only correct result for cosine and Pearson is the second k = 160. Based on the MAE, I think the item-based result is better than all of the user-based methods because the matrix is sparse and users' ratings are very random. Even if there are similar users, if the number of similar users is small, the prediction will be inaccurate, and even if the similarity is very high to 1, there may be edge cases, so it needs to be excluded or replaced by the average rating. Therefore, I divided the train data into 160:40 and used all 160 train users for prediction. The result is that as long as top k is larger, the MAE will be smaller. The advantage for user-based CF is that if we can have more data from the users, the predictions will be more accurate. The disadvantage is that the rating matrix actually resembles the real-world data, which is always sparse, so we cannot predict very well with user-based CF. The advantage of item-based is that it can accurately evaluate those movies that everyone thinks have high or low ratings. Hence, the MAE is significantly lower than those in user-based methods. The disadvantage is that item-based CF cannot detect users' serendipity, which often occurs in every user's rating. Some movies with an average rating of 4 or above may be rated 1 or 2 by this user, and vice versa. So the best method is the weighted average, which combines all methods and then finds the best weight suitable for each data set.

Instructions to use the code:

Just open the annotation section to test. But there are several things to do for standalone testing and method 8.

For standalone test, remember to comment out the astype for test_data and open the round() method in each method. I have written #todo instructions in each method.

To test method 8, first commenting out round() method and opening the next one, then using weighted_20 for test20 and weighted_5_10 for test5 and 10. To replicate the same result, use the same dynamic_top_k = [160, 160, 160, 160, 160]. To get the theoretically best result, use the dynamic_top_k = [200, 200, 200, 200, 200].