Project 2 Report

Tisha Harnlasiri
Dr. Yi Fang | COEN 169 | 5 June 2018

1. Introduction

This project mirrors the setup of the Netflix Prize competition held from 2006 to 2009. The goal of the competition was to develop an algorithm that predicts user ratings on films with an improved accuracy of 10% over Netflix's own Cinematch algorithm. Although Netflix's dataset consisted of well-over 100,000,000 ratings , for this project we were given a smaller training dataset of ratings made by 200 users for 1000 movies. A movie rating was an integer in the 1 to 5 range with 1 meaning "least liked" and 5 meaning "most liked." A 0 meant that the user had not yet rated that movie. We were also given three test datasets (test5, test10,and test20), each containing data in the form of (user id, movie id, rating) for 100 users. The test datasets provided 5, 10, and 20 known ratings for each user in those files respectively. Our task was to best predict the unknown ratings in the test datasets by designing, implementing, and building intuition about various collaborative filtering algorithms.

2. Programming Language

The algorithms were implemented with Python 3.6 using the NumPy and Pandas libraries.

3. Results & Analysis

Mean Absolute Errors (MAEs) Given 5, 10, and 20 Known Ratings Per Test User

	Given 5	**Given 10**	**Given 20**	**Overall**
Cosine similarity	0.82593	0.7925	0.77197	0.79474
Pearson	0.85819	0.80316	0.75084	0.79897
Pearson w/ inverse user frequency (IUF)	0.85707	0.79933	0.75161	0.79798
Pearson w/ case amplification	0.86220	0.822	0.77293	0.81432
Item-Based	0.92585	0.8275	0.78219	0.84050
Custom	0.76853	0.74483	0.72827	0.74556

I) Comparison of Accuracies

All four of the user-based, non-custom algorithms achieved overall MAEs of about 0.8. Cosine similarity performed the best overall, although it only beat out the Pearson-based algorithms by about a 0.003-0.004 margin. Pearson with case amplification performed the worst out of the four user-based algorithms. This might have occurred because it over-amplified higher weights, which could have been misleading due to data sparsity. Looking at the bigger picture, these four algorithms performed the worst given only 5 known user ratings; their MAEs in this case was about 0.85. This was expected considering that 5 known ratings per user provide insufficient data to make accurate predictions. In contrast, the algorithms each performed the best given 20 known user ratings, since there is much more data to work with.

The item-based algorithm performed the worst out of all six algorithms with a 0.84 MAE. As expected, it performed the worst given only 5 ratings and the best given 20 ratings. However, the item-based algorithm's worst and best MAEs were still "much" higher relative to the user-based algorithms. At 5 and 20 ratings, its MAE sits 0.07 and 0.02 above the other algorithms.

The custom algorithm, which combines the adjusted cosine and Pearson-IUF models, performed the best overall as well as across each test dataset. Its overall MAE was 0.746, which is about 0.04 lower than the overall MAEs of the other five algorithms. Given only 5 known user ratings, it significantly improved the MAE by about 0.08 and given 10 user ratings, it improved the MAE by 0.05. These significant improvements, particularly for the test5 case, were key to "drastically" lowering the overall MAE.

II) Analysis of Results

The results described above are reasonable considering the advantages and disadvantages of each algorithm and how corner cases were handled. In fact, the results relied significantly on the methods with which the algorithms were tweaked for better performance. In this section, I will be discussing the pros and cons of each algorithm and how this affected how well an algorithm performed. In parallel, I will be mentioning some of the tweaks I made to counter the negative side-effects and the various ways I attempted to handle special cases that initially skewed results.

A. Cosine Similarity

Cosine similarity measures how similar the ratings are for active user ***a*** and training user ***u*** across movies that both ***a*** and ***u*** have rated. It therefore works very well if two users have given several of the same movies the same (or very nearly the same) ratings. The disadvantage, however, also comes from the algorithm's simplicity. If users have varying expectations for what makes a movie "good" or "bad," cosine similarity has no way of accounting for such differences in user-rating scales. In addition, cosine similarity would not work well if we do not have lots of data for which several users have rated *several* of the same movies in the *same* way. So in this case, positive feedback tends to be more useful than negative feedback. I believe this generally makes the method more reliable.

Another major disadvantage is that if ***a*** and ***u*** have rated only one movie in common, the cosine similarity is 1. This marks ***u*** as being highly similar even though the two ratings on that movie are not similar at all. This is likely the reason why my first version of cosine similarity had a 0.802 MAE. I countered this issue by multiplying all computed weights by *min(# of common items, # of items a has rated) / (# of items a has rated)*, as

inspired by reading a GroupLens social computing article [1]. If ***a*** and ***u*** have rated many of the same movies, this value will be higher; thus, sim(***a***,***u***) will also be higher.

B. Pearson Correlation Coefficient (with and without modifications)

The advantage of the three Pearson-related algorithms—Pearson, Pearson with IUF, and Pearson with case amplification—is that unlike cosine similarity, they each take into account a user's personal rating scale. For example, user A might be a harsh critic who tends to give movies lower ratings from 1 to 3, while user B might give higher ratings from 3 to 5. Thus, a rating of 3 from user A might be equal to a rating of 5 from user B. The Pearson correlation coefficient accounts for this difference by subtracting users A's and B's ratings for a movie from their average ratings, respectively. This centers the ratings around a more standard scale. By doing this, we are able to leverage information from negative weights or from "opposite" users.

The disadvantage, however, is the case when active user *a* consistently gives movies the same or about the same ratings. Since we must subtract *a*'s average rating from its rating for an item, the correlation coefficient will be zero or close to zero and will therefore not provide us much information. To prevent the case of the zero-correlation-coefficient, I first checked if ***a*'s** known ratings were equal or nearly equal to ***a*'s** average ratings by computing *norm(**a**'s known ratings - **a**'s average)*. If this value was less than one, I simply predicted a target movie's rating to be ***a***'s average rather than compute the Pearson correlation coefficient at all. This lowered MAEs by about 0.02-0.03.

Pearson with IUF likely performed slightly better than all the Pearson-based algorithms because it takes into account how frequently rated a movie is. Ratings for a movie that is generally well-liked/disliked might skew prediction results because the ratings are already expected to be similar for that movie; thus, the movie should not be given much importance when computing the correlation coefficient. IUF helps to achieve this by scaling down movie ratings based on how many users rated each movie.

Pearson with case amplification attempts to give more importance to weights during the prediction computation. This would work if we were quite sure that the similarity weights were quite accurate. However, because it occurs after similarity computations have been completed, it does not really help with measuring how useful or similar other users are. This may be the reason why it performed worse than the other user-based algorithms.

For each of these three Pearson algorithms, I also penalized sim(a,u) if a and u have only rated a few items in common. I did this by multiply the weights by *min(# of common items, # of items a has rated) / (# of items a has rated)*, just like I did with cosine similarity. This is likely why all four user-based algorithms performed generally the same.

C. Item-Based with Adjusted Cosine Similarity

One of the advantages of the item-based algorithm is that it is not as prone to changes in user-interests over time. Due to this, computing item-to-item similarity may be more reliable than user-to-user similarity. For this project, we were asked to implement the item-based algorithm using adjusted cosine similarity. This takes into account a user's average rating across items, which is an advantage since it considers a user's personal tendencies. One problem with the item-based algorithm is that it depends quite heavily on how many items an active user *a* has rated. For the test5 and test10 datasets for this project, a target movie whose rating we were trying to predict for *a* could only be compared to five and ten movies that the user has rated. This is a rather small, insufficient

number and is likely why the item-based algorithm performed so poorly for these two datasets (MAEs were greater than 0.8.). Furthermore, while adjusted cosine similarity does have its advantages as previously mentioned, it is still prone to changes in user interests. Therefore, I did test out an item-based version of the Pearson method instead and obtained much more accurate results with an overall MAE of 0.741.

D. Custom Algorithm

My custom algorithm averages the rating predictions from two models. The advantage to taking the average is that it balances out any predictions from either of the two models which happen to be skewed too low or too high.

Specifically, the custom algorithm combines the Pearson-IUF method and a user-based adjusted cosine similarity method. The selection of Pearson-IUF was because it was a top performer among the five algorithms we were assigned to implement. While I had initially chosen cosine similarity as the second model to use, my final choice of user-based adjusted cosine similarity was inspired by a GroupLens social computing article I read online [1]. In this version of adjusted cosine similarity, ***a***'s and ***u'***s averages are subtracted from their movie ratings and then multiplied, just like in the user-based Pearson algorithm. However, the sum in the numerator is then normalized across all 1000 movies ratings *regardless of whether* **a** *and* ***u*** *both rated that movie*. The similarity weight for (***a***,***u***) will not be high if they have rated only a few items in common, because the magnitudes of the vectors will naturally dampen the value [1]. The damping compensates for the corner case in which cosine similarity is 1; that is, when ***a*** and ***u*** only rated one movie in common. Therefore, the custom algorithm combines 1) Pearson's and adjusted cosine similarity's advantage of comparing ratings on a mean-centered scale, 2) IUF's scaling factor which lessens the importance of weights for generally well-liked/disliked movies, and 3) adjusted cosine similarity's damping factor.

* Note: I did achieve a 0.741 MAE using the adjusted cosine similarity method for item-based collaborative filtering. However, since that method was only attempted for fun, I chose to describe this method instead. The algorithm for the item-based method is still included in my code, labeled 'pearson' in the item_based.py file.

E. Other Modifications and Tweaks

Choosing which value to set k to for the k-nearest-neighbors method was crucial to obtaining smaller MAEs I initially took the square root of the total number of users *n* who had rated the target movie and rated at least one other item that the active user had rated. However, because this number *n* generally fell in the 0 to 100 range, sqrt(*n*) was at most ten. A neighborhood of ten users failed to provide enough data to yield accurate results and thus, produced a high MAE of 0.85. After realizing this issue, I tweaked the *k* value to 40/50 and obtained much better results.

In addition, for collaborative filtering, when an active user ***a*** had no similar users, I initially defaulted a target movie's rating to 3. However, I found that using ***a***'s average yielded slightly better results.

Sources

[1] M. Ekstrand, "Similarity Functions for User-User Collaborative Filtering," 04-Oct-2013. [Online]. Available: https://grouplens.org/blog/similarity-functions-for-user-user-collaborative-filtering/.