St. Francis Institute of Technology, Mumbai-400 103

**Department Of Information Technology**

A.Y. 2024-2025

Class: TE-ITA/B, Semester: VI

Subject: **Business Intelligence Lab**

**Experiment – 9 To implement Apriori association mining using any one language (JAVA/Python)**

1. **Aim: :** Implementation of Apriori Association in Data Mining using any one Language
2. **Objectives:** After study of this experiment, the students will be able toimplement Apriori Algorithm in (JAVA/R/Python)
3. **Outcomes:** After study of this experiment, the students will be able to

**CO 5:** Design and Implement various frequent data mining techniques and formulate association rules   on large data sets

1. **Prerequisite:** Introduction to algorithms of Associativity
2. **Requirements:** Personal Computer, Windows XP operating system/Windows 7, Internet Connection, Microsoft Word, WEKA tool.
3. **Theory:**
4. What are Association Rules in Data Mining?

Association rules in data mining are a popular technique used to identify relationships or patterns among variables in large datasets. These rules are typically represented in the form of "if-then" statements, where the presence of certain items in a dataset implies the presence of other items. One of the most common applications of association rules is in market basket analysis, where businesses analyze the purchasing behavior of customers to find combinations of products that are frequently bought together. The strength of these rules is measured using key metrics such as support (how often items appear together), confidence (how often the rule holds true), and lift (the strength of the relationship compared to random chance).

1. How Association helps in Boosting the Business Profit?

Association rules significantly help in boosting business profits by enabling data-driven decision-making. By identifying which products are commonly purchased together, businesses can strategically place those items near each other in physical stores or recommend them online, thereby increasing the likelihood of multiple purchases. This technique is widely used in product bundling and cross-selling strategies. Furthermore, businesses can personalize marketing campaigns and promotions based on customer buying patterns, leading to improved customer engagement and higher conversion rates. Inventory management also benefits, as companies can ensure frequently associated items are well-stocked, reducing missed sales opportunities. Overall, the use of association rules empowers businesses to understand customer behavior better, optimize product offerings, and implement more effective marketing strategies, all of which contribute to increased profitability.

1. **Laboratory Exercise:** Implementation of Apriori Algorithm in Java/Python, printout of implementation along with coding and snapshot.
2. **Post-Experiments Exercise**
3. **Questions:**
   * MCQ type test
   * Compare Apriori and FP Tree
4. **Conclusion:**
   * Summary of Experiment
   * Importance of Experiment
   * Application of Experiment

1. **Reference:** Data Mining: Concept & Techniques, 3rd Edition, Jiawei Han, Micheline  Kamber, Jian Pei, Elsevier.

**Code:**

import csv

from itertools import combinations

from collections import defaultdict

def load\_transactions\_from\_csv(filename):

transactions = []

with open(filename, "r") as file:

reader = csv.reader(file)

for row in reader:

items = row[2:] # Skip Order ID and Price

cleaned\_items = [item.strip() for item in items if item.strip()]

if cleaned\_items:

transactions.append(cleaned\_items)

return transactions

def get\_item\_support(transactions, itemsets):

item\_support = defaultdict(int)

for transaction in transactions:

transaction\_set = set(transaction)

for itemset in itemsets:

if set(itemset).issubset(transaction\_set):

item\_support[itemset] += 1

return item\_support

def apriori(transactions, min\_support):

num\_transactions = len(transactions)

min\_count = min\_support \* num\_transactions

# Frequent 1-itemsets

item\_counts = defaultdict(int)

for transaction in transactions:

for item in transaction:

item\_counts[(item,)] += 1

current\_itemsets = {itemset for itemset, count in item\_counts.items() if count >= min\_count}

frequent\_itemsets = dict()

k = 1

while current\_itemsets:

item\_support = get\_item\_support(transactions, current\_itemsets)

frequent\_itemsets.update({itemset: count for itemset, count in item\_support.items() if count >= min\_count})

items = sorted(set(item for itemset in current\_itemsets for item in itemset))

next\_itemsets = set(combinations(items, k + 1))

current\_itemsets = {itemset for itemset in next\_itemsets

if get\_item\_support(transactions, [itemset])[itemset] >= min\_count}

k += 1

return frequent\_itemsets

# Main execution

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

filename = "C:\Users\Student\Downloads\resturant\_trans.csv"

min\_support = 0.2

transactions = load\_transactions\_from\_csv(filename)

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

print("Top 10 frequent itemsets:")

for itemset, count in sorted(frequent\_itemsets.items(), key=lambda x: -x[1])[:10]:

print(f"{itemset}: {count}")

**Output:**

