A4_fpgrowth-1

September 19, 2022

[ ]:
```python
from collections import defaultdict, OrderedDict
from csv import reader
from itertools import chain, combinations
from optparse import OptionParser
import pandas as pd

# if not installed yet: pip install mlxtend
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import fpgrowth, association_rules
```

0.1 Find frequent itemsets using FPGrowth

[ ]:
```python
dataset = [['Corn', 'Light Cream', 'Chicken', 'Beef', 'Wine', 'Ice Cream'],
           ['Dill', 'Onion', 'Carrot', 'Beef', 'Wine', 'Ice Cream'],
           ['Milk', 'Wine', 'Beef', 'Ice Cream'],
           ['Light Cream', 'Chicken', 'Corn', 'Kidney Beans', 'Yogurt', 'Wine'],
           ['Corn', 'Onion', 'Light Cream', 'Kidney Beans', 'Chicken',␣
 ↪'Yogurt']]

te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)

frequent_itemsets = fpgrowth(df, min_support=0.6, use_colnames=True)
frequent_itemsets
```

[ ]:

	support	itemsets
0	0.8	(Wine)
1	0.6	(Light Cream)
2	0.6	(Ice Cream)
3	0.6	(Corn)
4	0.6	(Chicken)
5	0.6	(Beef)
6	0.6	(Wine, Ice Cream)
7	0.6	(Light Cream, Corn)
8	0.6	(Corn, Chicken)
9	0.6	(Light Cream, Chicken)

```
10       0.6  (Light Cream, Corn, Chicken)
11       0.6               (Ice Cream, Beef)
12       0.6                    (Wine, Beef)
13       0.6         (Wine, Beef, Ice Cream)
```

0.1.1 The association rules can be found in given dataset with the minimum support 60% and the minimum confidence 70%

If we are only interested in rules with the confidence level above the 70 percent threshold (min_threshold=0.7), we can find the set of rules with the following specifications accordingly using *generate_rules()* function. 1. Specify your metric of interest 2. Threshold

[ ]:
```
association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)
```

[ ]:
```
               antecedents            consequents  antecedent support  \
0                   (Wine)            (Ice Cream)                 0.8
1              (Ice Cream)                 (Wine)                 0.6
2            (Light Cream)                 (Corn)                 0.6
3                   (Corn)          (Light Cream)                 0.6
4                   (Corn)              (Chicken)                 0.6
5                (Chicken)                 (Corn)                 0.6
6            (Light Cream)              (Chicken)                 0.6
7                (Chicken)          (Light Cream)                 0.6
8      (Light Cream, Corn)              (Chicken)                 0.6
9   (Light Cream, Chicken)                 (Corn)                 0.6
10         (Corn, Chicken)          (Light Cream)                 0.6
11           (Light Cream)        (Corn, Chicken)                 0.6
12                  (Corn)  (Light Cream, Chicken)                 0.6
13               (Chicken)     (Light Cream, Corn)                 0.6
14             (Ice Cream)                 (Beef)                 0.6
15                  (Beef)            (Ice Cream)                 0.6
16                  (Wine)                 (Beef)                 0.8
17                  (Beef)                 (Wine)                 0.6
18            (Wine, Beef)            (Ice Cream)                 0.6
19       (Wine, Ice Cream)                 (Beef)                 0.6
20       (Ice Cream, Beef)                 (Wine)                 0.6
21                  (Wine)      (Ice Cream, Beef)                 0.8
22                  (Beef)      (Wine, Ice Cream)                 0.6
23             (Ice Cream)           (Wine, Beef)                 0.6

    consequent support  support  confidence      lift  leverage  conviction
0                  0.6      0.6        0.75  1.250000      0.12         1.6
1                  0.8      0.6        1.00  1.250000      0.12         inf
2                  0.6      0.6        1.00  1.666667      0.24         inf
3                  0.6      0.6        1.00  1.666667      0.24         inf
4                  0.6      0.6        1.00  1.666667      0.24         inf
5                  0.6      0.6        1.00  1.666667      0.24         inf
```

6	0.6	0.6	1.00	1.666667	0.24	inf
7	0.6	0.6	1.00	1.666667	0.24	inf
8	0.6	0.6	1.00	1.666667	0.24	inf
9	0.6	0.6	1.00	1.666667	0.24	inf
10	0.6	0.6	1.00	1.666667	0.24	inf
11	0.6	0.6	1.00	1.666667	0.24	inf
12	0.6	0.6	1.00	1.666667	0.24	inf
13	0.6	0.6	1.00	1.666667	0.24	inf
14	0.6	0.6	1.00	1.666667	0.24	inf
15	0.6	0.6	1.00	1.666667	0.24	inf
16	0.6	0.6	0.75	1.250000	0.12	1.6
17	0.8	0.6	1.00	1.250000	0.12	inf
18	0.6	0.6	1.00	1.666667	0.24	inf
19	0.6	0.6	1.00	1.666667	0.24	inf
20	0.8	0.6	1.00	1.250000	0.12	inf
21	0.6	0.6	0.75	1.250000	0.12	1.6
22	0.6	0.6	1.00	1.666667	0.24	inf
23	0.6	0.6	1.00	1.666667	0.24	inf

The association rules can be found in given dataset with the minimum support 60% and the minimum lift value 1.2

[ ]:
```
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.2)
rules
```

[ ]:

	antecedents	consequents	antecedent support \
0	(Wine)	(Ice Cream)	0.8
1	(Ice Cream)	(Wine)	0.6
2	(Light Cream)	(Corn)	0.6
3	(Corn)	(Light Cream)	0.6
4	(Corn)	(Chicken)	0.6
5	(Chicken)	(Corn)	0.6
6	(Light Cream)	(Chicken)	0.6
7	(Chicken)	(Light Cream)	0.6
8	(Light Cream, Corn)	(Chicken)	0.6
9	(Light Cream, Chicken)	(Corn)	0.6
10	(Corn, Chicken)	(Light Cream)	0.6
11	(Light Cream)	(Corn, Chicken)	0.6
12	(Corn)	(Light Cream, Chicken)	0.6
13	(Chicken)	(Light Cream, Corn)	0.6
14	(Ice Cream)	(Beef)	0.6
15	(Beef)	(Ice Cream)	0.6
16	(Wine)	(Beef)	0.8
17	(Beef)	(Wine)	0.6
18	(Wine, Beef)	(Ice Cream)	0.6
19	(Wine, Ice Cream)	(Beef)	0.6
20	(Ice Cream, Beef)	(Wine)	0.6

```
21             (Wine)      (Ice Cream, Beef)                  0.8
22             (Beef)      (Wine, Ice Cream)                  0.6
23        (Ice Cream)           (Wine, Beef)                  0.6

    consequent support  support  confidence      lift  leverage  conviction
0                  0.6      0.6        0.75  1.250000      0.12         1.6
1                  0.8      0.6        1.00  1.250000      0.12         inf
2                  0.6      0.6        1.00  1.666667      0.24         inf
3                  0.6      0.6        1.00  1.666667      0.24         inf
4                  0.6      0.6        1.00  1.666667      0.24         inf
5                  0.6      0.6        1.00  1.666667      0.24         inf
6                  0.6      0.6        1.00  1.666667      0.24         inf
7                  0.6      0.6        1.00  1.666667      0.24         inf
8                  0.6      0.6        1.00  1.666667      0.24         inf
9                  0.6      0.6        1.00  1.666667      0.24         inf
10                 0.6      0.6        1.00  1.666667      0.24         inf
11                 0.6      0.6        1.00  1.666667      0.24         inf
12                 0.6      0.6        1.00  1.666667      0.24         inf
13                 0.6      0.6        1.00  1.666667      0.24         inf
14                 0.6      0.6        1.00  1.666667      0.24         inf
15                 0.6      0.6        1.00  1.666667      0.24         inf
16                 0.6      0.6        0.75  1.250000      0.12         1.6
17                 0.8      0.6        1.00  1.250000      0.12         inf
18                 0.6      0.6        1.00  1.666667      0.24         inf
19                 0.6      0.6        1.00  1.666667      0.24         inf
20                 0.8      0.6        1.00  1.250000      0.12         inf
21                 0.6      0.6        0.75  1.250000      0.12         1.6
22                 0.6      0.6        1.00  1.666667      0.24         inf
23                 0.6      0.6        1.00  1.666667      0.24         inf
```

The difference from the two sets of associations rules above is prominent with the rule wine -> beef: the confidence of the rule (0.75) barely reaches the threshhold of 1(a) (0.7), but the high lift (1.25) suggests a positive correlation between those 2. Then we can assume that wine could be a complement to dishes made of beef, so they're often bought together.

1 The algorithm to construct a FPGrowth tree from scratch

[ ]:
```python
class Node:
    def __init__(self, itemName, frequency, parentNode):
        self.itemName = itemName
        self.count = frequency
        self.parent = parentNode
        self.children = {}
        self.next = None

    def increment(self, frequency):
```

```
        self.count += frequency

    def display(self, ind=1):
        print('  ' * ind, self.itemName, ' ', self.count)
        for child in list(self.children.values()):
            child.display(ind+1)


def constructTree(itemSetList, frequency, minSup):
    headerTable = defaultdict(int)
    # Counting frequency and create header table
    for idx, itemSet in enumerate(itemSetList):
        for item in itemSet:
            headerTable[item] += frequency[idx]

    # Deleting items below minSup
    headerTable = dict((item, sup) for item, sup in headerTable.items() if sup␣
 ↪>= minSup)
    if(len(headerTable) == 0):
        return None, None

    # HeaderTable column [Item: [frequency, headNode]]
    for item in headerTable:
        headerTable[item] = [headerTable[item], None]

    # Init Null head node
    fpTree = Node('Null', 1, None)
    # Update FP tree for each cleaned and sorted itemSet
    for idx, itemSet in enumerate(itemSetList):
        itemSet = [item for item in headerTable if item in itemSet]
        # Traverse from root to leaf, update tree with given item
        currentNode = fpTree
        for item in itemSet:
            currentNode = updateTree(item, currentNode, headerTable,␣
 ↪frequency[idx])

    return fpTree, headerTable

def updateHeaderTable(item, targetNode, headerTable):
    if(headerTable[item][1] == None):
        headerTable[item][1] = targetNode
    else:
        currentNode = headerTable[item][1]
        # Traverse to the last node then link it to the target
        while currentNode.next != None:
            currentNode = currentNode.next
        currentNode.next = targetNode
```

```python
def updateTree(item, treeNode, headerTable, frequency):
    if item in treeNode.children:
        # If the item already exists, increment the count
        treeNode.children[item].increment(frequency)
    else:
        # Create a new branch
        newItemNode = Node(item, frequency, treeNode)
        treeNode.children[item] = newItemNode
        # Link the new branch to header table
        updateHeaderTable(item, newItemNode, headerTable)

    return treeNode.children[item]

def ascendFPtree(node, prefixPath):
    if node.parent != None:
        prefixPath.append(node.itemName)
        ascendFPtree(node.parent, prefixPath)

def findPrefixPath(basePat, headerTable):
    # First node in linked list
    treeNode = headerTable[basePat][1]
    condPats = []
    frequency = []
    while treeNode != None:
        prefixPath = []
        # From leaf node all the way to root
        ascendFPtree(treeNode, prefixPath)
        if len(prefixPath) > 1:
            # Storing the prefix path and it's corresponding count
            condPats.append(prefixPath[1:])
            frequency.append(treeNode.count)

        # Go to next node
        treeNode = treeNode.next
    return condPats, frequency

def mineTree(headerTable, minSup, preFix, freqItemList):
    # Sort the items with frequency and create a list
    sortedItemList = [item[0] for item in sorted(list(headerTable.items()), key=lambda p:p[1][0])]
    # Start with the lowest frequency
    for item in sortedItemList:
        # Pattern growth is achieved by the concatenation of suffix pattern with frequent patterns generated from conditional FP-tree
        newFreqSet = preFix.copy()
        newFreqSet.add(item)
```

```
        freqItemList.append(newFreqSet)
        # Find all prefix path, constrcut conditional pattern base
        conditionalPattBase, frequency = findPrefixPath(item, headerTable)
        # Construct conditonal FP Tree with conditional pattern base
        conditionalTree, newHeaderTable = constructTree(conditionalPattBase,␣
↪frequency, minSup)
        if newHeaderTable != None:
            # Mining recursively on the tree
            mineTree(newHeaderTable, minSup,
                       newFreqSet, freqItemList)

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

def getSupport(testSet, itemSetList):
    count = 0
    for itemSet in itemSetList:
        if(set(testSet).issubset(itemSet)):
            count += 1
    return count

def associationRule(freqItemSet, itemSetList, minConf):
    rules = []
    for itemSet in freqItemSet:
        subsets = powerset(itemSet)
        itemSetSup = getSupport(itemSet, itemSetList)
        for s in subsets:
            confidence = float(itemSetSup / getSupport(s, itemSetList))
            if(confidence > minConf):
                rules.append([set(s), set(itemSet.difference(s)), confidence])
    return rules

def getFrequencyFromList(itemSetList):
    frequency = [1 for i in range(len(itemSetList))]
    return frequency
```

[ ]:
```
def fpgrowth(itemSetList, minSupRatio, minConf):
    frequency = getFrequencyFromList(itemSetList)
    minSup = len(itemSetList) * minSupRatio
    fpTree, headerTable = constructTree(itemSetList, frequency, minSup)

    if(fpTree == None):
        print('No frequent item set')
    else:
        freqItems = []
        mineTree(headerTable, minSup, set(), freqItems)
        rules = associationRule(freqItems, itemSetList, minConf)
```

```python
        return freqItems, rules
```

[ ]:
```python
freqItemSet, rules = fpgrowth(dataset, minSupRatio=0.6, minConf=0.7)
freqItemSet, rules
```

[ ]:
```
([{'Corn'},
  {'Light Cream'},
  {'Corn', 'Light Cream'},
  {'Chicken'},
  {'Chicken', 'Light Cream'},
  {'Chicken', 'Corn'},
  {'Chicken', 'Corn', 'Light Cream'},
  {'Beef'},
  {'Ice Cream'},
  {'Ice Cream', 'Wine'},
  {'Beef', 'Ice Cream'},
  {'Beef', 'Ice Cream', 'Wine'},
  {'Wine'},
  {'Beef', 'Wine'}],
 [[{'Light Cream'}, {'Corn'}, 1.0],
  [{'Corn'}, {'Light Cream'}, 1.0],
  [{'Light Cream'}, {'Chicken'}, 1.0],
  [{'Chicken'}, {'Light Cream'}, 1.0],
  [{'Corn'}, {'Chicken'}, 1.0],
  [{'Chicken'}, {'Corn'}, 1.0],
  [{'Light Cream'}, {'Chicken', 'Corn'}, 1.0],
  [{'Corn'}, {'Chicken', 'Light Cream'}, 1.0],
  [{'Chicken'}, {'Corn', 'Light Cream'}, 1.0],
  [{'Corn', 'Light Cream'}, {'Chicken'}, 1.0],
  [{'Chicken', 'Light Cream'}, {'Corn'}, 1.0],
  [{'Chicken', 'Corn'}, {'Light Cream'}, 1.0],
  [{'Ice Cream'}, {'Wine'}, 1.0],
  [{'Wine'}, {'Ice Cream'}, 0.75],
  [{'Ice Cream'}, {'Beef'}, 1.0],
  [{'Beef'}, {'Ice Cream'}, 1.0],
  [{'Ice Cream'}, {'Beef', 'Wine'}, 1.0],
  [{'Beef'}, {'Ice Cream', 'Wine'}, 1.0],
  [{'Wine'}, {'Beef', 'Ice Cream'}, 0.75],
  [{'Beef', 'Ice Cream'}, {'Wine'}, 1.0],
  [{'Ice Cream', 'Wine'}, {'Beef'}, 1.0],
  [{'Beef', 'Wine'}, {'Ice Cream'}, 1.0],
  [{'Wine'}, {'Beef'}, 0.75],
  [{'Beef'}, {'Wine'}, 1.0]])
```

1.1 Report

The frequent itemsets are threshheld by support, which indicates the popularity of itemsets among transactions. We can notice that (wine) is the most popular with (light cream, chicken, corn) and (wine, beef, ice cream) being the maximal frequent itemsets, so I would suggest a bundle of wine, beef, ice cream to combine these 2 aspects. Then we look at the rules generated from itemsets. For the rules of high confidence, patterns behind the consumer behaviour can be deduced, e.g. (beef, wine) -> (ice cream) may suggest a remantic night in for couples, ice cream is always a delight after wining and dining. For rules of high lift, phenonmena can be observed, e.g. chicken -> corn, light cream and vice versa may suggest a popular recipe at the moment.