Collaborative-Filtering Algorithms

Results

Algorithm	**MAE**
User based CF with Cosine Similarity	0.8012064029911784
User Based CF with Pearson Correlation	0.8160454518899339
User Based CF with Pearson Correlation and IUF	0.7992492843372087
User Based CF with Pearson Correlation and Case Modification (amplification = 2)	0.8240345854997956
Item Based CF with adjusted cosine similarity	0.8130659578197114

Analysis

User-Based CF with cosine similarity

There are a few problems with Cosine similarity which may have affected its effectiveness for this dataset:

1. If only one movie is common between the users, then the similarity comes out to be 1. Using this value is not a correct representation of similarity. On the other hand, ignoring it will mean there is no representation of similarity between those users.
2. Even if the users are similar, it doesn't take into account how the current user generally rates, i.e., whether the user is a generous rater or not.
3. It doesn't take into account the number of common movies between 2 users.
   For example:
   The cosine similarity between user 1 and user 2 is 0.94 and they have rated 2 common movies.
   The cosine similarity between user 1 and user 3 is 0.92 and they have rated 4 common movies. It is obvious that user 1 is more similar to user 3 because they have more movies in common compared to user 2, without sacrificing much on the cosine similarity.
4. It doesn't consider users with opposite tastes. Such type of data can be useful too. This problem is solved by Pearson correlation.

User-Based CF with Pearson Correlation

There are few problems with Pearson correlation which may have affected its effectiveness for this dataset:

1. For some vectors, it is possible to have all the values in the vector equal to the average rating. This will make all the vector values as 0. This can happen for both types of vectors:
   a. The vector of the query user for whom we need to predict the rating.

b. The vector of the training user with which we are currently comparing for similarity.

Both these scenarios require custom handling. One way to handle (a) is to take the query user's average as the rating. For (b) we can take the training user's rating directly, however, it is hard to estimate a weight for that rating, because we couldn't compute the cosine similarity, which is why I did not consider this user for k-nearest users.

2. The ratings are from 1 – 5, which means there is not enough variance in data to see a correlation among movie ratings. If the ratings were from 1 – 10, it may have performed better because then the data would have been more spread out to see the relationship between the ratings.

User-Based CF with Pearson Correlation and IUF

Pearson Correlation with IUF has better MAE. This is mainly because it normalizes the rating of a movie based on number of users that have watched that movie. If less users have watched that movie, then that movie is given higher priority when computing similarity. This way we get a better representation of the similarity between users.

User-Based CF with Pearson Correlation with Case Modification

Case modification with amplification value 2 doesn't seem to improve the effectiveness for this dataset.

Item-Based CF with Adjusted Cosine Similarity

Item-Based CF has similar MAE as User-Based with Pearson Correlation, mainly because both the algorithms normalize the ratings based on user's average rating.

Other Algorithms

Here are some other algorithms that I tried.

Results

Algorithm	**MAE**
User's average and movie's average	0.7305456563650172
Weighted Slope One	0.771163171116434
Weighted Slope One + User's average and movie's average	0.7349418706549045
Bipolar Slope One	0.7565139919378395
Bipolar Slope One + User's average and movie's average	0.7289974878775486
User-Based CF with Item Based CF	0.751475141672022

Normalized Euclidean Distance	0.7939329321726938
Euclidean Distance with LogBase	0.7931880586551382
Euclidean Distance with Jaccard	0.8079102646491791

Description of Algorithms

User's average and movie's average

1. This algorithm first calculates the average ratings that all users have given to this movie.
2. Then it calculates the average rating of that query user.
3. The result is the average of these 2 averages.

The main intuition is to take into account how the user rates the movies in general. A user with low average rating means that user is not generous in his ratings. Hence, that user is more likely to rate a movie lower even if the average rating given by other users is good.

For example, assume that user1 has an average rating of 2.8 and user2 has an average rating of 3.5. This implies that user1 is a tough rater compared to user2.

If a movie has an average rating as 3.6 from all the users, then we estimate rating for user1 as:

$$R_{u1} = \frac{(2.8 + 3.6)}{2} = 3.2 \approx 3$$

Rating for user2 will be:

$$R_{u2} = \frac{(3.5 + 3.6)}{2} = 3.55 \approx 4$$

A simple average would have estimated this movie's rating as 4 for both the users. But this algorithm gives a better rating.

This algorithm is simple and efficient, yet quite effective in results.

Weighted Slope One Algorithm

Reference: https://arxiv.org/abs/cs/0702144

Below is the explanation with example. Suppose we have this data:

	Movie1	Movie2	Movie3	Movie4
User1	3	4	0	5
User2	0	0	2	3
User3	2	3	4	0
User4	3	5	2	0

First, we build a matrix where each cell quantifies how a movie is rated in general when compared to another movie. To calculate the average difference between two movies *p* and *q:*

1. For each user that has rated both the movies, take the difference of the ratings.

2. Calculate this difference for all other users that have rated both the movies and add it to a final sum.
3. Then divide it by the number of users that have rated both movie *p* and *q.*

$$diff_{p,q} = \frac{\sum_{i=1}^{n}(p_i - q_i)}{n} (\forall\ i\ users\ that\ have\ rated\ both\ p\ and\ q)$$

	Movie1	Movie2	Movie3	Movie4
Movie1	0	-1.33	-0.5	-2
Movie2	1.33	0	1	-1
Movie3	0.5	-1	0	-1
Movie4	2	1	1	0

Example:

$$d_{m1,m2} = \frac{(3-4)+(2-3)+(3-5)}{3} = -1.33$$

Now if we want to estimate user5's rating on movie3, given his ratings on movie1 and 2.

	Movie1	Movie2	Movie3	Movie4
User5	3	4	0	

We compare movie3 with movie1. Take user5's movie1 rating and add the difference from the above matrix [movie3, movie1] to that rating. Then multiply it with the number of common users that have rated both movie1 and movie3.

Similarly add the rating from movie2 and calculate a weighted average.

The intuition here is that the number of common users is acting like a weight and the rating of a movie X is adjusted according to how the query movie (the one we need to estimate) is rated in general compared to that movie X.

$$R_{m3} = \frac{(0.5+3)*2+(-1+4)*2}{2+2} = 3.25$$

Weighted slope-one + User's and Movie's average

This algorithm takes the average rating from the above 2 models.

Bipolar slope-one

Reference: https://arxiv.org/abs/cs/0702144

This algorithm optimizes the Weighted Slope-One algorithm. Please see the reference for more details.

User-Based CF with Item-Based CF

User-based CF works well when we have many items in common when computing user-similarity.

Item-based CF works well when we have many users in common when computing item-similarity.

If we use both algorithms together, then even if there is insufficient data for one, the other algorithm might have enough data to estimate a correct rating. This way we get the best of both worlds.

This algorithm takes the rating from User-based cosine similarity method and the rating from item-based CF with adjusted cosine similarity, and then takes a weighted average of these ratings to produce a final rating.

The weight ratio of user-based CF to item-based CF is assigned as 3:2. This is based on the intuition that there are a lot more items than users, so user-based result will be slightly more reliable than item-based because we can find more items in common between users than the other way around.

Normalized Euclidean Distance

This algorithm gives more weight to users that have a greater number of movies in common. So, we can normalize the Euclidean distance by dividing with the maximum possible distance between the two vectors. Let *p* and *q* be two user rating vectors with length *n.*

$$NED_{p,q} = \frac{\sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}}{\sqrt{\sum_{i=1}^{n}(maxRating - minRating)^2}}$$

maxRating and minRating are the maximum possible and minimum possible ratings respectively.

If there are a greater number of common movies between users, the denominator will be greater, making the numerator greater. Therefore, I subtracted it with 1 to make it inverse and give it higher rating if there are a greater number of common movies between users.

$$NED_{p,q} = 1 - \frac{\sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}}{\sqrt{\sum_{i=1}^{n}(maxRating - minRating)^2}}$$

For this dataset, maxRating is 5 and minRating is 1, so it can also be rewritten as:

$$NED_{p,q} = 1 - \frac{\sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}}{\sqrt{16n}}$$

After computing the similarity, prediction is done simply using the weighted average.

Euclidean Distance with LogBase

In this approach, Euclidean distance is used to calculate similarity between 2 vectors.

The similarity is computed as an inverse of the Euclidean distance. Lesser the Euclidean distance (i.e the denominator), higher the similarity.

$$ED_{p,q} = \frac{1}{1 + \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}}$$

n is the number of values in the vector *p* (or q)

On the other hand, LogBase similarity gives priority to the size of the vector. If two users have more number of movies in common, it means they are more similar because they watch and rate the same movies. By simply taking a log of number of common movies with the base as maximum number of common movies possible, we can assign them this similarity.

For example, number of movies rated by a user in test10.txt will be 10. If this user *p* and some user *q* have rated 5 movies in common (out of these 10 movies), then their similarity will be $log_{10}5$.

This approach will always give us similarity in the range [0, 1].

Formally, if *p* and *q* represent two users' rating vectors of common movies respectively. If *m* is the maximum possible movies that can match between the users (in other word: the number of movies rated by the query user) and *n* is the number of values in the vector *p* (or q), then LogBase similarity between the 2 vectors will be as follows:

$$LB_{p,q} = log_m n$$

Then we can compute the final similarity by multiplying the two similarity metrics:

$$LB_{p,q} * ED_{p,q}$$

Prediction is done using the usual weighted average.

Note: Even using just the LogBase similarity gives a similar MAE, which makes it an efficient and effective standalone metric.

Euclidean Distance with Jaccard

This algorithm tries to give additional weight to users that have more number movies in common.

Unlike the above methods, Jaccard similarity takes into account the total number of movies that a user has rated. Consider the below example where 1 represents that the movie was rated by that user and 0 represents that it wasn't:

	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6
User1	1	0	1	1	0	0
User2	1	1	0	1	1	1
User3	1	1	1	0	0	0

User1 and User2 have rated 2 movies in common – Movie1 and Movie4

User1 and User3 have rated 2 movies in common – Movie1 and Movie3

However, user2 rates almost every movie, while user3 has rated only 3 movies, out which 2 are common with user1. This means that user1 and user3 are more similar.

Jaccard similarity is represented like this.

$$J_{u1,u2} = \frac{|u1 \cap u2|}{|u1 \cup u2|}$$

In our example above, $J_{u1,u2} = \frac{2}{6} J_{u1,u3} = \frac{2}{4}$

Then we multiply Euclidean distance with Jaccard similarity to build our similarity.

Prediction is done using the weighted average.