Evan Johnson

Coen 169

3/12/19

**Project 2: Movie Recommendation System**

**User-Based Filtering**

|  |  |
| --- | --- |
|  | **MAE** |
| **Cosine Similarity** | 0.838162863240847 |
| **Pearson Correlation** | 0.815137087506157 |
| **Pearson with Case Amplification** | 0.811114759481202 |
| **Pearson with Case Amplification and IUF** | 0.764940075521261 |

**Cosine Similarity**

For my first attempt at using user-based filtering to make predictions, I used cosine similarity to measure the similarity between users based on their movie ratings. In order to do this, I constructed vectors for the ratings for a test user and computed the cosine similarity of this test user vector with each user vector in the training set. After computing the various similarities, I sorted them in descending order, and picked out the top 5 users in the sorted list with the greatest similarity to the test user whose movie ID we needed to predict. I then took these most similar users along with their respective weights and computed the weighted average of the ratings these users gave the specific movie we were trying to predict. I was able to achieve a MAE value of approximately 0.838, which is fairly good for a simple cosine similarity and weighted average algorithm.

**Pearson Correlation**

Next, I implemented the Pearson correlation algorithm in order to account for different rating levels. This algorithm has the advantage over simple cosine similarity because it accounts for different rating levels by allowing us to center our data around a standardized mean. For Pearson correlation, I calculated the mean of known ratings of the test user and the known ratings of each training user. I then subtracted each mean from each users’ respective ratings, and then found the similarity between the test user and every training user instance. After computing the similarity weights between users, I did the same thing as in cosine similarity where I singled out the top 5 most similar users and computed the Pearson weighted average to predict the unknown movie’s rating. As we can see, the Pearson correlation performed significantly better than simple cosine similarity, which makes sense since it normalizes the data in order to account for inconsistencies.

**Pearson Correlation with Case Amplification**

Case amplification improves our result by amplifying the weights of significant users and punishing insignificant users. This method can be very useful especially if there are a lot of different users with high correlations within the testing set. In this case, Case amplification helped to some extent because it was able to magnify the effect of the important weights on the weighted average. However, we did not see significant increase in the MAE, likely because many of the movies to be predicted did not have any other users in the training set rating them in the first place.

**Pearson Correlation with Case Amplification and IUF**

Finally, I added the IUF (Inverse User Frequency) computation to my algorithm. I accomplished this by gathering the number of users who rated each specific movie at the start. Next, just before computing Pearson correlation, I multiplied each training user rating by its respective IUF, which I calculated using log(N/Nm) where Nm tells us the number of users who rated movie m. IUF helps us diminish the effect that movies that are rated by a lot of people have on our weighted calculation. This is useful because universally liked movies are less useful in similarity calculations than uncommon ones. Ultimately, IUF helps us amplify the effect that rare movies have on our predictions, thereby resulting in increased accuracy from our Pearson Correlation with Case Amplification.

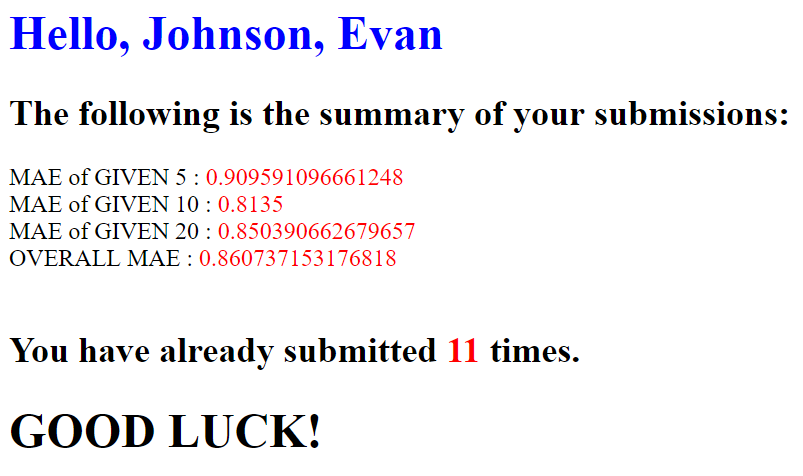
**Item-Based Filtering**

|  |  |
| --- | --- |
|  | **MAE** |
| **Adjusted Cosine Similarity** | 0.860737153176818 |

In my item-based filtering algorithm, I used adjusted cosine similarity to compute the similarity between different movies. In item-based filtering, the goal is to recommend a movie to a user based on that movie’s similarity to other movies that were rated by other users in the data set. In this particular case, I took the test users’ known movie ratings and looked up these movie IDs in the training set. I then built vectors containing each movie the test user rated from the training set, later subtracting each individual users’ average rating from each movie in the vector. Next, I calculated the cosine similarity between the train movie vectors to find how similar they were to the unknown movie. I then took the weighted average of these movies to find the predicted movie rating for the given test user.

After running the algorithm and examining the test data, I noticed several differences compared to User-based filtering. One of these was that the item-based filtering algorithm ran significantly faster, likely because the user-based filtering requires us to compute similarities between all users, while this one only requires us to compute similarities for movies the user had already rated. Furthermore, after examining the resulting predictions, I noticed that they were much more consistent across users. In this case, since many users have rated a lot of different movies, basing the weighting off of these movies creates a much more stable output.

Item-based filtering is more widely used in recommendation systems due to its stability and efficiency over user-based filtering. However, in this case, it seems to have underperformed user-based filtering. One possible reason here is because the algorithm is primarily useful in systems that have more items than users. Our system has users who have rated a large number of items, hence it is more informative to use a user-based recommendation system.



**Custom Algorithm**

|  |  |
| --- | --- |
|  | **MAE** |
| **Combination user and item based** | 0.78115252011164 |

For my custom algorithm, I implemented a combination of user-based and item-based filtering. I essentially used the user-based and item-based method from earlier, and ran the two algorithms concurrently for unknown test user movie predictions. After getting the respective prediction values from each algorithm, I took the average of the two. Surprisingly, I got very good results immediately. While user-based filtering allows us to make rigid, pinpoint predictions, item-based filtering more broadly captures the general trend of the data. Using a combination of the two helps smooth out the rough edges while maintaining great general accuracy.

