CSEN 272 Project 2 Report
Yanxu Wu (W1650780)

Result Table:

	test5 MAE	test10 MAE	test20 MAE	Overall MAE
Basic Cosine	0.828	0.779	0.766	0.789
Basic Pearson	0.879	0.80	0.778	0.816
Cosine-Case-Amplification	0.829	0.777	0.764	0.788
Cosine-IUF	0.828	0.779	0.766	0.789
Pearson-Case-Amplification	0.888	0.816	0.806	0.835
Pearson-IUF	0.879	0.80	0.778	0.816
Item-Based-Adjust-Cosine	0.816	0.781	0.761	0.784
My Own I	0.816	0.764	0.753	0.777
My Own II	0.808	0.733	0.742	0.752

Summary:

Implemented two versions of the basic user-based collaborative filtering algorithm as the Cosine similarity method and Pearson Correlation method.

Implemented the following two modifications to the standard algorithm (using Pearson Correlation and Cosine similarity): 1. Inverse user frequency; 2. Case amplification.

Implemented the item-based collaborative filtering algorithm based on adjusted cosine similarity.

Implemented my own algorithm based on observation of the results and research.
I: Combine weight of 0.65 * cosine + 0.35 * pearson
II: Refined Bias-SVD machine learning model

Discussion:

When implementing of the two basic algorithms, I found the similarity between users are high, from choosing k = 100 and k = 200, the results of 200 are better. So, I set k = 200, which means taking all users into consideration. Then using IUF both on Pearson and Cosine similarity have little positive effect, maybe it's because users rated too little movies. And in this dataset, case amplification (p = 2.5) has a positive effect on both two algorithms, especially cosine. Overall, of all the standard algorithms, cosine

method works better than Pearson, the best result comes from adjusted cosine with case amplification, with MAE of around 0.788.

Item-based cosine similarity performance is similar to user-based cosine similarity with this training data, because in this sparse matrix, there are not much similar movies to find, so we can expect better performance if a lot of movies have non-zero ratings.

My own method I uses a combination of cosine with case amplification and Pearson, with 0.65 * cosine_rating + 0.35 * pearson_rating, this way we can leverage both methods in rating movies, a little bit like F-measure(F=2PR/(P + R)) in evaluation metrics, both must be high for combined rating to be high. But considering this dataset, I assigned more weights on cosine since it's more effective in sparse matrix, and the result MAE = 0.777 is the second best of all results, proving it's an effective rating algorithm.

When searching online, I learned Singular Value Decomposition (SVD) model is used in the Netflix contest to predict movie ratings. So I implemented a simple refined version of it. The final MAE result is 0.752, making the best result of mine. The term bu represents the user bias, which is a parameter that needs to be trained. It reflects a specific user's rating habits. Similarly, bi represents the item bias. It indicates the rating situation of a specific item. Adding such bias into the standard SVD model is very reasonable and has proven to be very effective.

The code (SVD.ipynb) implements a collaborative filtering recommendation system using to predict movie ratings. First load the training data file. Then, I decided to add known ratings in the test data and incorporate into the training set to create a combined dataset for better training results. The combined data is then split into training and validation sets, and I used RMSE and MSE in addition to MAE metrics for cross validation.

The SVD class is defined, encapsulating the initialization of user and item factor matrices, bias terms, and methods for summing item-specific factors, predicting ratings, and training the model. The training method uses early stopping based on validation performance to prevent overfitting. The best hyperparameters (number of epochs, number of factors, learning rate, regularization rate) are specified by grid search. During training, the model's performance is evaluated on the validation set, and training stops if MAE has no impartments, meaning the delta values is less than 0.0001 in each epoch.

After training, the model predicts unknown ratings (initially marked as zero) for a specific range of users (401 to 500) in the test data. These predictions are collected, sorted, and saved to a file.

In conclusion, I think the results are reasonable, showing the advantages/disadvantages of different algorithms. For example, in this sparse matrix, Pearson correlation is less effective than cosine because it relies on having overlapping ratings between users. Cosine similarity, on the other hand, can handle sparsity a bit better since it only considers the angles between vectors. Also, Cosine similarity focuses on the direction of the rating vectors and is less influenced by the magnitude of the ratings. This can be beneficial if the magnitude of the ratings is not as important as the relative comparison of the items rated by users. The reason why case amplification works better than IUF is because IUF is used to down-weight the influence of popular items that many users have rated, as these items are less informative for prediction, but lots of movies have not been rated in this data, so its effect is not obvious.

Code Running Instructions:

1.Downlaod and install jupyter notebook on your computer
2.Then in terminal, direct to folder p2, then type: jupyter notebook
3.The main file to run is p2.ipynb and SVD.ipynb

There are two utility classes in folder Utils, and online submission screenshots for different MAE results in folder result_screenshot, best result file are kept in folder best_result

Efficiency:

The execution time of this program based on my own method I for
test5: 14.94 sec
test10: 12.80 sec
test20: 38.10 sec

So, I think it's not a bad run time. The program code is attached in next page.