Assignment
Association Rule Mining.

1. Implement Apriori and FP-growth algorithm in your preferred language.

→ Apriori Algorithm

```
def generate_candidates (prev_candidates, k):
    candidates = set()
    for item1 in prev_candidates:
        for item2 in prev_candidates:
            union = item1.union(item2)
            if len(union) == k:
                candidates.add (union)
    return candidates

def prune_candidates (transactions, candidates, min_support:
    pruned_candidates = [ ]
    for candidate in candidates:
        count = sum(1 for transaction in transactions if
          candidate.issubset(transaction))
        support = count / len(transactions)
        if support >= min_support:
            pruned_candidates.append (candidate)
    return pruned_candidates.

def apriori (transactions, min_support):
    itemset = [(item) for transaction in transactions for item
      in transaction]
```

```
frequent_itemsets = []
k = 1
while itemset:
    candidates = generate_candidates(itemset, k)
    pruned_candidates = prune_candidates(transactions,
        candidates, min_support)
    frequent_itemsets.extend(pruned_candidates)
    itemset = pruned_candidates
    k += 1
return frequent_itemsets

transactions = [
    {'apple', 'banana', 'orange'},
    {'banana', 'orange'},
    {'apple', 'banana'},
    {'banana', 'orange'},
    {'apple', 'orange'}
]

min_support = 0.4
frequent_itemsets_apriori = apriori(transactions, min_support)
print("Frequent Itemsets (Apriori):")
for itemset in frequent_itemsets_apriori:
    print(itemset)
```

FP-growth Algorithm

```
from collections import defaultdict

class TreeNode:
    def __init__(self, item, count, parent):
        self.item = item
        self.count = count
        self.parent = parent
        self.children = {}

def build_tree(transactions, min_support):
    header_table = defaultdict(int)
    for transaction in transactions:
        for item in transaction:
            header_table[item] += 1
    for item in list(header_table):
        if header_table[item] < min_support:
            del header_table[item]
    frequent_items = set(header_table.keys())
    if not frequent_items:
        return None, None
    for item in header_table:
        header_table[item] = [header_table[item], None]
```

```
root = TreeNode(None, 1, None)
for transaction in transactions:
    filtered_transaction = [item for item in transaction if item
      in frequent_items]
    if filtered_transaction:
      sorted_transaction = sorted(filtered_transaction, key=lambda
      x: header_table[x][0], reverse=True)
      insert_tree(sorted_transaction, root, header_table)
return root, header_table


def find_prefix_path (item, node):
    paths=[]
  while node is not None:
    path=[]
    current = node
    while current.parent is not None:
      path.append(current.item)
      current = current.parent
    if len(path) > 1:
      paths.append(path[1:])
    node = node.next
return paths.
```

transactions = [{'A', 'B', 'C', 'D'}, {'A', 'B', 'D'}, {'A', 'B', 'E'}, {'A', 'B', 'D', 'E'}, {'B', 'C', 'E'}]

print (frequent_itemsets_fp_growth(transactions, min_support=0.2))

3. What does 'best rules' mean in ARM? What criterion is used to decide for the best rules?

Ans → In Association Rule Mining (ARM), "best rules" refer to the rules that are considered the most interesting or useful based on certain evaluation criteria.

The criterion that is used to decide for the best rules are :-

1) Support – ~~How~~ The support of a rule is the portion of transactions in the dataset. Higher support indicates that the rule is more frequently observed in the dataset.

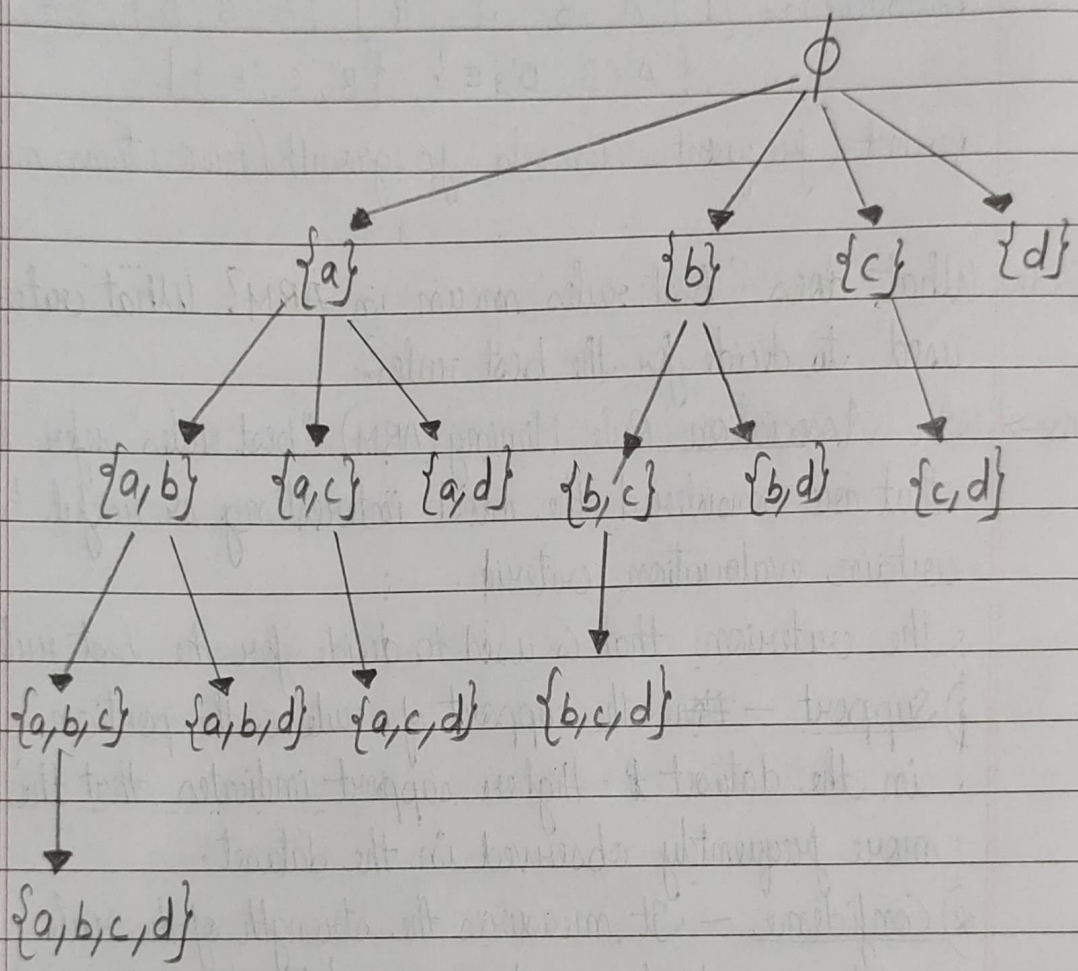
2) Confidence – It measures the strength of the rule. Higher confidence indicates a stronger relationship between the antecedant and the consequent.

3) Lift – Lift measures the strength of the association between the antecedent and the consequent. A lift value greater than 1 indicates a positive association, while less than 1 indicates a negative association.

4) Conviction – Conviction measures the degree of dependency between the antecedent & consequent. Higher conviction values indicate stronger dependencies.

4. Draw an enumeration tree for four items A, B, C, D?

Ans →



5. The following database has five transactions in Table 1. Use Apriori algorithm to find all the frequent item sets with minimum support 3.

TID	data	items (bought)
T1	10/15/00	{K, A, D, B}
T2	10/15/00	{D, A, C, E, B}
T3	10/15/00	{C, A, D, B}
T4	10/20/00	{A, B, D}
T5	10/18/00	{C, D, E}

Table 1.

Given, minimum support = 3

Sol^n →

1-item sets	Frequency
A	4
B	4
C	3
D	5
E	2
K	1

Frequency 1-item sets	Frequency
A	4
B	4
C	3
D	5

2-item sets	Frequency
A, B	4
A, C	1
A, D	4
B, C	2
B, D	4
C, D	3

Frequency 2-item sets	Frequency
A, B	4
A, D	4
B, D	4
C, D	3

~~Frequency~~ 3-item sets	Frequency
A, B, C	2
A, B, D	4
A, C, D	2
B, C, D	2

Frequency 3-item sets	Frequency
A, B, D	4

6. Answer the following questions:

a. Suppose you are given a task of finding all the association rules in a database whose supports are between 20% and 80%, and the accuracy of the rules should be above 70%. Change the algorithm scheme of Apriori to find all the rules satisfying the above requirement.

Ans → Step 1: Initialization

- Set the minimum support threshold to 20% and the maximum support threshold to 80%.
- Set the minimum confidence threshold to 70%

Step 2: Generate Frequent Itemsets

- Generate frequent itemsets using the Apriori algorithm, ensuring that each itemset's support falls within the specified range.

Step 3: Generate Association Rules

- For each frequent itemset with more than one item:
  - Generate all possible association rules by splitting the itemset into antecedent and consequent parts.
  - Calculate the confidence of each rule.
  - Filter out rules that donot meet the minimum confidence threshold.

Step 4: Output

- Return the set of association rules that satisfy the support and confidence criteria.

b) Applying your algorithm to the following dataset and present you all the rules you found.

Transaction ID	items
T1	I_1, I_3, I_5
T2	I_3, I_2, I_4
T3	I_1, I_2, I_4
T4	I_1, I_3
T5	I_2, I_3, I_5
T6	I_1, I_2, I_3
T7	I_3, I_4, I_6
T8	I_1, I_2, I_4

Given,
Min support between 20% & 80%.
i.e Min support should be better between ~~4.6 to 9.6~~
~~2.6 to 9.6~~
1.6 to 6.4

Sol^n →

1-item sets	Frequency
I_1	5
I_2	5
I_3	6
I_4	4
I_5	2
I_6	1

Frequency 1-item Sets	Frequency
I_1	5
I_2	5
I_3	6
I_4	4
I_5	2

2-item sets	Frequency
I_1, I_2	3
I_1, I_3	3
I_1, I_4	2
I_1, I_5	1
I_2, I_3	3
I_2, I_4	3

2-item sets	Frequency
I_2, I_5	1
I_3, I_4	2
I_3, I_5	2
I_4, I_5	0

Frequency 2-item sets	Frequency
I_1, I_2	3
I_1, I_3	3
I_1, I_4	2
I_2, I_3	3
I_2, I_4	3
I_3, I_4	2
I_3, I_5	2

~~Frequency~~ 3-item sets	Frequency
I_1, I_2, I_3	1
I_1, I_2, I_4	2
I_1, I_2, I_5	0
I_1, I_3, I_4	0
I_1, I_3, I_5	1
I_1, I_4, I_5	0
I_2, I_3, I_4	1
I_2, I_3, I_5	1
I_2, I_4, I_5	0
I_3, I_4, I_5	0

Frequency 3-item sets	Frequency
I_1, I_2, I_4	2

Frequency 3-Item set $= 1 \Rightarrow \{I_1, I_2, I_4\}$

Non empty Subset are

$\rightarrow \{\{I_1\}, \{I_2\}, \{I_4\}, \{I_1, I_2\}, \{I_1, I_4\}, \{I_2, I_4\}$

Rule: $X \rightarrow Y$ $\rightarrow$ Support $= \frac{freq(X,Y)}{N}$

Confidence $= \frac{freq(X,Y)}{freq(X)}$

Rule 1: $\{I_1\} \rightarrow \{I_2, I_4\}$

Support $= \frac{2}{8} \times 100\% = 25\%$

Confidence $= \frac{2}{5} \times 100\% = 40\%$ which is not greater than 70%

So, not valid

Rule 2: $\{I_2\} \rightarrow \{I_1, I_4\}$

Support $= \frac{2}{8} \times 100\% = 25\%$

Confidence $= \frac{2}{5} \times 100\% = 40\%$ which is less than 70%

So, not valid

Rule 3: $\{I_4\} \rightarrow \{I_1, I_2\}$

Support $= \frac{2}{8} \times 100\% = 25\%$

Confidence $= \frac{2}{4} \times 100\% = 50\%$ which is less than 70%

So, not valid

Rule 4: $\{I_1, I_2\} \rightarrow \{I_4\}$

Support $= \frac{2}{8} \times 100\% = 25\%$

Confidence $= \frac{2}{3} = 66.67\%$ which is less than 70%

So, not valid.

Rule 5: $\{I_1, I_4\} \rightarrow \{I_2\}$

Support $= \frac{2}{8} \times 100\% = 25\%$

Confidence $= \frac{2}{2} \times 100\% = 100\%$ which is greater than 70%.

So, it is valid.

Rule 6: $\{I_2, I_4\} \rightarrow \{I_1\}$

Support $= \frac{2}{8} \times 100\% = 25\%$

Confidence $= \frac{2}{3} \times 100\% = 66.67\%$ which is not greater than 70%.

So, it is not valid.

7. Compare the difference between association and classification, and discuss how we can we adapt an association algorithm to ~~algorith~~ classification.

Ans → The difference between association and classification are:-

1) Association mining focuses on discovering relationships between variables without the need for predefined class labels, while classification aims to predict class labels for instances based on their features.

2) Association typically deals with transactional data or datasets where the relationships between items are of interest, while classification often deals with datasets where each instance is labelled with a class.

3) Association rules are typically in the form of "if X then Y", indicating associations between itemsets, while classification models output class labels for instances.

We can adapt association algorithm to classification in some scenarios, especially when dealing with high-dimensional data or when the class labels are not readily available:

1) Feature Engineering
- We can use association rules to discover meaningful patterns in the data and derive new features that can be used for classification.
- These derived features can capture complex relationships between variables, which may improve the performance of classification models.

2) Rule-Based Classifiers
- We can use association rules directly as a basis for rule-based classification.
- Each rule can serve as a decision rule for assigning class labels to instances based on their feature values.

9. Consider the market basket transaction shown in fig.

a) What is the maximum number of association rules that can be extracted from this data (including rules that have zero support)?

b) What is the maximum of frequent itemsets that can be

c) Write an expression for the maximum number of size 3 itemsets that can be derived from this data set.

d) Find an itemset (of size 2 or larger) that has the largest support.

e) Find a pair of items, a and b, such that the rules {a} → {b} and {b} → {a} have the same confidence.

Ans → a) 6 unique items i.e. (Milk, Beer, Diaper, Bread, Butter, Cookies)

$3^d - 2^{(d+1)} + 1$

$= 3^6 - 2^7 + 1$

$= 729 - 128 + 1$

$= 602$ possible rules.

b) Itemset {Milk, Diapers, Bread, Butter} has the biggest size with 4 items and is frequent (sup({Milk, Diapers, Bread, Butter}) = 2)

c) The minimum num of size = 3 itemsets is given by C(6,3), where C = combination

$$C(6,3) = \frac{6!}{3!\,3!} = \frac{6 \times 5 \times 4 \times \cancel{3!}}{\cancel{3} \times \cancel{2} \times \cancel{3!}} = 20$$

d & e

Initial supports (in descending order)

Item	Support
Diapers	7
Milk	5
Bread	5
Butter	5
Beer	4
Cookies	4

All 6 itemsets with one item are frequent

Sets with two items:

Itemset	Support
{Diapers, Milk}	4
{Diapers, Bread}	3
{Diapers, Butter}	3
{Diapers, Beer}	3
{Diapers, Cookies}	2
{Milk, Bread}	3
{Milk, Butter}	2
{Milk, Beer}	1
{Milk, Cookies}	1
{Bread, Butter}	5
{Bread, Beer}	0
{Bread, Cookies}	1
{Butter, Beer}	0
{Butter, Cookies}	1
{Beer, Cookies}	2

d) We only need to watch the supports of itemsets with 2 items, since adding items will never grow the support. The support will stay either the same or decrease.
Based on the table above:
{Bread, Butter} & {Diapers, Milk} have the largest support.

e) The formula for confidence is
$conf(A \rightarrow B) = sup(\{A, B\}) / sup(\{A\})$, where sup stands for support.

It can easily be seen from the tables in previous exercise that Bread and Butter are having the same initial support and therefore rules Bread → Butter and Butter → Bread are having the same confidence. Proving this,
$conf(Bread \rightarrow Butter) = sup(\{Bread, Butter\}) / sup(\{Bread\})$
$= 5/5 = 1$

and
$conf(Butter \rightarrow Bread) = sup(\{Bread, Butter\}) / sup(\{Butter\})$
$= 5/5 = 1$

Actually if the denominators are the same then the confidence has to be the same. Based on this also Milk and Bread, Milk and Butter, Beer and Cookies suffice the condition.

2. Modify the algorithms to achieve the same task (preferably with some improvement). Clearly mention the difference in the modified algorithm.

Ans → Apriori Algorithm

```
from collections import defaultdict
from itertools import combinations

class AprioriImproved:
    def __init__(self, ~~transact~~ min_support=0.5):
        self.min_support = min_support

    def generate_itemsets(self, transactions, k):
        item_counts = defaultdict(int)
        for transaction in transactions:
            for itemset in combinations(transaction, k):
                item_counts[itemset] += 1
        return {itemset: count for itemset, count in item_counts.
            items() if count >= self.min_support}

    def fit(self, transactions):
        itemsets = [{(item,): count for item, count in
            transaction.items()} for transaction in transactions]
        frequent_itemsets = []
        k = 1
```

```
while itemsets:
    frequent_itemsets.extend([(item, count)])
    itemsets = [self.generate_itemsets(transactions, k+1)]
    k += 1
return frequent_itemsets
```

In this improved version of Apriori

1) Instead of generating candidate itemsets using a join step, we directly generate all possible itemsets of size k from the transactions using the 'combinations' function.
2) We use a hash tree structure to efficiently count the support of each item.

Modified FP Growth Algorithm

1) Path Compression → Compress paths of frequent itemset during tree construction
2) Verticle Data Representation → store transaction vertically for better cache locality.