ASSIGNMENT 3 REPORT

—

Team Mates:

B Krupa Kiranmai, *2021201022*
N C S Jagannath, *2021201024*

Associative Mining

1. First we analyzed the data that was given and preprocessed the data as given
    a. constructed a transactional database which was in the format

       {user_id: [movie_id_list]}

    b. We filtered out the movies that the user gave the rating greater than 2 to form the database.
    c. We removed the users who gave less than 10 movie ratings. The final database has 609 users (transactions).
    d. we split the data into 80-20% ,80% for the training and remaining 20 for the testing
2. We choose apriori algorithm to construct the associative rules of the movies.

Algorithm: Apriori. Find frequent itemsets using an iterative level-wise approach based on candidate generation.

Input:

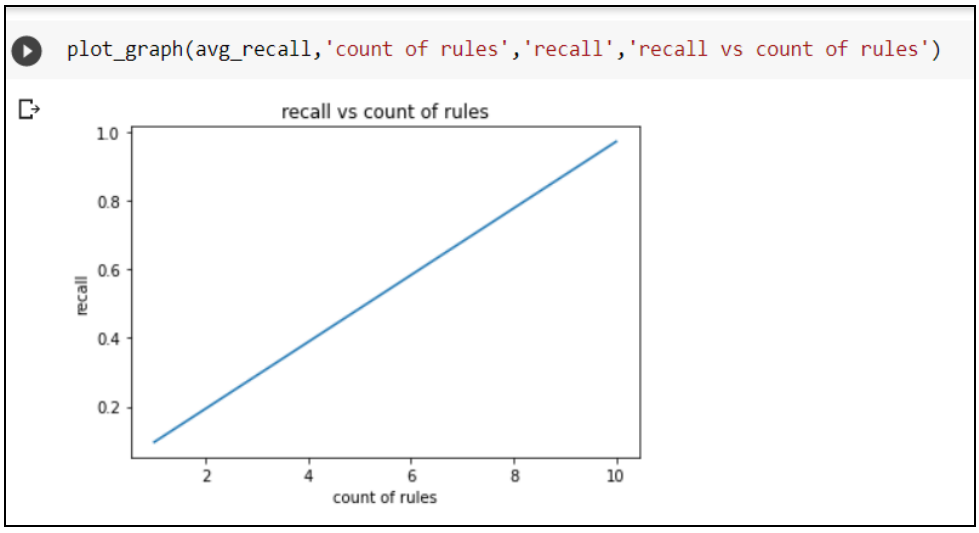
- D, a database of transactions;
- min_sup, the minimum support count threshold.

Output: L, frequent itemsets in D.

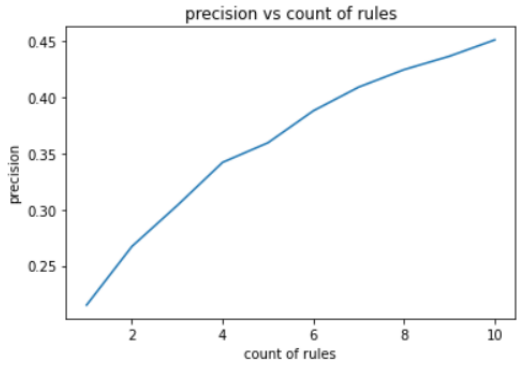
Method:

```
(1)   L1 = find_frequent_1-itemsets(D);
(2)   for (k = 2; Lk−1 ≠ φ; k++) {
(3)     Ck = apriori_gen(Lk−1);
(4)     for each transaction t ∈ D { // scan D for counts
(5)       Ct = subset(Ck, t); // get the subsets of t that are candidates
(6)       for each candidate c ∈ Ct
(7)         c.count++;
(8)     }
(9)     Lk = {c ∈ Ck|c.count ≥ min_sup}
(10)  }
(11)  return L = ∪k Lk;

procedure apriori_gen(Lk−1:frequent (k − 1)-itemsets)
(1)   for each itemset l1 ∈ Lk−1
(2)     for each itemset l2 ∈ Lk−1
(3)       if (l1[1] = l2[1]) ∧ (l1[2] = l2[2])
            ∧... ∧ (l1[k − 2] = l2[k − 2]) ∧ (l1[k − 1] < l2[k − 1]) then {
(4)         c = l1 ⋈ l2; // join step: generate candidates
(5)         if has_infrequent_subset(c, Lk−1) then
(6)           delete c; // prune step: remove unfruitful candidate
(7)         else add c to Ck;
(8)       }
(9)   return Ck;

procedure has_infrequent_subset(c: candidate k-itemset;
        Lk−1: frequent (k − 1)-itemsets); // use prior knowledge
(1)   for each (k − 1)-subset s of c
(2)     if s ∉ Lk−1 then
(3)       return TRUE;
(4)   return FALSE;
```

3. The reasons behind choosing of the apriori over FP growth were
    a. The dataset just consists of 609 transactions which was comparatively small, so multiple scans can be done easily.
    b. Construction of the candidate set would be easy than the cumbersome process of fp tree construction.
    c. The total number of movies are around 8000. We felt maybe the initial tree would not fit in the main memory.
4. We followed the textbook algorithm as it is and implemented it.
5. we stored the data in the set datatype of python because the checking and generating of subsets was fast and easy .This was one of the optimization we did.
6. After generating all the frequent itemsets from the apriori algorithm.For each frequent item set we generated subset of K-1 which serves as Y in the rule X->Y and the remaining element as X and generated all the rules possible.
7. For the second part we created three text files one was top 100 rules sorted with support,other was top 100 rules sorted with confidence,and another file with the common rules from both.
8. For the third part as the number of rules to be taken into consideration are to be varied, for each user from the train set each movie is taken and based on the number of rules to be considered created the recommendation set.
9. From the recommendation set we took the hit set and calculated the precision and recall.The graphs were as below:



```
[25] plot_graph(avg_precision,'count of rules','precision','precision vs count of rules')
```



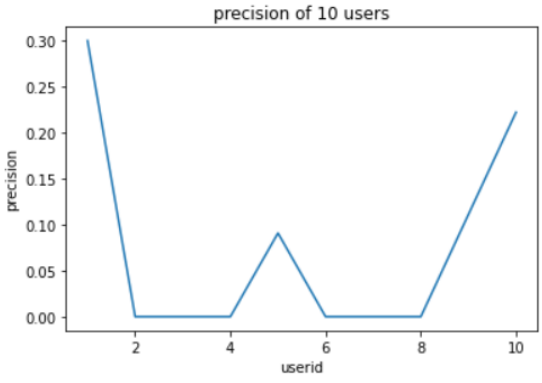
10. For the fourth question we took the first 10 users and calculated top 30 recommendations as per the given journal and calculated the precision and recall of the users. The table format :

```
print("{:<25} {:<25} {:<25}".format('User Id', 'Precision', 'Recall'))
for key, value in precision_users.items():
    print("{:<25} {:<25} {:<25}".format(key, value, recall_users[key]))
```

User Id	Precision	Recall
1.0	0.3	0.06521739130434782
2.0	0.0	0.0
3.0	0	0.0
4.0	0.0	0.0
5.0	0.09090909090909091	0.125
6.0	0.0	0.0
7.0	0.0	0.0
8.0	0.0	0.0
9.0	0.1111111111111111	0.125
10.0	0.2222222222222222	0.08333333333333333

Graphical representation:

```
[72] plot_graph(precision_users,'userid','precision','precision of 10 users')
```



```
plot_graph(recall_users,'userid','recall','recall of 10 users')
```