|  |
| --- |
| Experiment No. 7 |
| Implement frequent pattern mining algorithm(Apriori) |
| Date of Performance: 02/09/24 |
| Date of Submission: 23/09/24 |

**AIM:** To implement Apriori algorithm

**Objective:** Develop a program to implement Apriori Algorithm on the given dataset

**THEORY:**

Apriori is an algorithm for frequent item set mining and [association rule learning](https://en.wikipedia.org/wiki/Association_rule_learning) over transactional [databases](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Association_rules) which highlight general trends in the [database](https://en.wikipedia.org/wiki/Database): this has applications in domains such as [market basket analysis](https://en.wikipedia.org/wiki/Market_basket_analysis).

**Procedure or algorithm description:**

Level-wise algorithm:

a. Let k = 1

b. Generate frequent itemsets of length 1

c. Repeat until no new frequent itemsets are identified

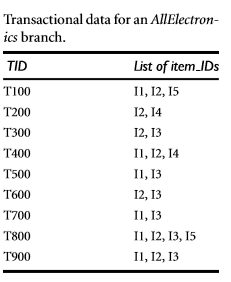
1. Generate length (k+1) candidate itemsets from length k frequent itemsets

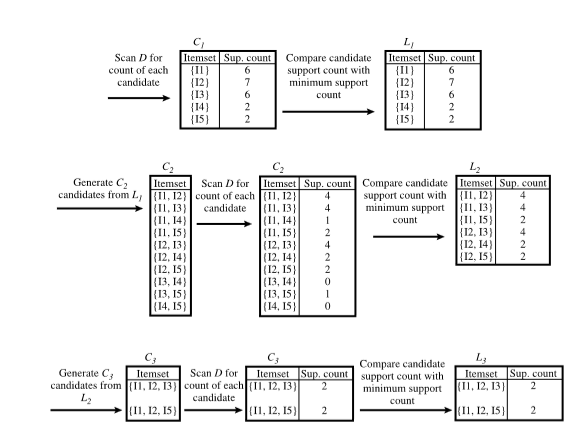
2. Prune candidate itemsets containing subsets of length k that are infrequent

3. Count the support of each candidate by scanning the DB

4. Eliminate candidates that are infrequent, leaving only those that are frequent {\displaystyle {\begin{aligned}&\mathrm {Apriori} (T,\epsilon )\\&\qquad L\_{1}\gets \{\mathrm {large~1-itemsets} \}\\&\qquad k\gets 2\\&\qquad \mathrm {\textbf {while}} ~L\_{k-1}\neq \ \emptyset \\&\qquad \qquad C\_{k}\gets \{a\cup \{b\}\mid a\in L\_{k-1}\land b\not \in a\}-\{c\mid \{s\mid s\subseteq c\land |s|=k-1\}\nsubseteq L\_{k-1}\}\\&\qquad \qquad \mathrm {{\textbf {for}}~transactions} ~t\in T\\&\qquad \qquad \qquad C\_{t}\gets \{c\mid c\in C\_{k}\land c\subseteq t\}\\&\qquad \qquad \qquad \mathrm {{\textbf {for}}~candidates} ~c\in C\_{t}\\&\qquad \qquad \qquad \qquad {\mathit {count}}[c]\gets {\mathit {count}}[c]+1\\&\qquad \qquad L\_{k}\gets \{c\mid c\in C\_{k}\land ~{\mathit {count}}[c]\geq \epsilon \}\\&\qquad \qquad k\gets k+1\\&\qquad \mathrm {\textbf {return}} ~\bigcup \_{k}L\_{k}\end{aligned}}}

**Apriori Algorithm:**

****

****

**Code and output**:

import pandas as pd

from itertools import combinations

def create\_c1(transactions):

"""Create the first candidate itemset C1"""

C1 = []

for transaction in transactions:

for item in transaction:

if [item] not in C1:

C1.append([item])

return list(map(sorted, C1))

def scan\_d(transactions, candidates, min\_support):

"""Scan the database to find frequent itemsets"""

item\_count = {}

for transaction in transactions:

for candidate in candidates:

if set(candidate).issubset(set(transaction)):

if tuple(candidate) in item\_count:

item\_count[tuple(candidate)] += 1

else:

item\_count[tuple(candidate)] = 1

num\_items = len(transactions)

frequent\_itemsets = []

support\_data = {}

for key, count in item\_count.items():

support = count / num\_items

if support >= min\_support:

frequent\_itemsets.append(list(key))

support\_data[key] = support

return frequent\_itemsets, support\_data

def apriori(transactions, min\_support=0.5):

"""The main function to run the Apriori algorithm"""

C1 = create\_c1(transactions)

L1, support\_data = scan\_d(transactions, C1, min\_support)

frequent\_itemsets = [L1]

k = 2

while len(frequent\_itemsets[k - 2]) > 0:

Ck = list(map(sorted, [list(i) for i in combinations(set(item for sublist in frequent\_itemsets[k - 2] for item in sublist), k)]))

Lk, support\_k = scan\_d(transactions, Ck, min\_support)

support\_data.update(support\_k)

frequent\_itemsets.append(Lk)

k += 1

return frequent\_itemsets, support\_data

def generate\_rules(frequent\_itemsets, support\_data, min\_confidence=0.5):

"""Generate association rules from frequent itemsets"""

rules = []

for itemset in frequent\_itemsets:

if len(itemset) < 2:

continue # Only consider itemsets with more than one item

for i in range(1, len(itemset)):

for antecedent in combinations(itemset, i):

antecedent = list(antecedent)

consequent = list(set(itemset) - set(antecedent))

if not consequent:

continue

antecedent\_support = support\_data[tuple(antecedent)]

itemset\_support = support\_data[tuple(itemset)]

confidence = itemset\_support / antecedent\_support

if confidence >= min\_confidence:

rules.append((antecedent, consequent, confidence))

return rules

# Example usage

if \_name\_ == "\_main\_":

transactions = [

['bread', 'jelly', 'butter'],

['bread', 'butter'],

['bread', 'milk', 'butter'],

['coke', 'bread'],

['coke', 'milk'],

]

min\_support = 0.3

frequent\_itemsets, support\_data = apriori(transactions, min\_support)

print("Frequent Itemsets:")

for i, itemset in enumerate(frequent\_itemsets):

print(f"Level {i + 1}: {itemset}")

print("\nSupport Data:")

for item, support in support\_data.items():

print(f"{item}: {support:.2f}")

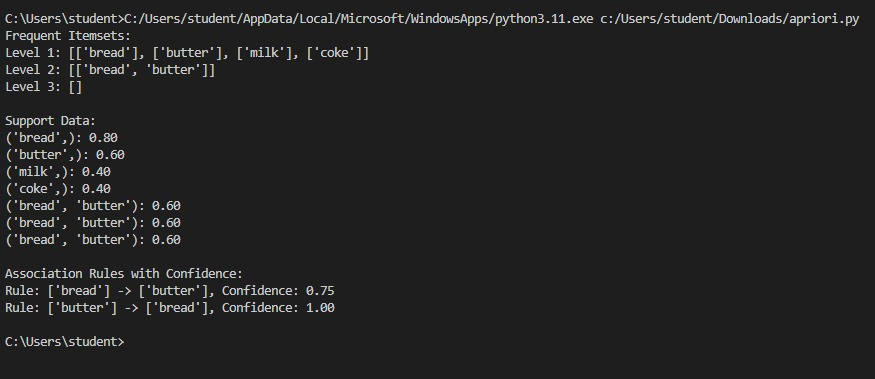
# Generate rules with confidence

rules = generate\_rules(frequent\_itemsets[1], support\_data, min\_confidence=0.5)

print("\nAssociation Rules with Confidence:")

for antecedent, consequent, confidence in rules:

print(f"Rule: {antecedent} -> {consequent}, Confidence: {confidence:.2f}")



**Conclusion**: Comment on the rules generated by the algorithm