IT 414

Assignment 2 - FP-Tree Algorithm

Name: Niraj Nandish

Roll no.: 191IT234

```
import pandas as pd
from collections import defaultdict
from itertools import chain, combinations
from IPython.display import display, HTML
```

```
class Node:
    def __init__(self, item, frequency, parent):
        self.item = item
        self.count = frequency
        self.parent = parent
        self.children = {}
        self.next = None

    def increment(self, frequency):
        self.count += frequency
```

```
def update_header_table(item, targetNode, header_table):
    if header_table[item][1] == None:
        header_table[item][1] = targetNode
    else:
        curr_node = header_table[item][1]
        while curr_node.next != None:
            curr_node = curr_node.next
        curr_node.next = targetNode

def update_FP_tree(item, treeNode, header_table, frequency):
    if item in treeNode.children:
        treeNode.children[item].increment(frequency)
    else:
        newItemNode = Node(item, frequency, treeNode)
        treeNode.children[item] = newItemNode
        update_header_table(item, newItemNode, header_table)

    return treeNode.children[item]
```

```
    return treeNode.children[item]

def build_FP_tree(complete_itemset_list, frequency, min_support_count):
    header_table = defaultdict(int)

    for idx, itemset in enumerate(complete_itemset_list):
        for item in itemset:
            header_table[item] += frequency[idx]

    header_table = dict((item, sup) for item, sup in header_table.items() if sup >= min_support_count)
    if(len(header_table) == 0):
        return None, None

    for item in header_table:
        header_table[item] = [header_table[item], None]

    FP_tree = Node('Null', 1, None)

    for idx, itemset in enumerate(complete_itemset_list):
        itemset = [item for item in itemset if item in header_table]
        itemset.sort(key=lambda item: header_table[item][0], reverse=True)
        curr_node = FP_tree
        for item in itemset:
            curr_node = update_FP_tree(item, curr_node, header_table, frequency[idx])

    return FP_tree, header_table

def traverse(node, prefixPath):
    if node.parent != None:
        prefixPath.append(node.item)
        traverse(node.parent, prefixPath)

def findPrefixPath(basePat, header_table):
    treeNode = header_table[basePat][1]
    condPats = []
    frequency = []
    while treeNode != None:
        prefixPath = []
        traverse(treeNode, prefixPath)
        if len(prefixPath) > 1:
            condPats.append(prefixPath[1:])
            frequency.append(treeNode.count)
        treeNode = treeNode.next
    return condPats, frequency

def full_traverse(header_table, min_support_count, preFix, freqItemList):
    sortedItemList = [item[0] for item in sorted(list(header_table.items()), key=lambda p:p[1][0])]
    for item in sortedItemList:
```

```
        newFreqSet = preFix.copy()
        newFreqSet.add(item)
        freqItemList.append(newFreqSet)
        conditionalPattBase, frequency = findPrefixPath(item, header_table)
        conditionalTree, new_header_table = build_FP_tree(conditionalPattBase, frequency, min_support_count)
        if new_header_table != None:
            full_traverse(new_header_table, min_support_count, newFreqSet, freqItemList)

def powerset(s):
    return chain.from_iterable(combinations(s, r) for r in range(1, len(s)))

def get_support(testSet, complete_itemset_list):
    count = 0
    for itemset in complete_itemset_list:
        if(set(testSet).issubset(itemset)):
            count += 1
    return count

def assoc_rules(freq_itemset, complete_itemset_list, min_confidence, freq_itemset_length):
    rules = []
    for itemset in freq_itemset:
        if len(itemset) < freq_itemset_length:
            continue
        subsets = powerset(itemset)
        itemsetSup = get_support(itemset, complete_itemset_list)
        for s in subsets:
            confidence = float(itemsetSup / get_support(s, complete_itemset_list))
            if(confidence > min_confidence):
                rules.append([set(s), set(itemset.difference(s)), confidence])
    return rules
```

Q1 - All Electronics dataset

Support count = 2

Minimum confidence threshold = 70%

Frequent item set count - 3

```
raw_df = pd.read_excel('AllElectronics.xlsx', sheet_name='Sheet1', header=None)
raw_df.head()
```

	0	1	2	3
0	I1	I2	I5	NaN
1	I2	I4	NaN	NaN
2	I2	I3	NaN	NaN
3	I1	I2	I4	NaN

```
data = []

for idx, row in enumerate(raw_df.to_numpy()):
  row = [item for item in row if str(item) != 'nan']
  data.append([f'T{idx + 1}', set(map(lambda x: str(x), row))])
```

```
df = pd.DataFrame(data, columns=['TID', 'items_bought'])
df.head()
```

	TID	items_bought
0	T1	{I2, I5, I1}
1	T2	{I2, I4}
2	T3	{I2, I3}
3	T4	{I2, I1, I4}
4	T5	{I1, I3}

```
list_of_items = df['items_bought'].tolist()
frequency_of_items = [1 for row in list_of_items]
min_support_count = 2
min_confidence = 0.7
freq_itemset_length = 3
```

```
fp_tree, header_table = build_FP_tree(list_of_items, frequency_of_items, min_support_count)
```

```
freq_itemsets = []
full_traverse(header_table, min_support_count, set(), freq_itemsets)
rules = assoc_rules(freq_itemsets, list_of_items, min_confidence, freq_itemset_length)
```

```
print('Frequent Itemsets:')
for row in freq_itemsets:
  print(f'{", ".join(map(lambda x: str(x), row))} ({get_support(row, list_of_items)})')
```

```
Frequent Itemsets:
I5 (2)
I5, I1 (2)
I2, I5 (2)
I2, I5, I1 (2)
I4 (2)
I4, I2 (2)
I1 (6)
I2, I1 (4)
I3 (6)
I2, I3 (4)
I2, I1, I3 (2)
I1, I3 (4)
I2 (7)
```

```
print('Rules:')
for row in rules:
  a = list(row[0])
  b = list(row[1])
  print(f'{", ".join(a)} -> {", ".join(b)} ({row[2]})')
```

```
Rules:
I5 -> I2, I1 (1.0)
I2, I5 -> I1 (1.0)
I5, I1 -> I2 (1.0)
```

Q2 - Goods dataset

Support count = 2

Minimum confidence threshold = 70%

Frequent item set count - 4

```
raw_df = pd.read_excel('GoodsServiceDataset.xlsx', sheet_name='Sheet1', nrows=100)
raw_df.head()
```

	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	...	Unnamed: 66	Unnamed: 67	Unnamed: 68	Unnamed: 69	Unnamed: 70	Unnamed: 71	Unnamed: 72	Unnamed: 73	Unnamed: 74	Unnar
0	30.0	31.0	32.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	33.0	34.0	35.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	36.0	37.0	38.0	39.0	40.0	41.0	42.0	43.0	44.0	45.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

```
data = []

for idx, row in enumerate(raw_df.to_numpy()):
  row = [item for item in row if str(item) != 'nan']
  data.append([f'T{idx + 1}', set(map(lambda x: int(x), row))])
```

```
df = pd.DataFrame(data, columns=['TID', 'items_bought'])
df.head()
```

	TID	items_bought
0	T1	{32, 30, 31}
1	T2	{33, 34, 35}
2	T3	{36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46}
3	T4	{48, 47, 38, 39}
4	T5	{38, 39, 48, 49, 50, 51, 52, 53, 54, 55, 56, 5...

```
list_of_items = df['items_bought'].tolist()
frequency_of_items = [1 for row in list_of_items]
min_support_count = 3
min_confidence = 0.7
freq_itemset_length = 4
```

```
fp_tree, header_table = build_FP_tree(list_of_items, frequency_of_items, min_support_count)
```

```
freq_itemsets = []
full_traverse(header_table, min_support_count, set(), freq_itemsets)
rules = assoc_rules(freq_itemsets, list_of_items, min_confidence, freq_itemset_length)
```

```
print('Frequent Itemsets:')
for row in freq_itemsets:
  print(f'{", ".join(map(lambda x: str(x), row))} ({get_support(row, list_of_items)})')
```

Frequent Itemsets:
37 (3)
37, 38 (3)
60 (3)
79 (3)
39, 79 (3)
147 (3)
48, 147 (3)
161 (3)
170 (3)
170, 38 (3)
48, 170 (3)
48, 170, 38 (3)
170, 39 (3)
48, 170, 39 (3)
170, 38, 39 (3)
48, 170, 38, 39 (3)
179 (3)
186 (3)
237 (3)
237, 39 (3)
242 (3)
242, 39 (3)
258 (3)
310 (3)
340 (3)
89 (4)
89, 39 (3)
48, 89, 39 (3)
48, 89 (4)
105 (4)
105, 38 (3)
105, 38, 39 (3)
105, 39 (4)
110 (4)
41, 110 (3)
38, 110 (3)
110, 39 (3)
38, 110, 39 (3)
152 (4)
152, 39 (3)
225 (4)
65 (5)
48, 65 (3)
48, 65, 39 (3)
65, 39 (4)
36 (8)
41, 36 (3)
41, 36, 38 (3)
48, 36 (4)
48, 36, 38 (4)

```
    36, 39 (6)
    36, 38, 39 (6)
    36, 38 (8)
    32 (10)
    32, 38 (3)
    32, 38, 39 (3)
```

```
print('Rules:')
for row in rules:
  a = map(lambda x: str(x), list(row[0]))
  b = map(lambda x: str(x), list(row[1]))
  print(f'{", ".join(a)} -> {", ".join(b)} ({row[2]})')
```

```
    Rules:
    170 -> 48, 38, 39 (1.0)
    48, 170 -> 38, 39 (1.0)
    170, 38 -> 48, 39 (1.0)
    170, 39 -> 48, 38 (1.0)
    48, 170, 38 -> 39 (1.0)
    48, 170, 39 -> 38 (1.0)
    170, 38, 39 -> 48 (1.0)
    32, 38 -> 48, 39 (1.0)
    32, 48, 38 -> 39 (1.0)
    32, 38, 39 -> 48 (1.0)
    32, 41 -> 48, 39 (0.75)
    32, 41, 48 -> 39 (1.0)
    32, 41, 39 -> 48 (1.0)
```