COMP4601 A2

By: Minh Nguyen - 101154921, Leo Xu - 101149896, Kenji Isak Laguan - 101160737

1) Which algorithms have you implemented? What are the details of these algorithms?

We implemented both the user-based and the item-based recommender system algorithms. Just as a set up we have loadTestData function to read in and parse the data from the assignment2-data.txt into a usable matrix for our code. For both JavaScript files we implemented both types of recommender systems which are named recommenderItemBased.js and recommenderUserBased.js. Starting with User Based we used our implementation of it from the previous Lab 8. Which has a calcAllMean which takes in the matrix and calculates each Row/User's average, as well as keeping track of the counts of non-negative values for that row to be able to recalculate the row/user's average in O(1) time which is later explained in detail on how we minimize the runtime for it in a later question. This function implementation is the same for both systems.

Next is the calcPCC function which will calculate all the Pearson Correlation Coefficients given the Row/User Index we are trying to predict a value for, the matrix, and all the means of the users as that is all the information we need to do so. To follow along with the Pearson Correlation Coefficient equation, we iterate through the whole matrix except the User's Row(in the sense so we don't compare the ratings in that User's Row with itself), then we check if both the User's Rating and the Other Rating is valid (not a 0) and if satisfies that condition we subtract their rating's with their corresponding mean and multiply them together, then add it to the numerator counter. For the denominator, we would subtract their rating's with their corresponding mean to the power of 2. We keep track of the denominator by separating the User's denominator and the Others. We can square root the User's denominator accumulator multiplied by the Other's denominator accumulator that's square rooted. Now if the denominator is still 0 we set the PCC to 0, otherwise we can set it to the numerator divided by the denominator, and we can set another attribute of which row index we compared that User to be saved. We then push that calculated PCC tuple of info into an array that we return back after we finished iterating through the whole matrix and calculated all the PCCs.

Next calcPred is the function we use to calculate the predicted rating. Since we're experimenting with top-K based or threshold based we have supported those as well by boolean parameters passed in as thresholdBased and includeNeg which if thresholdBased is true then we pass in a valid threshold value, else if it's false, we pass in a valid k value for top-k neighbourhood size and boolean includeNeg to determine if we want to include negatives or not for this top-k based run.
So to calculate the predicted value following the user based pred equation. We iterate through all the PCCs and accumulate the numerator and denominator. The denominator is just the sum of the similarities so the PCC, and the numerator is the accumulation of each of the similarities multiplied by that Other user's actual rating subtracted by its mean. So we only add those to the accumulators depending on if its threshold based or top-k based(including or excluding negatives). If its threshold based then we check if the current PCC were iterating is above the

given threshold and if that Other user's rating for that PCC is a valid rating(non 0). We only take the PCCs that are above the threshold so since the given AllPCCs are sorted once we reach a PCC lower than the threshold then the rest after will also be lower than, and we break out of the loop. If it was top-k based we check if the current PCC is either above 0 if we dont include negatives, and if that Other user's rating for that PCC is a valid rating(non 0). If we include negatives then we dont check if the PCC is above 0 and all PCC are valid, we would just check if that Other user's rating for that PCC is a valid rating(non 0). We only take the top-k neighbours so we break out of the loop once we hit that count. And so we calculate the predicted value by dividing the numerator by the denominator accumulators and added by the mean for the User were trying to predict.

For ItemBased essentially the calcPCC and calcCosSim are identical but instead of the accumulating be the set of products both user a and b has rated, itll be the set of users that has rated both product a and b. So the function for calcCosSim is similar but converted for itemBased and instead of save the user Index(row) we save the itemIndex(column). ItemBased's calcPred is also similar to userBased version but again just with itemIndex instead of userIndex and using COS similarities instead of PCC. We chose to not pre calculate the difference in rating and average rating for the whole matrix as it was assumed to be a longer runtime since the average would change for a row so recomputing the differences in that row would need to be done for each value in it.

Lastly calcMAE applies the leave one out strategy to evaluate the recommender systems we've implemented. So we calculate all the means and counts so that we only have to recalculate the average for the row in O(1) time rather than recalculating for the whole matrix each time we do a prediction, then we iterate through the whole matrix and apply the leave one out strategy and predict a valid rating. So for each valid rating(non 0) we save that value, set that cell to 0 in the matrix, recalculate the mean for that row, calculate all the PCC or COS similarities, sort those similarities, then calcPred to calculate the predicted value. If that predicted value ends up as not a valid number, then we make a best guess and the predicted value will be set as that Users average rating. Then we can reset the original values for both the user average and the cell to the actual rating. Then set the lower and upper bounds of the predicted rating if its under 0.5 to 0.5 and over 5 to 5. To calculate the MAE for the whole matrix, it will be the sum of all predicted values subtracted by their actual values, divided by the total number of predictions weve made after we finished with the whole matrix.

2) How have you minimized the runtime of your implementations?

We are able to reduce the runtime of our implementation, by reducing the recalculation of the averages for each user, since when we're predicting a value using the leave one out strategy. We would calculate all the averages of the users before predicting a value, and before we iterate through the matrix. So when we found a rating that is valid and not a 0. Instead of recalculating everyone's average through each prediction we make, which would be a horrible runtime. We can recalculate the average for just the row/user that we are currently trying to make a leave one out, prediction on. So Instead of iterating through and recalculating the

average of that row in O(n) time (where we have a matrix of size m*n.), by reiterating adding the total up again then dividing by total number of non-negative values. We can reduce the run time of recalculating that row/user's average to be optimized to O(1). We can do this by storing the number of non-negative values we divide by for that row to get the average along with the average as well, so we would have a tuple of info. Then we can recalculate the average for that row in O(1) time by multiplying that row's average by the count of that row's non-negative values (which is stored in allCounts[row]) then subtracting that by the actual value we are currently trying to predict. Then divide that by the count of that row's non-negative values - 1. Since doing the leave one out strategy, we set the value we're trying to predict as 0. So we remove the row's number of non-negative values by 1. And so we properly recalculated the average for the leave one out strategy in O(1) to reduce the runtime.

When calculating the predicted value using the calcPred function, we were able to also minimize the runtime for it. For threshold based for both user and item based implementations, we break out of the iterating loop that iterates through all the sorted similarities that was sorted highest to lowest. We break out of it at the moment we reach a similarity that's not above the threshold as all the remaining non visited similarities will also be below the threshold as the list of similarities is sorted. For top k based, we break out of the iterating loop when we've reached the correct number of top k neighbors, so the function will have a runtime of O(k) instead of O(n) where it fully iterates through the list.

3) How have you implemented the leave-one out cross validation strategy?

We implemented the leave-one out cross validation strategy in our calcMAE(startingMatrix) function by going through our starting matrix values and checking if any of the values are not equal to 0(they are valid ratings) and if they are, we would save the actual rating/value then set the value of that cell in the starting matrix to 0 to leave one out. We would then recalculate RatingAvg for just that User/Row after rating removed in O(1) time, since removing a value would change the average. We would then calculate all the PCCs or COS and sort the similarities from highest to lowest values. We then calculate for the predicted values by using our calcPred function and checking the predicted value is Nan or Infinity(it isnt a finite number), and if it is then the best guess instead would be the User Rating Avg for the predicted value. We would then reset to original user values after performing the leave one out strategy and check if the predicted value is in the upper or lower bounds setting 0.5-5. Lastly, we can add the predicted value subtracted by its actual value to the numerator accumulator for calculating the MAE, and accumulate the number of total predictions made.

4) Is user-based or item-based nearest neighbour recommendation more accurate for this Data?

For this data, we've sorted all runs of different parameters, for top k based, k was set to 0 to 100 in increments of 10 for each run and ran for both including and excluding negatives. For threshold based, it was values from -1 to 1 in increments of 0.1. This was experimented for both

user and item based recommendation implementations.

In our implementation we found out item based to be more accurate than user based, comparing the lowest MAE numbers we got for item based to be `0.6652588710715049 =` to 0.65 with it being top K based, with the k value being 100 highest similarity neighbours, and including and excluding negatives still achieved the same MAE. While user based, the lowest MAE we've achieved was `0.662626732 =` 0.662 with it being top K based with the k being 24 highest similarity neighbours and only using positive similarities.

5) Is top-K (i.e., selecting the K most similar users/items) or threshold-based (i.e., selecting all users/items with similarity above some threshold X) more accurate for this data? **FIX**

From the provided results, it seems that the threshold-based approach for user-based recommendation yields lower MAE values compared to the top-K approach. Specifically, at a threshold of -0.1, the user-based MAE is 0.665, which is lower than the lowest MAE achieved using the top-K approach (MAE of 0.6626 with a neighborhood size of 24).

From the provided results, it seems that the top-K approach for user-based recommendation yields lower MAE values compared to the threshold-based approach. Specifically, at a threshold of -0.1, the user-based MAE is 0.665, which is higher than the lowest MAE achieved using the top-K approach (MAE of 0.6626 with a neighborhood size of 24). For item-based recommendation, the top-K approach with a neighborhood size of 380 yields lower MAE values compared to the threshold-based approach. The lowest MAE for the top-K approach is 0.663927249201704, the neighborhood size of 380, while the lowest MAE for the threshold-based approach is 0.722629591 at a threshold of 0.1.

In summary, based on the provided results, for user-based recommendation, the threshold-based achieved the lowest MAE of 0.662966 at a threshold of 0,and the top-K approach got the best accuracy at neighborhood size of 24 with MAE of 0.662627. On the other hand, the threshold-based achieved the lowest MAE of 0.722629591 at a threshold of 0.1,and the top-K approach got the best accuracy at neighborhood size of 380 with MAE of 0.663927249201704. Between the two, the user-based recommendation had the better performance using top-K with a neighborhood size of 24.

However, the choice between top-K and threshold-based approaches may depend on the specific characteristics of your data and the requirements of your recommendation system.

6) Which parameter values produce the most accurate results for this data (e.g., is 2 neighbours best? 10? 100? a threshold value of 0? 0.5?, etc.)? How does the prediction accuracy change as the parameter values change?

As we can see from these tests and observe the User-Based Algorithm MAE Table and the Item-Based Algorithm MAE Table at the bottom, the algorithm that would eventually provide us

with the most accurate result would be item-based algorithm because as the neighborhood value becomes bigger the MAE values become lower and more accurate. The lowest MAE values are when the values are set to 400 as it is the convergence point. The threshold values also did impact the MAE values heavily, but if we are only modifying the MAE values.

The algorithm that provided us the most accurate results for this set of data is the item-based algorithm that gives us the lowest MAE value when the neighborhood values are set from 10-14. The threshold values also did impact the MAE values heavily, but if we are only modifying the MAE values. The more negative the threshold value becomes, the bigger and more inaccurate the MAE value becomes as well. Looking at the provided data for different parameter values (threshold and neighborhood size) and their corresponding Mean Absolute Error (MAE) for both user-based and item-based approaches, we can derive insights into the most accurate parameter values.

Examining the dataset's MAE values across various parameter values for user-based and item-based recommendation approaches, specific patterns emerge to guide optimal parameter selection. For user-based recommendations, threshold values around 0 or slightly above (0.1) tend to yield superior accuracy, while employing a neighborhood size of 23 or higher correlates with improved predictive performance. Conversely, in item-based recommendations, threshold values of approximately 0.1 showcases a lower MAE, indicating better accuracy, particularly when coupled with smaller neighborhood sizes ranging between 380 and 400. These insights suggest that for user-based recommendations, a balance between threshold proximity to 0 and a substantial neighborhood size is beneficial, while item-based recommendations benefit from a modest threshold and a more focused neighborhood size. Nonetheless, the final parameter selection should be tailored to the specific nuances of the dataset and the unique demands of the recommendation task to optimize accuracy and effectiveness.

7) Does including negative correlations (e.g., top-k based on absolute value of correlation) improve the results?

The negative correlation for our user-based algorithm would always give a bigger MAE value and would provide us with a worse result than when we do not use the negative correlation. For the item-based algorithm, we found that the negative correlation had the same impact on the MAE values that we found comparatively to the user-based algorithm. Therefore, we had found out that the negative correlation on both the user-based and item-based recommender algorithm negatively impacts the results by making it worse in this data set. Stick to non-negative correlations seems to be giving the more accurate and proper results.

8) How long does prediction take for each algorithm/parameter combination? Is one solution faster than the other? Is this expected based on the algorithms or is it specific to your Implementation?

The analysis of runtime across different algorithmic setups for user-based and item-based recommendation systems reveals a stable performance pattern within each method. In user-based recommendations, adjustments in neighborhood size or threshold values consistently maintain the runtime without significant deviations. Interestingly, these variations don't notably affect the prediction time, indicating a consistent computational demand irrespective of the chosen parameters. Similarly, the item-based recommendation framework showcases stable runtime characteristics across various parameter variations, maintaining relatively constant execution times despite changes in neighborhood sizes or threshold values.

Remarkably, in this specific implementation, the user-based recommendation algorithm outpaces its item-based counterpart significantly, exhibiting a substantial disparity in runtime. The user-based approach executes computations roughly 2.5 times faster than the item-based algorithm. The runtimes for each for both the User-Based Runtime Table and the Item-Based Runtime Table can be found at the bottom. Although user-based algorithms simplify computations based on user-user similarity, while item-based calculations involve more intricate item-item similarity computations, both methods demonstrate consistent and predictable runtime behaviors across parameter configurations. Despite this notable disparity in runtime, the stability of runtime across parameter adjustments highlights a consistent performance within each approach, underscoring their reliability across various parameter values.

This is expected because it is due to the fact that user-based recommendation algorithms tend to be faster than item-based recommendation algorithms. User-based approaches often involve computations based on user-user similarity, which can be computationally simpler compared to item-based approaches.However, the speed difference might vary depending on the specific implementation, dataset characteristics, and the complexity of similarity calculations involved in the algorithms.

9) Based on your analysis and knowledge of the algorithms, which algorithm/parameter combination would you use for a real-time online movie recommendation system? Provide some arguments in favor of this conclusion based on your experimental results and the computational requirements for the algorithm. You should also consider the benefits/drawbacks of each algorithm in your comparison (e.g., what values can be precomputed? how will this affect a real-world application?).

In a real-time scenario, considering computational requirements, scalability, and the nature of user interactions, an item-based recommender system might be more favorable as there would be a mass amount of movies that would require a larger neighbourhood size. Precomputed item-item similarities and better scalability could make it more suitable for real-time recommendation generation while still providing reasonably personalized suggestions. However, the specific choice should be validated through experimentation and performance evaluations on the dataset and system requirements. Comparing both user-based and item-based recommender system algorithms we would need to take into account the sparsity if the dataset is extremely sparse, item-based approaches might be more suitable due to more stable

item-item relationships. Scalability is also something we need to consider because for large user bases, item-based systems might be more scalable as the computation scales with the number of items, not users. If personalized recommendations are crucial and users have substantial interaction histories, user-based systems might perform better than item-based recommendations. The pros of item-based recommender systems exhibit robustness in handling sparsity within datasets, as item-item similarities tend to be more stable than user-user similarities. They offer scalability advantages, performing well with an increasing number of users since the number of items is typically fewer. Additionally, precomputed similarities between items enable reduced real-time computational requirements during recommendation generation. However, these systems might lack personalization by providing more generic recommendations based on item similarities rather than capturing specific user tastes or preferences. Moreover, they may face challenges when recommending new or unpopular items due to the reliance on item-item similarities, which require sufficient user interactions to generate reliable recommendations for such items.

10) How will your solution be affected by users with more/less reviews?

The impact of users with varying review counts on the recommendation system can affect the optimal parameter selection and prediction accuracy. Users with more reviews tend to influence the system significantly, potentially altering similarity calculations and neighborhood formations.

For the User-Based Recommendations algorithm, users with more reviews might dominate similarity calculations, affecting the neighborhood formation. If a few users have substantially more reviews compared to others, they could disproportionately influence the recommendations. As such, the optimal neighborhood size and threshold may be skewed towards accommodating these users, potentially impacting the accuracy of predictions.

For the Item-Based Recommendations algorithm, users with more reviews can also impact item-based recommendations. If certain users extensively review specific items, it might lead to more accurate predictions for those items due to better similarity calculations. Conversely, items with fewer reviews by a variety of users might have less accurate predictions.

In essence, users with differing review counts can introduce biases in the recommendation system. If a small subset of users dominates the dataset with numerous reviews, they may heavily influence the recommendations, potentially affecting the optimal parameter selection. Addressing this issue might involve techniques like normalization, weighting, or incorporating mechanisms to handle varying review counts to ensure fair and accurate recommendations for all users and items. Regularization techniques or considering review frequency might be necessary to mitigate the impact of users with significantly more or fewer reviews on the recommendation system's performance.

User-Based Algorithm MAE Table:

top-K neighborhood Size	MAE	Threshold	MAE
0	0.718213	-1	0.725854
2	0.752394	-0.5	0.708737
23	0.662694	-0.1	0.665204
24	0.662627	0	0.662966
25	0.662633	0.1	0.663537
30	0.662681	0.5	0.735352
50	0.734864	1	0.718816
100	0.736264		

Item-Based Algorithm MAE Table:

top-K neighborhood Size	MAE	Threshold	MAE
0	0.718212662087576	-1	1.200155175
10	0.703312341445305	-0.5	0.821038365
50	0.672577550654233	-0.1	0.728556016637
100	0.665258871071504	0	0.723918164
350	0.663943763815174	0.1	0.722629591
380	0.663927249201704	0.2	0.722654575027
390	0.663939172689000	0.5	0.734068159044
400	0.663932075729804	1	0.725733638495
800	0.663939084459184		
1000	0.663939084459184		

User-Based Runtime Table:

neighbourhood Size	User based runtime (milliseconds)	Threshhold	User based runtime (milliseconds)
0	4797.75	-1	4479.41
5	4679.343	0	5079.63
15	4709.64	0.5	4945.41
25	4686.03	1	5024.97
50	4644.62		
100	4628.72		

Item-Based Runtime Table:

neighbourhood Size	Item based runtime (milliseconds)	Threshhold	Item based runtime (milliseconds)
0	12183.0356	-1	13758.5305
5	12156.3917	0	12067.089
15	12648.7579	0.5	12778.1291
25	12506.3607	1	12530.0565
50	15995.2506		
100	13571.6349		