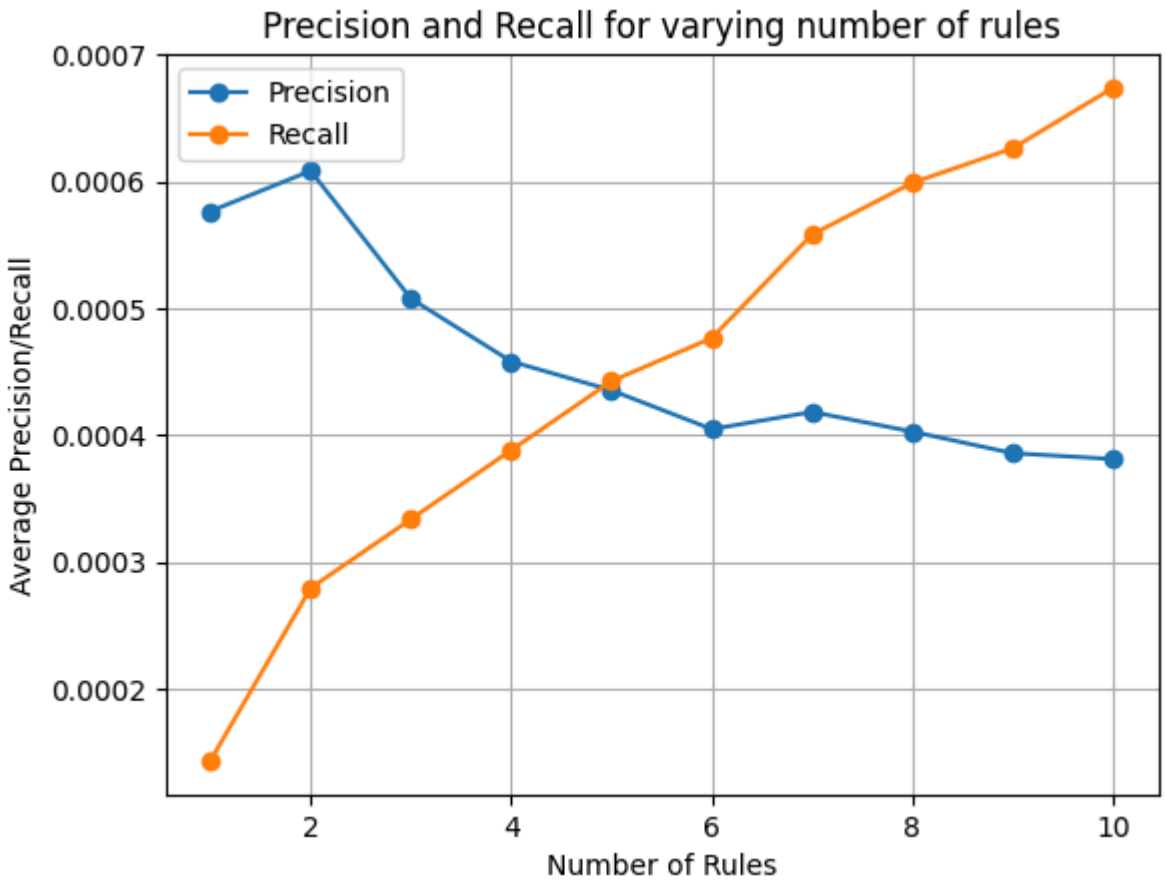
DATA ANALYTICS-1 ASSIGNMENT-3 REPORT

Hemanth Reddy [2023201058]
Sathvika Pericharla [2021101030]

Justification of Selection of Algorithm:

We chose the Apriori algorithm. The Apriori algorithm is a widely used algorithm in association rule mining for discovering interesting relationships and patterns in large datasets.

Average Precision Recall vs Rule Count



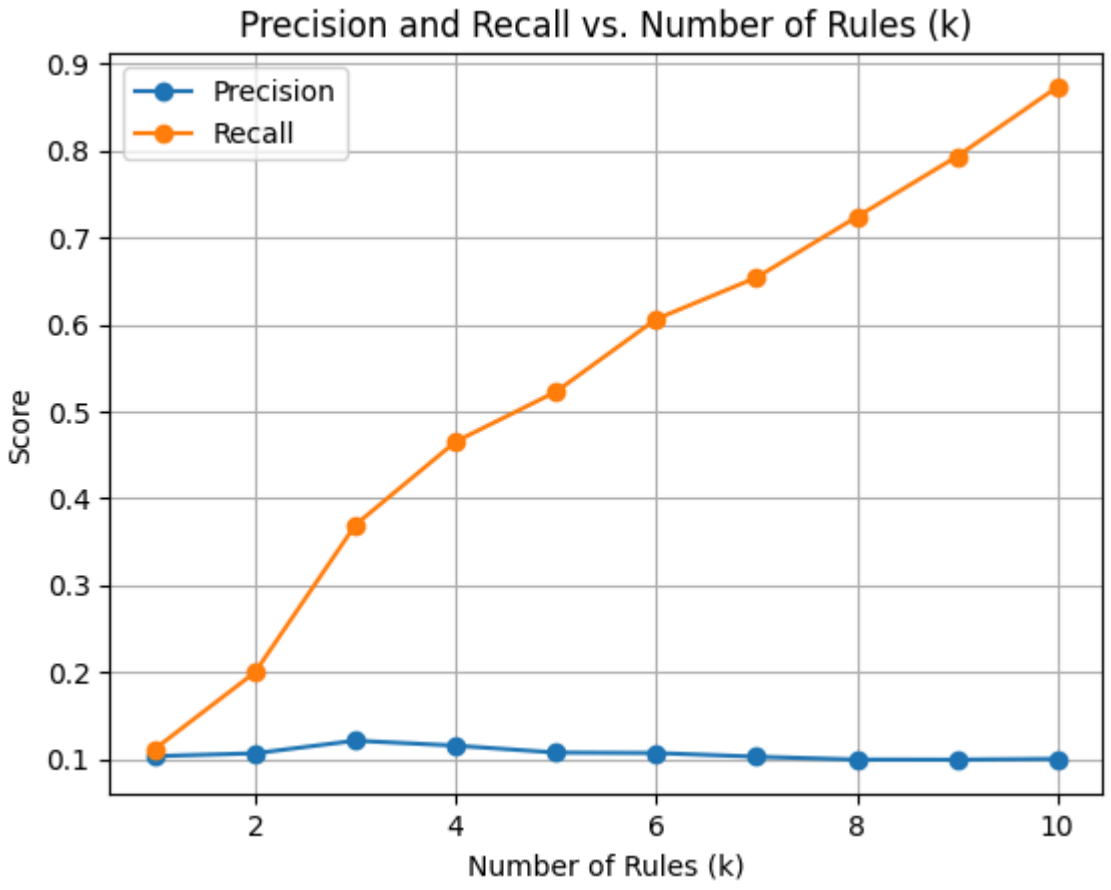
Decreasing Precision: On increasing the number of rules (k), the average precision is likely to decrease. This is because with more rules, we are

recommending a larger set of items, and some of those recommendations may not be relevant to the user. As a result, the precision, which measures how many of the recommended items are relevant, tends to decrease as k increases.

Increasing Recall: On the other hand, on increasing the number of rules (N), the average recall is likely to increase. This is because with more rules, we are recommending a larger set of items, which is more likely to include some of the relevant items from the test set. Recall measures how many of the relevant items are included in the recommendations, and as we provide more recommendations, we are more likely to cover a larger portion of the relevant items.

Trade-off Between Precision and Recall: The graph will likely show a trade-off between precision and recall. When we have fewer rules (lower k), we have a higher precision but a lower recall, and when we have more rules (higher k), we have a higher recall but a lower precision.

Precision and Recall vs k for 20 sample users



RECOMMENDATION SYSTEM

Data Preprocessing

1) **Data Import:** We used dataset from ratings.csv file
2) **Filtering:** Selected only rows which have rating greater than 2 , considering only positively rating movies
3) **User selection:** We select users who have rated more than 10 movies, ensuring we have a sufficient amount of data for building the recommendation system.
4) **Train-Test Split**: We split data into training and testing sets with an 80-20 ratio by using the library "from sklearn.model_selection import train_test_split"

Functions Implemented:

1) **Support :** support calculation, for every itemset (first_level_frequent_movies,third_level_frequent_itemsets and third_level_frequent_itemsets)

2) **Generate_frequent_temsets:**
   The system generates frequent itemsets of movies using the Apriori algorithm. It starts with single movies (first_level_frequent_movies) and gradually expands to larger itemsets (second_level_frequent_itemsets, third_level_frequent_itemsets, and l4_movies) based on a minimum support threshold (minsup). Only itemsets with support greater than or equal to minsup are considered frequent.This function generates itemsets of size given by the user or according to the itemsets size we want.

   - **For itemset of size 2:**

     1. It initializes a variable named candidates to store the generated itemset candidates.

2. Inside the list comprehension, it iterates through two loops, represented by the variables x and y, over the elements in the first_level_frequent_movies list.

3. The condition if x < y ensures that duplicate pairs are not generated, which would result in redundant candidates.
4. For each combination of x and y, a tuple (x, y) is created and added to the candidates list. This tuple represents a candidate itemset of size 2.

- **<u>For itemset of size >2:</u>**

1. The outer loop iterates over the index i of the item_sets list, and the inner loop iterates over the index j of the item_sets list. These nested loops are used to compare and combine item sets.
2. The condition if sorted(item_sets[i][:-1]) == sorted(item_sets[j][:-1]) checks if the first itemset_size - 1 item in the i-th and j-th item sets are identical. This condition ensures that the two item sets have the same prefix.
3. If the condition is met, it means that the two item sets share a common prefix and can potentially be combined. The code then constructs a new candidate item set (candidate) by taking the prefix from one of the item sets (item_sets[i]), and appending the last item of each of the two item sets (item_sets[i][-1] and item_sets[j][-1]) to the candidate.
4. The constructed candidate is then added to the candidates list, which represents a candidate itemset of the specified size.

- **Support calculation:**
    1) For every candidate in candidate sets we will calculate support count if it is greater than minsup(0.1) we will add that candidate itemset in the frequent_itemsets
    2) Finally we will return frequent itemsets generated.

3) **Generate_association_rules:** The system generates association rules from frequent itemsets by calculating confidence values. Association rules consist of antecedent and consequent items. Rules with confidence greater than or equal to a minimum confidence threshold (minconf) are selected.

1) It takes two input parameters: item_list, which is a list of itemsets, and minconf, which is the minimum confidence threshold used to filter the association rules.

2) It initializes an empty list called ass_rules to store the generated association rules.

3) It iterates through each item set x in the item_list.

4) For each itemset x, it calculates the support s of that itemset. Support is a measure of how frequently an itemset appears in the dataset.

5) It then enters a nested loop that iterates through the indices of items in the itemset.

6) Within the nested loop, it constructs two sets of items:

- antecedent: A set of items that excludes the item at the current index. This represents the left-hand side of a potential association rule.
- [x[i]]: A set containing only the item at the current index. This represents the right-hand side of a potential association rule.

7) The code calculates the confidence of the potential association rule by dividing the support of the itemset x by the support of the antecedent. Confidence is a measure of how often the right-hand side item occurs when the left-hand side items are present.

8) If the calculated confidence is greater than or equal to the specified minconf, the potential association rule is considered strong enough to be included. The details of the association rule, including the antecedent, consequent, support, and confidence, are added to the ass_rules list.

9) After processing all itemsets, the function returns the list of generated association rules that meet the minimum confidence threshold.

Part1:

We took minconf as 0.1 and minsup as 0.1, and we created first_level_frequent_movies, second_level_frequent_itemsets and third_level_frequent_itemsets using the function "generate_frequent_itemsets". Finally created association rules from the third_level_frequent_itemsets and third_level_frequent_itemsets movies using the function " generate_association_rules"

Part2:

We generated two set of lists, based on the order of their support and confidence and we took common rules which are appearing in both lists

Part3:

Calculating precision and recall values for a recommendation system. It is doing this for different values of k (ranging from 1 to 10) to evaluate the performance of the recommendation system at various levels of recommendation depth. Here's a breakdown of how the code works:

1) The code initializes two empty lists, precision_avgs and recall_avgs, to store the average precision and recall values for different values of k.

2) Enters a loop that iterates from k = 1 to k = 10, representing the recommendation depth.

3) Inside the outer loop, there is another loop that iterates through users in the train_set.

4) For each user, we are retrieving their training data (train) and checking if the user is also present in the test_set.

5) If the user is in the test_set, it retrieves the test data (test) for that user.

6) Initialized variables recall_sum and precision_sum to keep track of the sum of recall and precision values for the current k and user.

7) Initialized an empty list called recommendation to store recommended items.
8) For each item x in the user's training data, iterating through association rules (conf_rules) to find items that are associated with x. and appending these associated items to the recommendation list, limited to a maximum of k items.

9) We created a hitset to keep track of the items in the recommendation that are also present in the user's test data. This represents the set of recommended items that the user actually interacted with.

10) We are calculating the recall value by dividing the size of the hitset by the size of the user's test data.

11) Then calculated the precision value as the size of the hitset divided by the size of the recommendation. If the recommendation is empty, precision is set to 0.

12) The recall and precision values are added to the recall_sum and precision_sum for the current user.

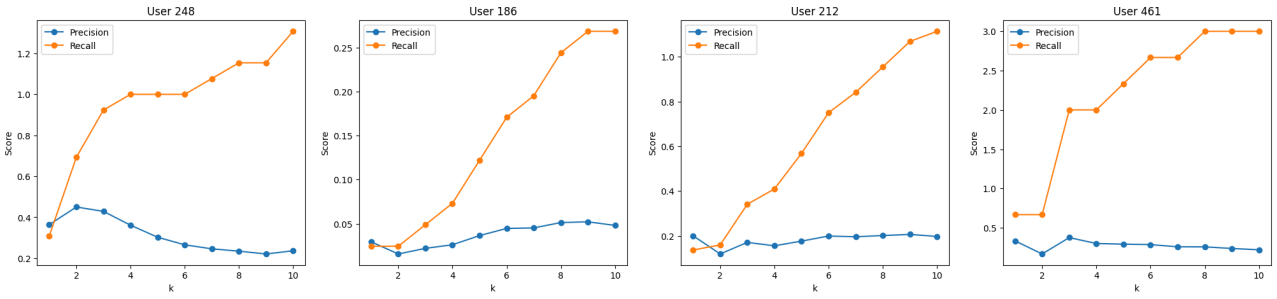
13) After processing all users, the average recall and precision values are calculated by dividing the sums by the number of users in the train_set.

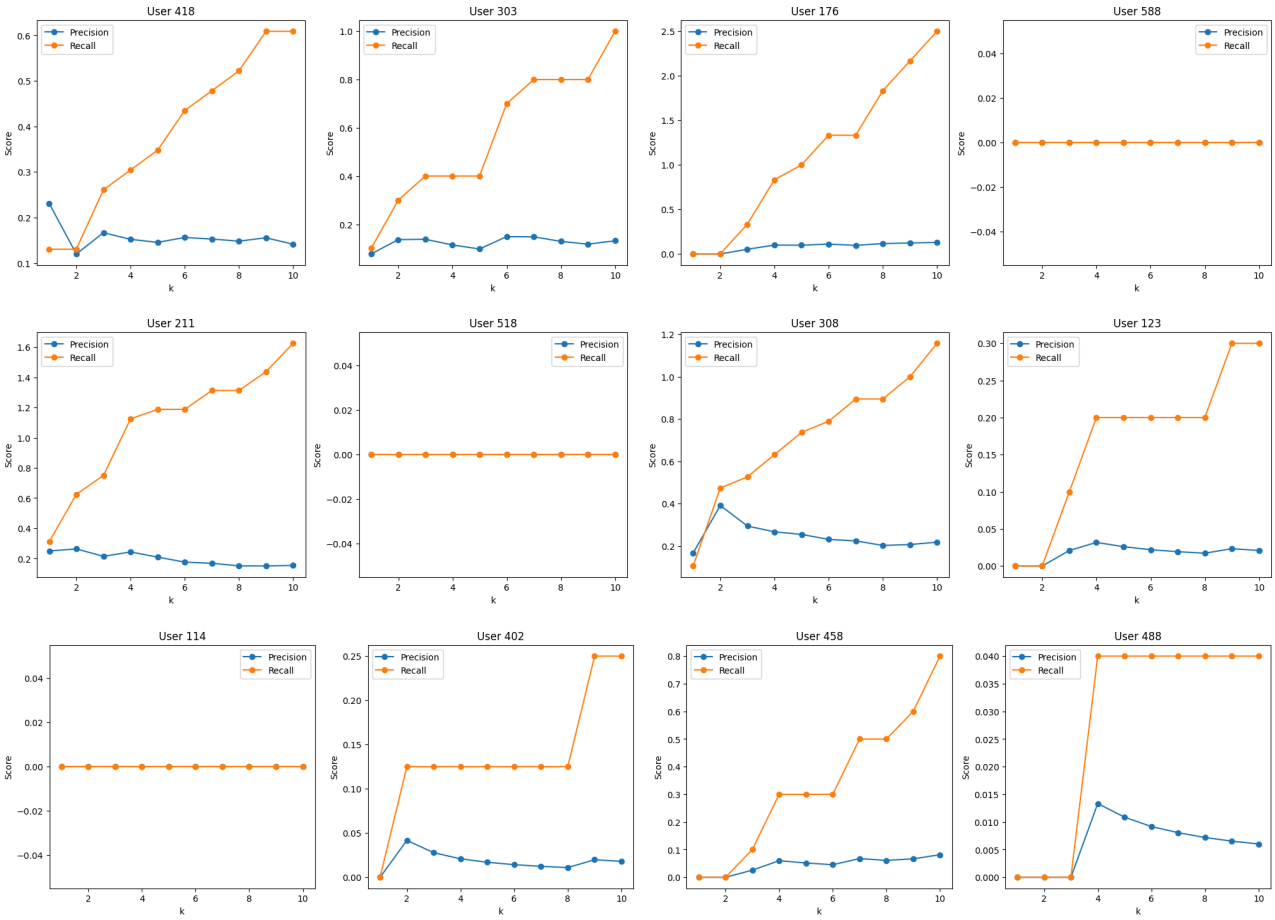
14) The average precision and recall values for the current k are appended to the precision_avgs and recall_avgs lists, respectively.

15) Finally, we are plotting the precision and recall values for different values of k using a line graph. The x-axis represents k, and the y-axis represents the average precision and recall values. A legend is added to distinguish between the two lines on the graph.

Part 4:

The evaluation process is extended to focus on individual users. A random sample of 16 users is selected from the test set. For each of these users, precision and recall are calculated for 'k' ranging from 1 to 10. This user-centric evaluation allows us to identify personalized performance characteristics.





Visualization

The markdown in the code file includes visualizations of precision and recall scores for each user in the sample, providing insights into individual user experiences and highlighting variations in recommendation quality.