Name:	
Roll No:	
Class/Sem:	TE/V
Experiment No.:	9
Title:	Implementation of association mining algorithm like Apriori using languages like JAVA/ python.
Date of Performance:	
Date of Submission:	
Marks:	
Sign of Faculty:	

Aim: To Implement Apriori algorithm using languages like JAVA/ python.

Objective:- Understand the working of Apriori algorithm and it's implemention using python/Java.

Theory:

Apriori is an algorithm for frequent item set mining and association rule learning over transactional databases. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in the database: this has applications in domains such as market basket analysis.

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:

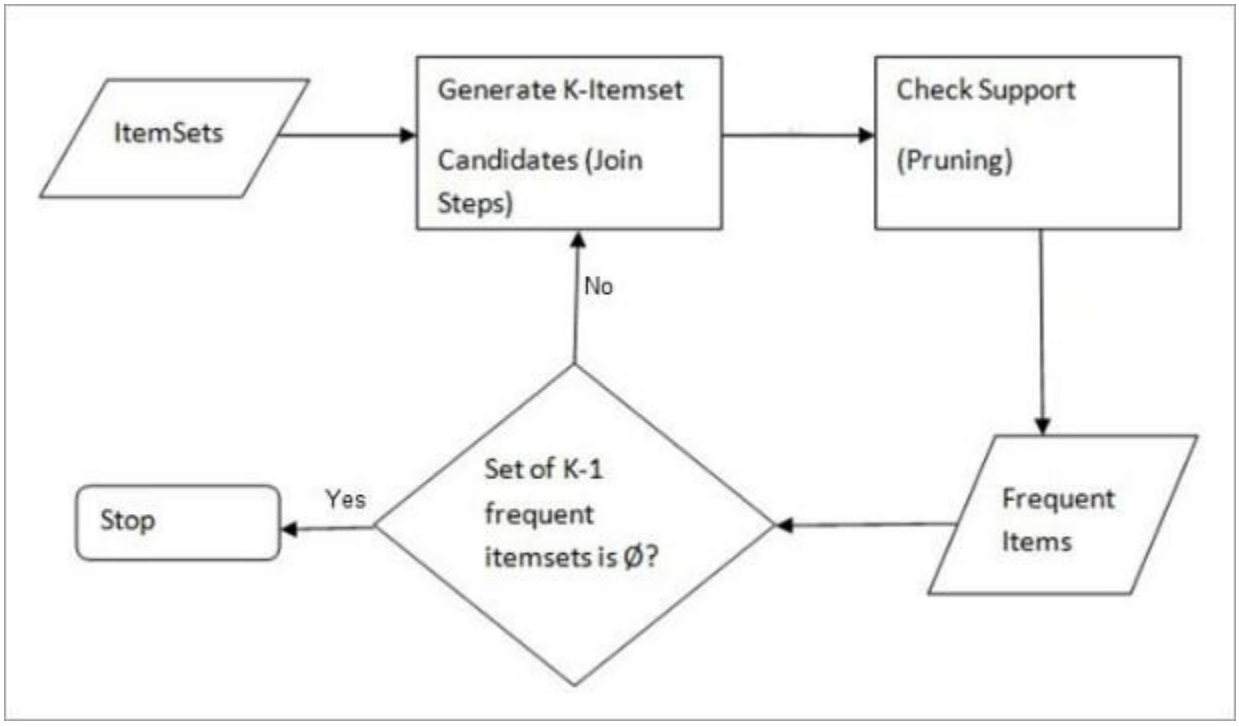
1. Join Step: This step generates (K+1) itemset from K-itemsets by joining each item with itself.
2. Prune Step: This step scans the count of each item in the database. If the candidate item does not meet minimum support, then it is regarded as infrequent and thus it is removed. This step is performed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



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;
```

Program:

data = [

['T100', ['11', '12', '15']],

['T200', ['12', '14']],

['T300', ['12', '13']], ['T400', ['11', '12', '14']],

```
['T500', ['11', '13']],
['T600', ['12', '13']],
['T700', ['11', '13']],
['T800', ['11', '12', '13', '15']],
['T900', ['11', '12', '13']],
]
init = [] for i in
data: for q in i[1]:
if q not in init:
        init.append(q)
init = sorted(init)
print(init)
sp = 0.4
s = int(sp * len(init))
s


from collections import Counter
c = Counter() for i in init: for d
in data: if i in d[1]:

        c[i] += 1
```

```
print("C1:")
for i in c:
    print(str([i]) + ": " + str(c[i]))
print()

l = Counter()
for i in c:
  if c[i] >= s:
    l[frozenset([i])] += c[i]
print("L1:") for i in l:
  print(str(list(i)) + ": " + str(l[i]))
print()
pl = 1
pos = 1

for count in range(2, 1000):
  nc = set() temp = list(l) for
  i in range(0, len(temp)):
    for j in range(i + 1, len(temp)):
      t = temp[i].union(temp[j])

    if len(t) == count:
```

```
        nc.add(temp[i].union(temp[j])
) nc = list(nc) c = Counter() for i in
nc:
    c[i] = 0 for q in
data: temp =
set(q[1]) if
i.issubset(temp):
        c[i] += 1 print("C" +
str(count) + ":") for i in c:
    print(str(list(i)) + ": " + str(c[i]))
print()
l = Counter()
for i in c:
    if c[i] >= s:
        l[i] += c[i] print("L" +
str(count) + ":") for i in l:
    print(str(list(i)) + ": " +
str(l[i])) print() if len(l) == 0:

    break pl
= l pos =
count
```

```
print("Result:") print("L"
+ str(pos) + ":") for i in
pl:
    print(str(list(i)) + ": " + str(pl[i]))
print()

from itertools import combinations for l1 in pl: c =
[frozenset(q) for q in combinations(l1, len(l1) - 1)]
    mmax = 0
for a in c: b
    = l1 - a
    ab = l1
    sab = 0
    sa = 0 sb
    = 0
for q in data: temp =
    set(q[1]) if
    a.issubset(temp):
        sa += 1
    if b.issubset(temp):
        sb += 1
    if ab.issubset(temp):
```

```
        sab += 1 temp = sab
    / sa * 100 if temp >
    mmax: mmax = temp
    temp = sab / sb * 100 if
    temp > mmax:
        mmax = temp
    print(str(list(a)) + "->" + str(list(b)) + " = " + str(sab / sa * 100) + "%")
print(str(list(b)) + "->" + str(list(a)) + " = " + str(sab / sb * 100) + "%")
curr = 1
print("Choosing:", end='
') for l1 in pl: c = pl - l1
ab = pl sab = 0 sa = 0 sb =
0
for q in data:
    temp = set(q[l1]) if
    l1.issubset(temp):
    sa += 1

    if c.issubset(temp):
        sb += 1
    if ab.issubset(temp):
```

```
            sab += 1 temp = sab
        / sa * 100 if temp ==
        mmax:
            print(curr, end=' ')
            curr += 1 temp = sab
        / sb * 100 if temp ==
        mmax:
            print(curr, end=' ')
            curr += 1
    print()
    print()
```

Output:

['11', '12', '13', '14', '15']

C1:

['11']: 6

['12']: 7

['13']: 6

['14']: 2

['15']: 2

L1:

['11']: 6

['12']: 7

['13']: 6

['14']: 2

['15']: 2

C2:

['15']: 0

['15', '14']: 0

['13', '15']: 0

['11', '15']: 0

['15', '12']: 2

L2: ['15',

'12']: 2

Result:

L2:

['15', '12']: 2

['12']->['15'] = 100.0%

['15']->['12'] = 100.0%

Conclusion: Implementing the Apriori algorithm in Java or Python is a practical approach for discovering associations and patterns in transactional data. It empowers data analysts and data scientists to extract valuable insights, make data-driven decisions, and uncover meaningful relationships within large datasets.