EXPERIMENT NO 3

NAME: Vishal Shashikant Salvi **UID: 2019230069**

DATE:04/08/2020 **BATCH:C**

Aim: To implement apriori algorithm.

Theory:

Apriori Algorithm:
Apriori algorithm was the first algorithm that was proposed for frequent itemset mining. It was later improved by R Agarwal and R Srikant and came to be known as Apriori. This algorithm uses two steps "join" and "prune" to reduce the search space. It is an iterative approach to discover the most frequent itemsets.

Apriori says:
The probability that item I is not frequent is if:

- P(I) < minimum support threshold, then I is not frequent.
- P (I+A) < minimum support threshold, then I+A is not frequent, where A also belongs to itemset.
- If an itemset set has value less than minimum support then all of its supersets will also fall below min support, and thus can be ignored. This property is called the Antimonotone property.

The steps followed in the Apriori Algorithm of data mining are:

1. **Join Step**: This step generates (K+1) itemset from K-itemsets by joining each item with itself.
2. **Prune Step**: This step scans the count of each item in the database. If the candidate item does not meet minimum support, then it is regarded as infrequent and thus it is removed. This step is performed to reduce the size of the candidate itemsets.

Steps In Apriori

Apriori algorithm is a sequence of steps to be followed to find the most frequent itemset in the given database. This data mining technique follows the join and the prune steps iteratively until the most frequent itemset is achieved. A minimum support threshold is given in the problem or it is assumed by the user.

1) In the first iteration of the algorithm, each item is taken as a 1-itemsets candidate. The algorithm will count the occurrences of each item.

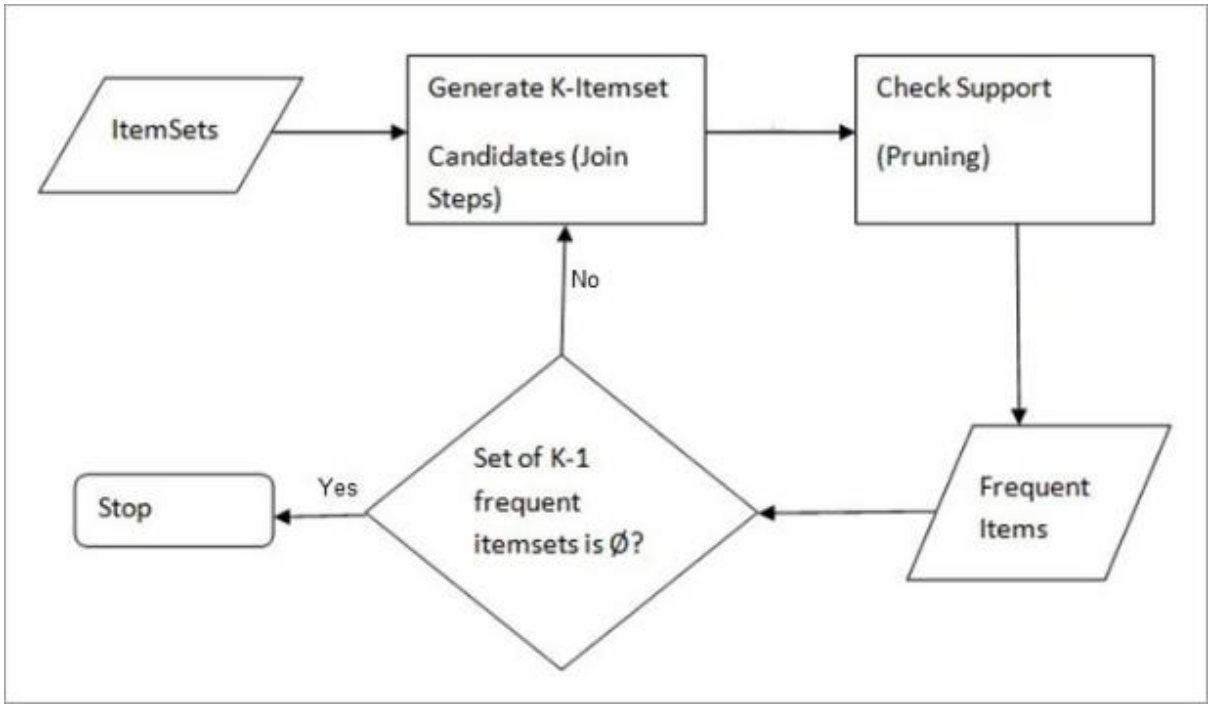
2) Let there be some minimum support, min_sup (eg 2). The set of 1 – itemsets whose occurrence is satisfying the min sup are determined. Only those candidates which count more than or equal to min_sup, are taken ahead for the next iteration and the others are pruned.

3) Next, 2-itemset frequent items with min_sup are discovered. For this in the join step, the 2-itemset is generated by forming a group of 2 by combining items with itself.

4) The 2-itemset candidates are pruned using min-sup threshold value. Now the table will have 2 –itemsets with min-sup only.

5) The next iteration will form 3 –itemsets using join and prune step. This iteration will follow antimonotone property where the subsets of 3-itemsets, that is the 2 –itemset subsets of each group fall in min_sup. If all 2-itemset subsets are frequent then the superset will be frequent otherwise it is pruned.

6) Next step will follow making 4-itemset by joining 3-itemset with itself and pruning if its subset does not meet the min_sup criteria. The algorithm is stopped when the most frequent itemset is achieved.



<u>Example of Apriori:</u> Support threshold=50%, Confidence= 60%

TABLE-1

Transaction	List of items
T1	I1,I2,I3
T2	I2,I3,I4
T3	I4,I5
T4	I1,I2,I4
T5	I1,I2,I3,I5
T6	I1,I2,I3,I4

Solution:

Support threshold=50% => 0.5*6= 3 => min_sup=3

1. Count Of Each Item

TABLE-2

Item	Count
I1	4
I2	5
I3	4
I4	4
I5	2

2. Prune Step: TABLE -2 shows that I5 item does not meet min_sup=3, thus it is deleted, only I1, I2, I3, I4 meet min_sup count.

TABLE-3

Item	Count
I1	4
I2	5
I3	4
I4	4

3. Join Step: Form 2-itemset. From **TABLE-1** find out the occurrences of 2-itemset.

TABLE-4

Item	Count
I1,I2	4
I1,I3	3
I1,I4	2
I2,I3	4
I2,I4	3
I3,I4	2

4. Prune Step: TABLE -4 shows that item set {I1, I4} and {I3, I4} does not meet min_sup, thus it is deleted.

TABLE-5

Item	Count
I1,I2	4
I1,I3	3
I2,I3	4
I2,I4	3

5. Join and Prune Step:

Form 3-itemset. From the **TABLE- 1** find out occurrences of 3-itemset. From **TABLE-5**, find out the 2-itemset subsets which support min_sup.

We can see for itemset {I1, I2, I3} subsets, {I1, I2}, {I1, I3}, {I2, I3} are occurring in **TABLE-5** thus {I1, I2, I3} is frequent.

We can see for itemset {I1, I2, I4} subsets, {I1, I2}, {I1, I4}, {I2, I4}, {I1, I4} is not frequent, as it is not occurring in **TABLE-5** thus {I1, I2, I4} is not frequent, hence it is deleted.

TABLE-6

Item
I1,I2,I3
I1,I2,I4
I1,I3,I4
I2,I3,I4

Only {I1, I2, I3} is frequent.

6. Generate Association Rules: From the frequent itemset discovered above the association could be:
{I1, I2} => {I3}

Confidence = support {I1, I2, I3} / support {I1, I2} = (3/ 4) * 100 = 75%

{I1, I3} => {I2}

Confidence = support {I1, I2, I3} / support {I1, I3} = (3/ 3) * 100 = 100%

{I2, I3} => {I1}

Confidence = support {I1, I2, I3} / support {I2, I3} = (3/ 4) * 100 = 75%

{I1} => {I2, I3}

Confidence = support {I1, I2, I3} / support {I1} = (3/ 4) * 100 = 75%

{I2} => {I1, I3}

Confidence = support {I1, I2, I3} / support {I2 = (3/ 5) * 100 = 60%

{I3} => {I1, I2}

Confidence = support {I1, I2, I3} / support {I3} = (3/ 4) * 100 = 75%

This shows that all the above association rules are strong if minimum confidence threshold is 60%.

Code:

```
import itertools

minSupport = int(input("Enter Min. Support: "))

n = int(input("Enter the no. of transactions: "))
print("------------------------------")

transactions = {}

for i in range(n):
    transactions[i] = []

for i in range(n):
    itemno = int(input("Enter no. of items for transaction " + str(i + 1) + "\n"))
    for no in range(itemno):
        transactions[i].append(str(input("Enter item " + str(no + 1) + " : ")))

support = {}

for i in range(n):
    for j in range(len(transactions[i])):
        if transactions[i][j] not in support:
            support[transactions[i][j]] = 1
        else:
            support[transactions[i][j]] += 1

print("------------------------------")
print("Support:", support)

todelete = []
for k, v in support.items():
```

```
    if v < minSupport:
        todelete.append(k)

for d in todelete:
    del support[d]

print("Support 1 item:", support)

archive = support.copy()

temp = list(support.keys())

itemListCount = 2

supportTemp = {}

while itemListCount < 5:
    list1 = list(itertools.combinations(temp, itemListCount))
    list2 = []

    for i in range(len(list1)):
        list2.append(0)

    count = 0
    for i in list1:
        for j in range(len(transactions.keys())):
            if all(elem in transactions[j] for elem in i):
                list2[count] += 1
        count += 1

    support2 = {}
    for i in range(len(list2)):
```

```
        support2[list1[i]] = list2[i]

    todelete = []
    for k, v in support2.items():
        if v < minSupport:
            todelete.append(k)

    for d in todelete:
        del support2[d]

    print("Support", itemListCount, "items:", support2)

    itemListCount += 1

    if not support2:
        break

    supportTemp = support2.copy()

    if not archive:
        archive = support2.copy()
    else:
        archive.update(support2)

assRule = {}

for key, value in supportTemp.items():
    for i in key:
        newList1 = []
        newList2 = []
        tup = []
        newList1.append(i)
```

```
        for j in key:
            if i == j:
                continue
            else:
                tup.append(j)
        newList1.append(tuple(tup))
        newList2.append(tuple(tup))
        newList2.append(i)

        assRule[tuple(newList1)] = round((archive[key] / archive[i]) * 100, 2)

        if len(tup) == 1:
            assRule[tuple(newList2)] = round((archive[key] / archive[tup[0]]) * 100, 2)
        else:
            assRule[tuple(newList2)] = round((archive[key] / archive[tuple(tup)]) * 100, 2)

print("------------------------------")
print("Association Rules :\n",assRule)
```

Output:

```
Enter Support: 2
Enter the number of transactions: 4
------------------------------
Enter number of items in transaction: 1
3
Enter item 1 : A
Enter item 2 : C
Enter item 3 : D
Enter number of items in transaction: 2
3
Enter item 1 : B
Enter item 2 : C
Enter item 3 : E
Enter number of items in transaction: 3
4
Enter item 1 : A
Enter item 2 : B
Enter item 3 : C
Enter item 4 : E
Enter number of items in transaction: 4
2
Enter item 1 : B
Enter item 2 : E
------------------------------
Support: {'A': 2, 'C': 3, 'D': 1, 'B': 3, 'E': 3}
Support 1 item: {'A': 2, 'C': 3, 'B': 3, 'E': 3}
Support 2 items: {('A', 'C'): 2, ('C', 'B'): 2, ('C', 'E'): 2, ('B', 'E'): 3}
Support 3 items: {('C', 'B', 'E'): 2}
Support 4 items: {}
------------------------------
```

```
Enter Support: 2
Enter the number of transactions: 4
------------------------------
Enter number of items in transaction: 1
3
Enter item 1 : A
Enter item 2 : C
Enter item 3 : D
Enter number of items in transaction: 2
3
Enter item 1 : B
Enter item 2 : C
Enter item 3 : E
Enter number of items in transaction: 3
4
Enter item 1 : A
Enter item 2 : B
Enter item 3 : C
Enter item 4 : E
Enter number of items in transaction: 4
2
Enter item 1 : B
Enter item 2 : E
------------------------------
Support: {'A': 2, 'C': 3, 'D': 1, 'B': 3, 'E': 3}
Support 1 item: {'A': 2, 'C': 3, 'B': 3, 'E': 3}
Support 2 items: {('A', 'C'): 2, ('C', 'B'): 2, ('C', 'E'): 2, ('B', 'E'): 3}
Support 3 items: {('C', 'B', 'E'): 2}
Support 4 items: {}
------------------------------
Association Rules :
 {('C', ('B', 'E')): 66.67, (('B', 'E'), 'C'): 66.67, ('B', ('C', 'E')): 66.67, (('C', 'E'), 'B'): 100.0, ('E', ('C', 'B')): 66.67, (('C', 'B'), 'E'): 100.0}
PS C:\Users\Vishal\projects\vishu>
```

Conclusion: Thus, I studied and implemented apriori algorithm for finding frequent itemset in a dataset. Basically, Apriori algorithm is used to find the frequent patterns available in the database. I also came to know that if an itemset is infrequent, all its supersets will be infrequent. The major advantage of apriori algorithm is that it is easy to implement. The disadvantage is that it requires many database scans as well as very slow.