CMP2003 PROJECT

In this project, I sought to develop a solution for predicting ratings for movies based on user data. My approach involved using a data structure that maps objects to their IDs on an array, with each object in the array having a "next_object_index" integer that allows me to skip gaps between objects without having to check them one by one. This data structure has the characteristics of a hashmap and a linked list.

To apply this approach, I first read in the lines from the "train.csv" file and store them in a temporary object that holds two integers and one float. I then sort these lines by user ID using heap sort, which has a time complexity of O(n logn) and is therefore relatively efficient. Once the lines are sorted, I am ready to store them in my data structure (which I have named "storage"). I also store a separate "storage" in the user class to store ratings. The ratings are stored as "rating_class" objects.

In addition to predicting movie ratings, I also covered finding the top 10 users and items based on their rating counts. To do this, I used similar methods such as heap sort and a temporary object. For the heap sort, I modified it to be a max heap sort in order to rank the users and items by their rating counts in descending order. This feature provides a useful summary of the most highly rated users and items in my dataset.

Once all of the data has been stored in the appropriate data structures, I read in the "test.csv" file to obtain the questions for which I need to predict ratings. For each line, I extract the question ID, user ID, and item ID, and pass these values to the "predict" function to calculate the predicted rating.

In the 'predict' function, I first collect all of the item IDs that the question user has rated. This is done to minimize the time it takes to compare two users, as I need to repeat this operation for every other user who has watched the question movie. I then create an array to store the calculated similarities and common rating counts between the users. To determine the similarity between two users, I tested various measures, including the cosine and Manhattan measures. After evaluating the performance of my model on the Kaggle site, I found that the Manhattan measure gives better results than the cosine measure in my case. This is because the cosine measure only calculates the angle between vectors and does not take the values themselves into account. By contrast, the Manhattan similarity measure takes the values into account and gives more accurate results. While my performance on the Kaggle site is not among the highest, it is still better than the performance achieved using the cosine measure.

Next, I iterate over all of the users. If a user has watched the question movie, I pass the required values to the "calc_manhattan" function to calculate the sum of differences between the common rated items' ratings, divided by the maximum error. The maximum error is calculated as the maximum possible difference between any two ratings, which is 5.

After iterating over all of the users, I sort them by their similarities using heap sort. I then calculate the weighted average of the most similar 130 users (which has been found to be an optimal value on the Kaggle site) to obtain the final prediction. To avoid the overhead of repeatedly opening and closing the file for each prediction, I store the predictions in a string in the correct form. This allows me to save time and improve the efficiency of my solution. Once all of the predictions have been made, I can then write the string to the file in a single operation. The total running time of my solution is around 28 seconds, which is quite good result for me.

For finding top 10 users and items I used similar methods.

To test my solution, I used the Kaggle site. I evaluated the performance of my model using various metrics and found that it achieved good results. However, I did encounter some challenges, such as finding the optimal value for the number of most similar users to consider. I addressed these challenges by conducting sensitivity analysis to determine the optimal number of most similar users.

In conclusion, I have developed a solution for predicting movie ratings based on user data that achieves good results. In future work, I could consider expanding the scope of the data to include more users and movies, or exploring alternative algorithms for calculating similarities between users.