Name :- Prashant Suresh Shirgave

Roll No:-03 **Batch:**T3

Class: TY(CSE-AIML)

Experiment No. 11

Title :- Implement A-priori algorithm in data mining.

Aim:- Demonstrate A-priori algorithm in data mining.

Implementation:

```
from itertools import combinations

# Sample transaction dataset
transactions = [
    ['Milk', 'Bread', 'Butter'],
    ['Beer', 'Bread'],
    ['Milk', 'Bread', 'Butter', 'Beer'],
    ['Milk', 'Bread', 'Butter'],
    ['Bread', 'Butter']
]

# Minimum support threshold
min_support = 2

# Total number of transactions
num_transactions = len(transactions)

# Function to generate candidate itemsets of size k
def generate_candidates(frequent_itemsets, k):
    items = set()
    for itemset in frequent_itemsets:
        items.update(itemset)
    items = sorted(items)
    return [frozenset(comb) for comb in combinations(items, k)]

# Function to get itemsets with support >= min_support
def get_frequent_itemsets(transactions, candidates, min_support):
    itemset_counts = {}
    for transaction in transactions:
        transaction = set(transaction)
        for candidate in candidates:
            if candidate.issubset(transaction):
                itemset_counts[candidate] = itemset_counts.get(candidate, 0) + 1
    return [(itemset, count) for itemset, count in itemset_counts.items() if count >= min_support]

# Apriori algorithm implementation
def apriori(transactions, min_support):
    items = set(item for transaction in transactions for item in transaction)
    candidates = [frozenset([item]) for item in items]

    frequent_itemsets = []
```

```
    k = 1
    while candidates:
        current_frequent_itemsets = get_frequent_itemsets(transactions, candidates, min_support)
        if not current_frequent_itemsets:
            break
        frequent_itemsets.extend(current_frequent_itemsets)

        print(f"Frequent itemsets of size {k} (Support ≥ {min_support}):")
        for itemset, count in current_frequent_itemsets:
            print(f"{list(itemset)} : {count}")
        print("-" * 40)

        k += 1
        candidates = generate_candidates([itemset for itemset, _ in current_frequent_itemsets], k)

    return frequent_itemsets

# Generate Association Rules
def generate_association_rules(frequent_itemsets):
    print("\nAssociation Rules:")
    for itemset, itemset_count in frequent_itemsets:
        if len(itemset) >= 2:
            subsets = [frozenset(x) for i in range(1, len(itemset)) for x in combinations(itemset, i)]
            for antecedent in subsets:
                consequent = itemset - antecedent
                if consequent:
                    antecedent_count = next((count for i, count in frequent_itemsets if i == antecedent), 0)
                    if antecedent_count > 0:
                        confidence = itemset_count / antecedent_count
                        support = itemset_count / num_transactions
                        consequent_count = next((count for i, count in frequent_itemsets if i == consequent), 0)
                        lift = confidence / (consequent_count / num_transactions) if consequent_count else 0
                        print(f"Rule: {list(antecedent)} -> {list(consequent)}")
                        print(f"Support: {support:.2f}, Confidence: {confidence:.2f}, Lift: {lift:.2f}")
                        print("-" * 50)

# Run the Apriori algorithm
frequent_itemsets = apriori(transactions, min_support)

# Final Result Output
if not frequent_itemsets:
    print("No frequent itemsets found.")
else:
    print("\n✅ Final Frequent Itemsets (Support ≥ 2):")
    for itemset, count in frequent_itemsets:
        print(f"{list(itemset)} : {count}")

# Generate and print Association Rules
generate_association_rules(frequent_itemsets)
```

Output:

```
[Running] python -u "e:\ADBS\exp11.py"
Frequent itemsets of size 1 (Support ≥ 2):
['Milk'] : 3
['Bread'] : 5
['Butter'] : 4
['Beer'] : 2
----------------------------------------
Frequent itemsets of size 2 (Support ≥ 2):
['Bread', 'Butter'] : 4
['Milk', 'Bread'] : 3
['Milk', 'Butter'] : 3
['Beer', 'Bread'] : 2
----------------------------------------
Frequent itemsets of size 3 (Support ≥ 2):
['Milk', 'Bread', 'Butter'] : 3
----------------------------------------

✅ Final Frequent Itemsets (Support ≥ 2):
['Milk'] : 3
['Bread'] : 5
['Butter'] : 4
['Beer'] : 2
['Bread', 'Butter'] : 4
['Milk', 'Bread'] : 3
['Milk', 'Butter'] : 3
['Beer', 'Bread'] : 2
['Milk', 'Bread', 'Butter'] : 3
```

```
Association Rules:
Rule: ['Bread'] -> ['Butter']
Support: 0.80, Confidence: 0.80, Lift: 1.00
--------------------------------------------------
Rule: ['Butter'] -> ['Bread']
Support: 0.80, Confidence: 1.00, Lift: 1.00
--------------------------------------------------
Rule: ['Milk'] -> ['Bread']
Support: 0.60, Confidence: 1.00, Lift: 1.00
--------------------------------------------------
Rule: ['Bread'] -> ['Milk']
Support: 0.60, Confidence: 0.60, Lift: 1.00
--------------------------------------------------
Rule: ['Milk'] -> ['Butter']
Support: 0.60, Confidence: 1.00, Lift: 1.25
--------------------------------------------------
Rule: ['Butter'] -> ['Milk']
Support: 0.60, Confidence: 0.75, Lift: 1.25
--------------------------------------------------
Rule: ['Beer'] -> ['Bread']
Support: 0.40, Confidence: 1.00, Lift: 1.00
--------------------------------------------------
Rule: ['Bread'] -> ['Beer']
Support: 0.40, Confidence: 0.40, Lift: 1.00
--------------------------------------------------
Rule: ['Milk'] -> ['Bread', 'Butter']
Support: 0.60, Confidence: 1.00, Lift: 1.25
--------------------------------------------------
Rule: ['Bread'] -> ['Milk', 'Butter']
Support: 0.60, Confidence: 0.60, Lift: 1.00
--------------------------------------------------
Rule: ['Butter'] -> ['Milk', 'Bread']
Support: 0.60, Confidence: 0.75, Lift: 1.25
--------------------------------------------------
Rule: ['Milk', 'Bread'] -> ['Butter']
Support: 0.60, Confidence: 1.00, Lift: 1.25
--------------------------------------------------
```

Conclusion: Students are able to implement A-priori algorithm for Data mining