Practical:-7

Aim: Implement Apriori Algorithm in Java/.net/Python.

```
import csv
from itertools import combinations
def apriori_sm(min_support, min_confidence, file_loc)
    file1 = open("Apriori Output.txt","w")
    def read_data(file_loc='GroceryStoreDataSet.csv'):
        trans = dict()
        with open(file_loc) as f:
            filedata = csv.reader(f, delimiter=',')
            count = 0
            for line in filedata:
                count += 1
                trans[count] = list(set(line))
        return trans
    def frequence(items_lst, trans, check=False):
        items_counts = dict()
        for i in items_lst:
            temp_i = {i}
            if check:
                temp_i = set(i)
            for j in trans.items():
                if temp_i.issubset(set(j[1])):
                    if i in items_counts:
```

```
                items_counts[i] += 1
            else:
                items_counts[i] = 1
    return items_counts
def support(items_counts, trans):
    support = dict()
    total_trans = len(trans)
    for i in items_counts:
        support[i] = items_counts[i]/total_trans
    return support
def association_rules(items_greater_then_min_support):
    rules = []
    dict_rules = {}
    for i in items_greater_then_min_support:
        dict_rules = {}
        if type(i) != type(str()):
            i = list(i)
            temp_i = i[:]
            for j in range(len(i)):
                k = temp_i[j]
                del temp_i[j]
                dict_rules[k] = temp_i
                temp_i = i[:]
        rules.append(dict_rules)
    temp = []
```

```
    for i in rules:
        for j in i.items():
            if type(j[1]) != type(str()):
                temp.append({tuple(j[1])[0]: j[0]})
            else:
                temp.append({j[1]: j[0]})
    rules.extend(temp)
    return rules
def confidence(associations, d, min_confidence):
    ans = {}
    for i in associations:
        for j in i.items():
            if type(j[0]) == type(str()):
                left = {j[0]}
            else:
                left = set(j[0])
            if type(j[1]) == type(str()):
                right = {j[1]}
            else:
                right = set(j[1])
            for k in d:
                if type(k) != type(str()):
                    if left.union(right) - set(k) == set():
                        up = d[k]
                    if len(right) == len(set(k)) and right - set(k) == set():
```

```
                    down = d[k]
                else:
                    if len(right) >= len({k}):
                        if right - {k} == set():
                            down = d[k]
                    elif len(right) <= len({k}):
                        if {k} - right == set():
                            down = d[k]
            if up/down >= min_confidence:
                ans[tuple(left)[0]] = right, up/down, up, down
    file1.write(str(ans))
trans = read_data()
    number_of_trans = [len(i) for i in trans.values()]
    items_lst = set()
    itemcount_track = list()
    for i in trans.values():
        for j in i:
            items_lst.add(j)
     store_item_lst = list(items_lst)[:]
    items_greater_then_min_support = list()
    items_counts = frequence(items_lst, trans)
    itemcount_track.append(items_counts)
    items_greater_then_min_support.append({j[0]:j[1]
 for j in support(items_counts, trans).items()
if j[1]>min_support})
```

```
    for i in range(2, max(number_of_trans)+1):
        item_list = combinations(items_lst, i)
        items_counts = frequence(item_list, trans, check=True)
        itemcount_track.append(items_counts)
        if list({j[0]:j[1] for j in support(items_counts, trans).items()
            if j[1]>min_support}.keys()) != []:
            items_greater_then_min_support.append({j[0]:j[1]
                    for j in support(items_counts, trans).items()
                            if j[1]>min_support})
  d = {}
    {d.update(i) for i in itemcount_track}
    associations =
association_rules(items_greater_then_min_support[len(items_greater_then_min_support)-1])
    associations_greater_then_confidene = confidence(associations, d, min_confidence)
apriori_sm(0.03, 0.6, 'GroceryStoreDataSet.csv')
```

Output:-

Apriori Output - Notepad

File Edit Format View Help

```
{'TEA': ({'COFFEE', 'CORNFLAKES', 'MILK'}, 1.0, 1, 1), 'BISCUIT': ({'COCK'}, 0.6666666666666666, 2, 3),
'COFFEE': ({'TEA', 'CORNFLAKES', 'MILK'}, 1.0, 1, 1), 'MILK': ({'BISCUIT', 'CORNFLAKES', 'BREAD'}, 1.0, 1, 1),
'BREAD': ({'BISCUIT', 'CORNFLAKES', 'MILK'}, 1.0, 1, 1), 'COCK': ({'BISCUIT', 'COFFEE', 'CORNFLAKES'}, 1.0, 2,
2), 'CORNFLAKES': ({'TEA', 'COFFEE', 'MILK'}, 1.0, 1, 1), 'MAGGI': ({'TEA', 'BREAD', 'JAM'}, 1.0, 1, 1),
'JAM': ({'BREAD', 'MAGGI', 'MILK'}, 1.0, 1, 1)}
```

Ln 1, Col 479 | 100% | Windows (CRLF) | UTF-8