COMP 4601 - Fall 2022 - Assignment #2 Report

Group:

- Tom Lam (101114541)
- Minh Thang Cao (101147025)

Algorithms implemented

As required, we implemented both user-based and item-based recommendation algorithms, each implemented along with MAE calculation using the leave-one-out strategy and produced analyses based on their output. For each MAE calculation for both user-based and item-based, we ran them on large sets of different settings (neighbourhood sizes, similarity thresholds, absolute similarity thresholds and the combinations of neighbourhood sizes with the thresholds). We will provide result screenshots as well as data analysis for different settings.

Minimizing the running time

Each rating prediction is a combination of calculations of the correlation coefficient (similarity) and the predicted value itself. Each of them also includes computations of smaller values, which may be the same for different predictions. Therefore, we had to find a way to avoid recomputing those values. Let *m* and *n* be the number of rows (users) and columns (items) of the given rating matrix, respectively.

- **Memoization technique** to store the average rating of each user. For each prediction, we store an array of average ratings with each index representing a user. This will prevent the recalculation of the same user's average rating when we calculate the similarities of items rated by that user since the average ratings do not change during one prediction.

  For each rating, it takes $O(n)$ time to calculate the average rating with (mn) iterations for similarity and prediction calculations, so ther running time without memoization would be $O(m \cdot n^2)$. With memoization, we only calculate average rating once before for prediction so it takes $O(mn)$ in total, then it would be $O(1)$ for each iteration. Hence, we have reduced the running time of each prediction computation to $O(mn)$.

- **Minimalized modification** of average ratings. Because of the nature of the leave-one-out strategy to find the MAE, we remove a valid rating, so the average of the corresponding user must be changed. But in that case, only that user's number of valid ratings and the total of ratings are reduced along with the removed rating. Therefore, instead of calculating that user's average rating again which takes $O(mn)$, we make it $O(1)$ by storing the sum and the rating count at the beginning, then decrease those two values and easily get the average rating when we perform leave-one-out for that user.

- **Priority queue** (heap) to acquire the top neighbours' similarity coefficients. Having all the similarities, we can do a sort algorithm which takes $O(mlogm)$ for user-based and $O(nlogn)$ for item-based. Also, in the previous labs, having constant and predetermined neighbourhood sizes (2 and 5 specifically), we can get the top neighbours with $O(1)$ iterations, and each iteration takes $O(m)$ or $O(n)$, but since we do not know the exact constant value of neighbourhood size, it takes $O(km)$ or $O(kn)$ with k is the neighbourhood size.

  Now, using a max-heap, **each** heap pop operation to remove and get a top neighbour would take $O(logm)$ or $O(logn)$. Therefore, the total running time to get the top k neighbours is $O(klogm) \leq O(mlogm)$ or $O(klogn) \leq O(nlogn)$.

- **Multithreading** to divide the work for multiple threads. This technique is not really related to algorithm optimization, but we only did this after testing and making sure that our algorithms work in a reasonable running time for each setting (neighbourhood size or similarity threshold). We took advantage of the multithreading technique by giving one thread a different algorithm, e.g. one handles user-based and one handles item-based. For each algorithm, we also did a thread queue with limits to handle each neighbourhood size or similarity threshold setting. This technique drastically improved the running time of our algorithms by a factor equal to the number of threads limited.

Implementation of the leave-one-out cross validation strategy

In general, the leave-one-out strategy is to find the mean absolute error of a valid rating by unrating it (modify to 0) then predict that rating and use the predicted value to calculate. For each setting (user-based or item-based, neughbourhood size or similarity threshold or both), basically,

we loop through all cells of the rating matrix, if we see a valid rating, we set it to 0 temporarily, recalculate average ratings, similarity and rating prediction by reusing the corresponding functions that we have implemented. Then, save it in a rating prediction matrix (table) which later be used to calculate the MAE value.

```
For each settings on neighbourhood_size or similarity_threshold:
      For each row in rating_matrix:
            for each column in rating_matrix:
                  if matrix[row][column] = 0:
                        continue;
                  temp = matrix[row][column];
                  matrix[row][column] = 0; // leave this out

                  Calculate new average_rating;
                  Calculate new similarities;
                  prediction = Prediction at matrix[row][column];
                  Set prediction_matrix[row][column] = prediction;

                  matrix[row][column] = temp;
```

Then, by applying the calculated prediction matrix on the MAE formula, we get the MAE value.

It is clear that when we remove a rating, it mainly affects the average rating of the corresponding user, then following that, the similarities and the prediction change. But since only one user's average and by just a value, we have found a way to **optimize running time** of the algorithm that applies the average rating modification in just $O(1)$ time instead of $O(mn)$ (more details mentioned in the above section).

Data analysis

Given all modifiable variables: neighborhood size, similarity threshold, absolute similarity threshold; we decided to split each algorithm into 5 combinations:

1. Neighborhood size only: uses the given number of the highest-similarity neighbors to predict the rating
2. Similarity threshold only: uses all neighbors that has similarity larger than or equal to the given threshold to predict the rating
3. Absolute similarity threshold only: uses all neighbors that has the absolute value of their similarity larger than or equal to the given threshold (this includes negative similarity) to predict the ratings
4. Neighborhood size + Similarity threshold: uses the given number of the highest-similarity neighbors that are larger than the given threshold to predict the ratings
5. Neighborhood size + Absolute similarity threshold: uses the given number of the highest-absolute-similarity neighbors that are larger than the given threshold (this includes negative similarity) to predict the ratings

Since there are 2 algorithms (User-based and Item-based) there will be in total 10 datasets. We use matplotlib to graph the recorded data.

Item-based data

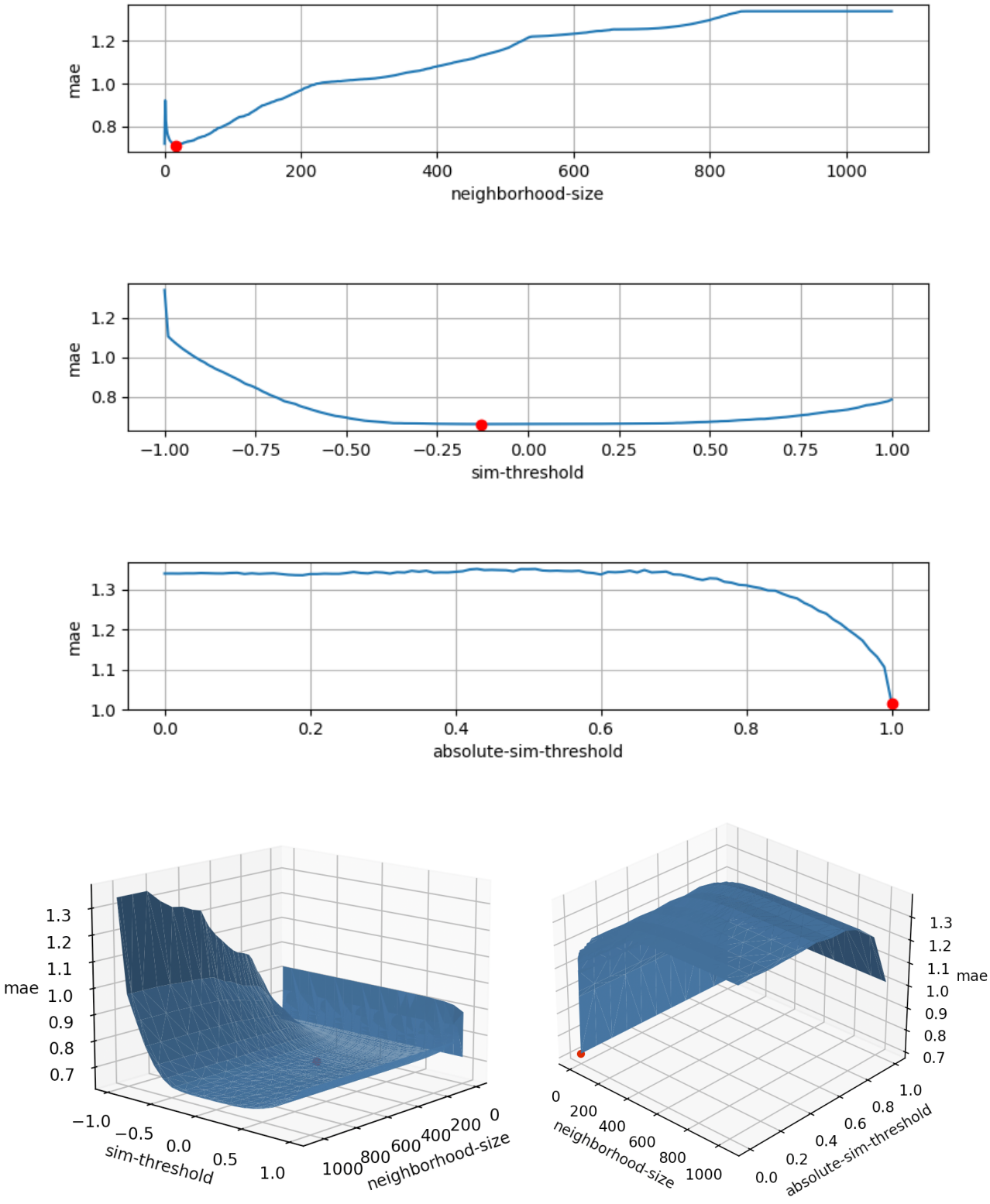


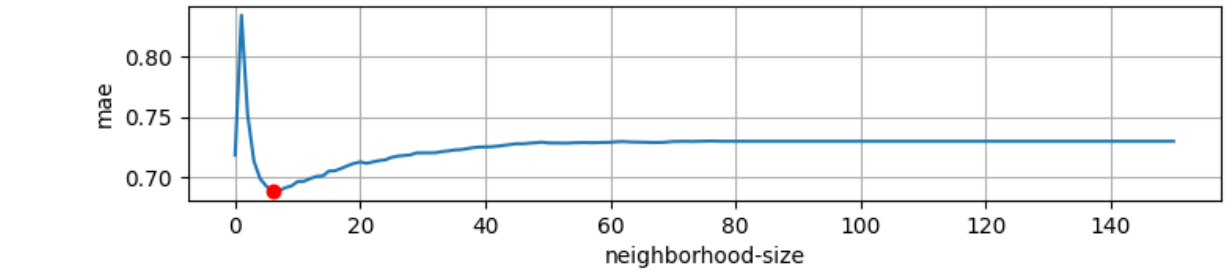
Figure 1: Item-based MAE trend graphs on different settings

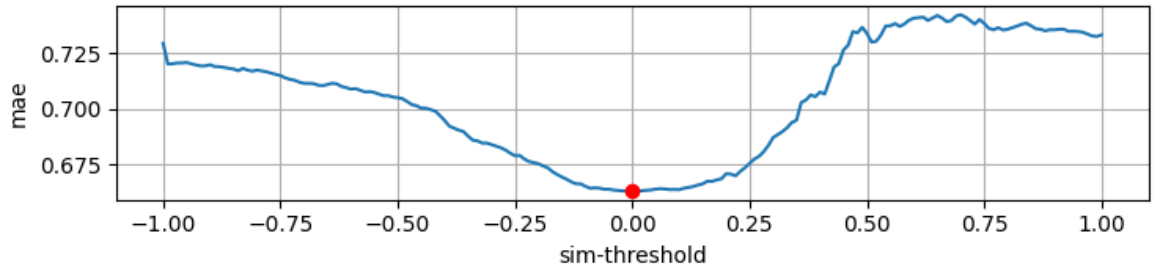
Figure 1 shows the trend of running the item based recommendation prediction using neighborhood-size only, similarity-threshold only, and absolute-similarity-threshold only.

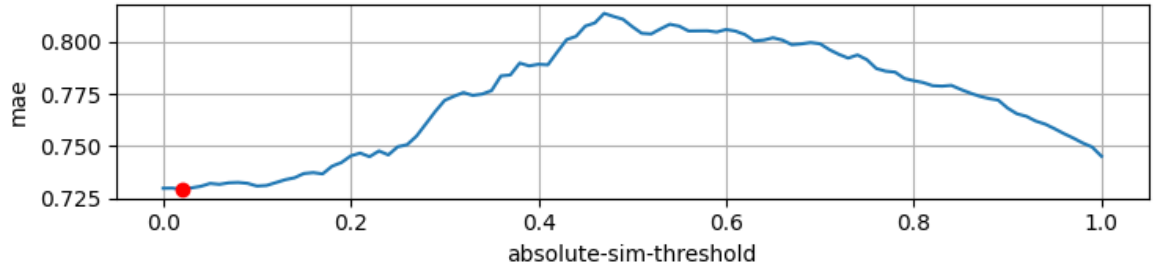
- The neighborhood-size-only dataset shows that the MAE decreases sharply as the size increases from 1. The MAE reaches the **minimum of 0.7111 when the size is 17**. The MAE increases as the size increases further. This shows that there's an optimum range for neighborhood size to produce good predictions. As the size increases above that range, the prediction will be less accurate due to the incorporation of item ratings that are not similar enough to the item under prediction.
- The similarity-threshold-only dataset shows that the MAE decreases as the similarity threshold increases from -1. The MAE reaches the **minimum of 0.6636 when the threshold is -0.13**. The MAE increases as the threshold increases further. Because 0.6636 is the lowest MAE of all the item-based combinations, using only the similarity threshold -0.13 is the best configuration for item-based recommendation.
- The absolute-similarity-only dataset shows that the MAE decreases gradually as the threshold increases. The MAE reaches the **minimum of 1.0165 when the threshold is 1.0.** This data illustrates that including negative similarity results in worse prediction for item based recommendation.
- The neighborhood-size and similarity-threshold graph shows that the MAE decreases as the neighborhood size and similarity threshold increase. The trend is mostly dominated by the increase of the threshold. The MAE reaches the **minimum of 0.6638 when the size is 304 and the similarity threshold is -0.1**. As the 2 variables increase further, the MAE increases gradually. This result illustrates that using neighborhood size with similarity threshold doesn't produce better results than only using the similarity threshold while being more complex.
- The neighborhood-size and absolute-similarity-threshold graph shows the MAE decreases as the size decreases and the threshold increases. The MAE reaches the **minimum of 0.7182 when the size is 0 and threshold is 0.0**. This again shows that including negative values produces worse predictions for item based recommendation.

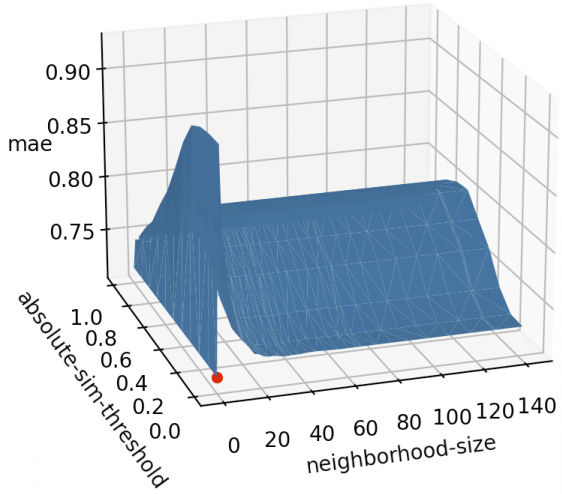
In conclusion, for item-based recommendation, choosing neighbors that have a similarity higher than -0.13 results in the best prediction. Moreover, including negative similarity results in worse prediction.

User-based data









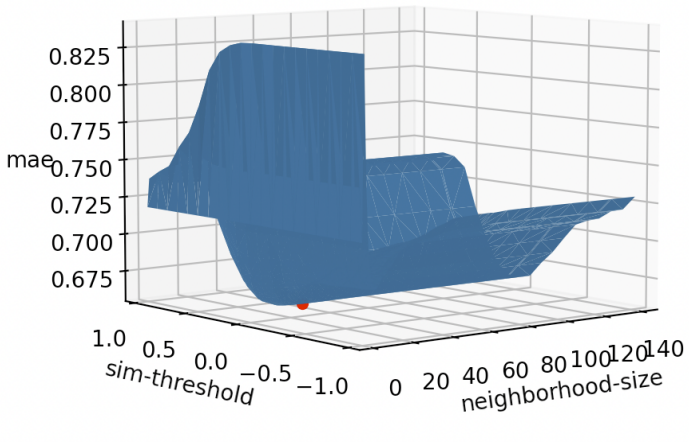


Figure 2: User-based MAE trend graphs on different settings

Figure 2 shows the trend of running the user-based recommendation prediction using neighborhood-size only, similarity-threshold only, and absolute-similarity-threshold only.

- Same as item-based, the neighborhood-size-only dataset shows that the MAE decreases sharply as the size increases from 1. The MAE reaches the **minimum of 0.6883 when the size is 6**. The MAE increases as the size increases further and **converges** with MAE around **0.73** at and after size **60**.
- The similarity-threshold-only dataset shows that the MAE decreases as the similarity threshold increases from -1. The MAE reaches the **global minimum of 0.6629 when the threshold is 0**. The MAE increases as the threshold increases further away from 0. The graph trend of this setting shows that this is one of the most accurate settings for rating prediction.
- The absolute-similarity-only dataset shows that the MAE increases gradually as the threshold increases until it reaches **0.45**. The MAE then decreases slowly as the threshold gets larger. It reaches the **global minimum of 0.7291 when the threshold is 0.02**
- The neighborhood-size and similarity-threshold graph shows that the MAE decreases as the neighborhood size decreases and similarity threshold gets closer to 0.0. The MAE reaches the **minimum of 0.6627 when the size is 26 and the similarity threshold is 0**. As the neighborhood size increases further and the similarity threshold gets away from 0.0, the MAE increases gradually.
- The neighborhood-size and absolute-similarity-threshold graph shows the MAE decreases as the size decreases and the threshold decreases. The MAE reaches the **minimum of 0.7182 when the size is 0 and threshold is 0.0**. This again shows that including negative values produces worse predictions for item based recommendation.

In conclusion, for user-based recommendation, choosing neighborhoods of sizes 26 that have similarity threshold higher than 0.0 results in the best prediction. Moreover, including negative similarity results in worse prediction.

Running time of the algorithms

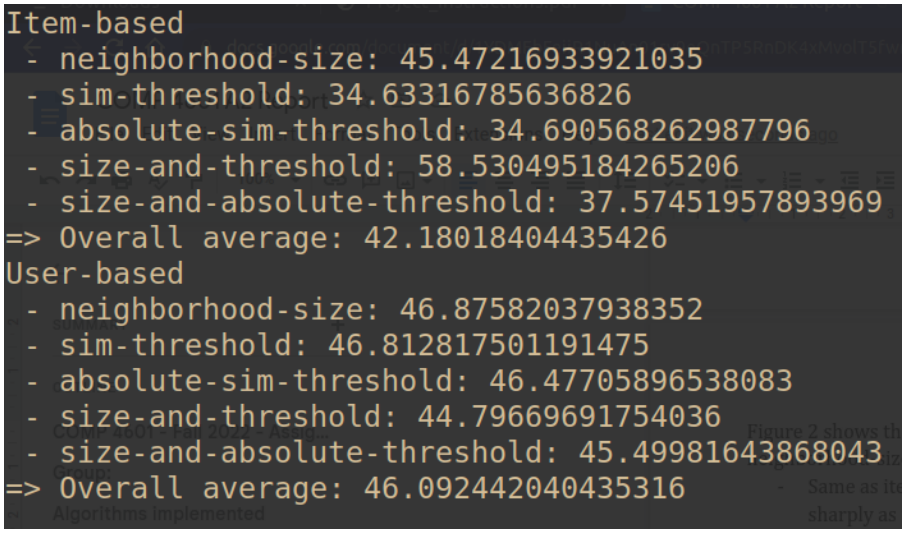


Figure 3: User-based vs Item-based running time for each settings

Based on the **average running times** obtained from Figure 3, the **item-based** model is **faster** in general. Specifically about each setting, the **neighborhood size** running time difference between both is quite **small** but for others, the differences are at least 10 seconds in which item-based is faster. Only about each setting, the **neighborhood size** running time difference between both is quite **small** butr the combination of size and absolute threshold, the user-based model is faster (44.8s compared to 58.5s).

Application on a real time online movie recommendation system

Based on only the accuracy, the user-based algorithm produces the most accurate prediction, with the MAE of 0.6627 in 44.8 seconds. However, the item-based algorithm produces only slightly higher MAE of 0.6636 (0.14% higher) with considerably faster running time of 34.6 seconds (23% faster). Therefore, based on experimental data, the item-based is the overall winner.

Realistically, user-based recommendation might be a better fit for applications. User based neighbors are chosen based on the set of all viewers that rate the same movie while item based neighbors are chosen based on the set of all movies that are rated by the same viewer. In practice, applications usually have exponentially more viewers than movie titles, hence, can predict more accurately using user-based recommenders.

Effects of the number of ratings by users

If there are more users with more reviews, the rating predictions for both item-based and user-based models would get more accurate as there is more data. This suggests that the algorithm would have more evidence to perform the prediction accurately. On the other hand, if there are less users or the users have less reviews, there is less data and there are less clues to produce accurate predictions as the algorithms have to use less related data sets instead of getting the top similar ones.

The given data set for this assignment has more items than users. Hence, it proved that with more data, it is easier to find top item neighbours that have high similarities and produce more accurate data. For the real-time online movie recommendation system above, there should be a lot more users, so a user-based model would be a better choice. In general, we can conclude that the performance of a recommendation algorithm depends heavily on the data set it uses.