COEN 272 PROJECT II

Yue Liu

1.1 User-Based Collaborative Filtering Algorithms

1) Cosine

#		Test5	Test10	Test20	OVERALL	Conclusion
1	cosine	0.8238	0.7893	0.7694	0.7921	
2	cosine + significance smoothint(25,25,25)	0.8330	0.7913	0.7728	0.7971	significance smoothint with value 25 doesn't work well for the dataset
3	Cos +Dirichelet smoothing(1,1,1)	0.8250	0.7876	0.7688	0.7919	Dirichelet smoothing with value (1,1,1) help to improvement the performance
4	Cos +Dirichelet smoothing(1,1,1) + IUF	0.8260	0.7886	0.7687	0.7924	IUF doesn't help
5	Cos +Dirichelet smoothing(1,1,1) + CM (2.5)	0.8371	0.7866	0.7683	0.7954	CM doesn't help
6	Cos +Dirichelet smoothing(1,1,1) + neighbor (100) + threshold (>=0.0)	0.8250	0.7876	0.7688	0.7919	Choosing 100 neighbors is the same with choosing 200 users
7	Cos + Dirichelet smoothing(1,1,1) + neighbor (200) + threshold (>=0.5)	0.8240	0.7866	0.7687	0.7913	Best performance when choose threshold >= 0.5

Summary:

1. The dataset is not large, the "helpful" train users who have common rating with test user is less than 100(case 3 and case 6 shows), so it's not good to choose a smaller value for K neighbor.

2. Many “helpful” train users just have 1 common rating with test user, so we can’t discard these train users, so let Dirichelet smoothing = 1 will take them into account.
3.Train user who has 1 common rating with test user will have similarity = 0.5(with Dirichelet smoothing = 1), so let threshold >= 0.5 make sense, which means train users whose similarity with test user is less than 0.5 will not help to make a prediction.
4. Dirichelet smoothing gives better performance than significance smoothing in the dataset, so I use Dirichelet smoothing in other algorithms also.
5.

- Dirichelet smoothing : $W'uv = \frac{|Iuv|}{|Iuv| + b} * Wuv$
- Significance smoothing: : $W'uv = \frac{\min\{|Iuv|, r\}}{r} * Wuv$

2) Pearson

#		Test5	Test10	Test20	OVERALL	Conclusion
1	pearson	0.8737	0.8161	0.7739	0.8171	Worse than cosine if we don’t tune the algorithm
2	pearson + smoothing average + Dirichelet smoothing(1,2,4)	0.8451	0.8058	0.7605	0.7994	Smoothing helps
3	Pearson + smoothing average + Dirichelet smoothing(1,2,4) + neighbor (?) + threshold(?)	I tries some combination values of neighbors and threshold, but it doesn’t help increase the performance, it’s hard to determine the best value for K neighbors and threshold for in the given dataset				
4	pearson + smoothing average + Dirichelet smoothing(1,2,4) + threshold(>=0)	0.8049	0.7763	0.7364	0.7687	Only taking “positive users” into account helps to increase the performance

Summary:
1.Basic pearson method doesn’t work better than basic cosine method since there are too many train users with only one common rating with test user(both cosine method and pearson method can’t resolve one common item problem if we don’t use some other tuning method like smoothing).
2.The dataset is too small, it’s hard to find a better value for K neighbors and threshold.
3.”Positive users” is more importance than “negative” users, it makes sense, there are many examples in real life.

1.2. Extensions to the basic user-based collaborative filtering algorithms

1) Pearson + IUF

#		Test5	Test10	Test20	OVERALL	conclusion
1	Pearson + IUF	0.8445	0.7545	0.7339	0.7753	The performance when using IUF is amazing
2	Pearson + IUF + smoothing average + Dirichelet smoothing(1,2,4) + threshold(>=0)	0.7990	0.7473	0.7309	0.7573	Smoothing always helps to increase the performance, and only taking "positive users" into account also helps in this case

Summary:

1. Smoothing always helps to increase the performance
2. IUF is amazing which increase the performance a lot.

2) Pearson + CM(1.5)

#		Test5	Test10	Test20	OVERALL	conclusion
1	pearson + CM + smoothing average + Dirichelet smoothing(1,2,4)	0.8406	0.8131	0.7339	0.7999	With/without CM is the same
2	pearson + CM + smoothing average + Dirichelet smoothing(1,2,4) + threshold(>=0)	0.8055	0.778	0.7421	0.7717	CM decrease the performance compared to without CM when only taking "positive users" into account

Summary:

1.CM doesn't help, compared to without CM(case2 in Pearson and case 1 in Pearson + CM)
2.CM even decreases the performance when we only take "positive users" into account, compared to without CM(case4 in Pearson and case2 in Pearson+CM)

2. Item-Based Collaborative Filtering Algorithm

#		Test5	Test10	Test20	OVERALL	conclusion
1	Item based	0.8890	0.8148	0.7566	0.8144	

2	Item based + smoothing average + Dirichelet smoothing(50)	0.8638	0.7983	0.7496	0.7991	Smoothing always help

Summary:

1. Smoothing always helps to increase the performance.

1. The number of users giving rating to both item1 and item2 is small, which from X to 1X from my observation.

2. A reasonable Dirichelet smoothing value from a paper is 50, I have a try and it works. I guess item-based algorithm works better when more users give both rating to item 1 and item2, if the number is below 50, it's better to decrease these users' influence which avoiding noisy.

3. Implement your own algorithm

My best algorithm is combing these algorithms:

	Test5	Test10	Test20	OVERALL
0.1 * Item based 0.3 * User based Cosine 0.6 * User based Person with IUF	0.7719	0.7273	0.7182	0.7380

4. Results Discussion

1). The comparison between with only one algorithm is as follows

cosine	pearson	Pearson + IUF	Pearson + CM	Item based
0.7913	0.7687	0.7573	0.7717	0.7991

Summary:

1.All the algorithms help to make prediction to the test users' rating.

2.We must tune the algorithm based on the attribute of the given data set.

3.In the data set, IUF helps a lot and item based algorithm is not as good as I expected, the reason is the number of users who rates both item1 and item2 is too small.

4.Combining algorithms together helps a lot as we learned in class, since every algorithm has its disadvantage.

2). Efficiency

Summary:

1

I did some preprocess to speed up the prediction system, it takes 3.7 seconds to run all the algorithms I implemented in Java

- Extracting the train data from dick to main memory once and it will be used for all the algorithms.
- Recording the position of missing rating for test user when preprocess, so that I don't need to traverse the whole sparse 2-dimention user matrix every time.

I also programmed my own test case which choosing 150 random train users to predict the other 50 train users, which helps a lot to tune my implementation.

2.Time complexity analysis

- Ut: number of train user
  Un:number of test user
  I : number of items
- User-based algorithm will always take O(Nt * Un * I) to generate the 2-dimention user-similarity matrix, plus sorting in case we need to find the K-similar users.
- Item-based algorithm will always take O(I * I * Ut) to generate the 2-dimention item-similarity matrix, plus sorting in case we need to find the K-similar items.