**Experiment 9**

**Aim: Implementation of Association Rule Miningalgorithm (Apriori)**

**Theory:**

Apriori is a widely used algorithm in data mining and machine learning for finding frequent itemsets in transactional databases. It is a fundamental algorithm for association rule mining, which aims to discover relationships or patterns in data. Apriori is specifically used to identify frequent itemsets and generate association rules based on these itemsets.

Components of Apriori algorithm

The given three components comprise the apriori algorithm.

* Support: Support measures how frequently an itemset appears in the dataset. It is defined as the ratio of the number of transactions containing the itemset to the total number of transactions. High support indicates that an itemset is common in the dataset.
* Confidence: Confidence measures the strength of an association rule in terms of the likelihood that the consequent (the item on the right side of the rule) will be purchased when the antecedent (the item(s) on the left side of the rule) is/are purchased.
* Lift: Lift measures how much more likely the consequent is to be purchased when the antecedent is purchased compared to when the consequent is purchased without the antecedent.

**Program:**

import csv

from itertools import combinations

def find\_frequency(string, freqs):

for dictionary in freqs:

if (string in list(dictionary.keys())):

return dictionary[string]

def power\_set(string):

power\_set = set({})

for i in range(0, len(string) + 1):

for element in combinations(string, i):

power\_set.add(' '.join(element))

power\_set -= set({''})

power\_set.remove(' '.join(string))

power\_set = list(power\_set)

power\_set.sort()

return power\_set

def association(keys, confidence, all\_freqs):

final\_rules = []

for key in keys:

lhs = power\_set(key.split(' '))

rhs = []

elements = set(key) - {' '}

for elem in lhs:

to\_join = list(elements - set(elem) - {' '})

to\_join.sort()

rhs.append(' '.join(to\_join))

for l, r in zip(lhs, rhs):

rule = l + ' -> ' + r

string = l + " " + r

temp\_lst = string.split(' ')

temp\_lst.sort()

string = ' '.join(temp\_lst)

conf = (find\_frequency(string, all\_freqs) / find\_frequency(l, all\_freqs)) \* 100

if (conf >= confidence):

print(rule + " with a confidence of " + str(conf) + "%")

def combine(keys, key\_len):

final\_keys = []

for lst1 in keys:

for lst2 in keys:

temp\_lst = lst1[:]

temp\_lst.extend(lst2)

temp\_lst = list(set(temp\_lst))

temp\_lst.sort()

if (len(temp\_lst) == key\_len + 1):

if (temp\_lst not in final\_keys):

final\_keys.append(temp\_lst)

return final\_keys

def filter\_comb(keys, data, support):

final\_keys = []

frequency = {}

for key in keys:

present = True

freq = 0

for value in list(data.values()):

for item in key:

if (item not in value):

present = present and False

if present == True:

freq = freq + 1

present = True

frequency[' '.join(key)] = freq

frequency = dict(filter(lambda elem: elem[1] >= support, frequency.items()))

return frequency

items = []

item\_set = []

data = {}

previous = {}

support = int(input("Enter the minimum support (in percent): "))

confidence = int(input("Enter the minimum confidence (in percent): "))

frequency = {}

key\_len = 1

all\_freqs = []

# Provided dataset

transactions = [

[1, 2, 3, 4],

[1, 4, 2, 5, 4],

[3, 4, 5],

[1, 3, 4],

[3, 4, 6],

[1, 3, 4, 6],

[1, 6],

[1, 3, 4],

[3, 4, 2, 6],

[1, 4, 3],

[1, 3],

[1, 3, 6, 5]

]

# Convert transaction data to a dictionary

for i, transaction in enumerate(transactions):

data[i + 1] = list(map(str, transaction))

# Combine items into a single list

items = [item for sublist in data.values() for item in sublist]

# Get unique items

item\_set = list(set(items))

item\_set.sort()

# Calculate support threshold

support = (support / 100) \* len(data)

# Calculate item frequency

for item in item\_set:

frequency[item] = items.count(item)

# Filter items based on support

frequency = dict(filter(lambda elem: elem[1] >= support, frequency.items()))

key\_list = []

# Convert frequent items to a list of lists

for key in list(frequency.keys()):

key\_list.append([key])

while (True):

key\_list = combine(key\_list, key\_len)

all\_freqs.append(frequency)

previous = frequency.copy()

frequency = filter\_comb(key\_list, data, support)

key\_list = []

for key in list(frequency.keys()):

key\_l = key.split(' ')

key\_list.append(key\_l)

key\_len += 1

if (len(key\_list) <= 1):

break

print("The strong association rules are as follows:")

if (not frequency):

association(previous, confidence, all\_freqs)

else:

all\_freqs.append(frequency)

association(frequency, confidence, all\_freqs)

**Output:**





