```
In [1]: 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
```

How to find frequent itemsets using FPGrowth?

```
In [2]: 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', 'Yogur

        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
```

Out[2]:

	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	(Ice Cream, Wine)
7	0.6	(Light Cream, Corn)
8	0.6	(Chicken, Corn)
9	0.6	(Light Cream, Chicken)
10	0.6	(Light Cream, Chicken, Corn)
11	0.6	(Ice Cream, Beef)
12	0.6	(Wine, Beef)
13	0.6	(Ice Cream, Wine, Beef)

Part 1(a):

What 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

In [3]:
```
association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)
```

Out[3]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	lever
0	(Ice Cream)	(Wine)	0.6	0.8	0.6	1.00	1.250000	
1	(Wine)	(Ice Cream)	0.8	0.6	0.6	0.75	1.250000	
2	(Light Cream)	(Corn)	0.6	0.6	0.6	1.00	1.666667	
3	(Corn)	(Light Cream)	0.6	0.6	0.6	1.00	1.666667	
4	(Chicken)	(Corn)	0.6	0.6	0.6	1.00	1.666667	
5	(Corn)	(Chicken)	0.6	0.6	0.6	1.00	1.666667	
6	(Light Cream)	(Chicken)	0.6	0.6	0.6	1.00	1.666667	
7	(Chicken)	(Light Cream)	0.6	0.6	0.6	1.00	1.666667	
8	(Light Cream, Chicken)	(Corn)	0.6	0.6	0.6	1.00	1.666667	
9	(Light Cream, Corn)	(Chicken)	0.6	0.6	0.6	1.00	1.666667	
10	(Chicken, Corn)	(Light Cream)	0.6	0.6	0.6	1.00	1.666667	
11	(Light Cream)	(Chicken, Corn)	0.6	0.6	0.6	1.00	1.666667	
12	(Chicken)	(Light Cream, Corn)	0.6	0.6	0.6	1.00	1.666667	
13	(Corn)	(Light Cream, Chicken)	0.6	0.6	0.6	1.00	1.666667	
14	(Ice Cream)	(Beef)	0.6	0.6	0.6	1.00	1.666667	
15	(Beef)	(Ice Cream)	0.6	0.6	0.6	1.00	1.666667	
16	(Wine)	(Beef)	0.8	0.6	0.6	0.75	1.250000	
17	(Beef)	(Wine)	0.6	0.8	0.6	1.00	1.250000	
18	(Wine, Ice Cream)	(Beef)	0.6	0.6	0.6	1.00	1.666667	
19	(Ice Cream, Beef)	(Wine)	0.6	0.8	0.6	1.00	1.250000	
20	(Wine, Beef)	(Ice Cream)	0.6	0.6	0.6	1.00	1.666667	
21	(Ice Cream)	(Wine, Beef)	0.6	0.6	0.6	1.00	1.666667	
22	(Wine)	(Ice Cream, Beef)	0.8	0.6	0.6	0.75	1.250000	
23	(Beef)	(Wine, Ice Cream)	0.6	0.6	0.6	1.00	1.666667	

Part 1(b):

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

In [4]:
```
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.2
rules
```

Out[4]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	lever
0	(Ice Cream)	(Wine)	0.6	0.8	0.6	1.00	1.250000	
1	(Wine)	(Ice Cream)	0.8	0.6	0.6	0.75	1.250000	
2	(Light Cream)	(Corn)	0.6	0.6	0.6	1.00	1.666667	
3	(Corn)	(Light Cream)	0.6	0.6	0.6	1.00	1.666667	
4	(Chicken)	(Corn)	0.6	0.6	0.6	1.00	1.666667	
5	(Corn)	(Chicken)	0.6	0.6	0.6	1.00	1.666667	
6	(Light Cream)	(Chicken)	0.6	0.6	0.6	1.00	1.666667	
7	(Chicken)	(Light Cream)	0.6	0.6	0.6	1.00	1.666667	
8	(Light Cream, Chicken)	(Corn)	0.6	0.6	0.6	1.00	1.666667	
9	(Light Cream, Corn)	(Chicken)	0.6	0.6	0.6	1.00	1.666667	
10	(Chicken, Corn)	(Light Cream)	0.6	0.6	0.6	1.00	1.666667	
11	(Light Cream)	(Chicken, Corn)	0.6	0.6	0.6	1.00	1.666667	
12	(Chicken)	(Light Cream, Corn)	0.6	0.6	0.6	1.00	1.666667	
13	(Corn)	(Light Cream, Chicken)	0.6	0.6	0.6	1.00	1.666667	
14	(Ice Cream)	(Beef)	0.6	0.6	0.6	1.00	1.666667	
15	(Beef)	(Ice Cream)	0.6	0.6	0.6	1.00	1.666667	
16	(Wine)	(Beef)	0.8	0.6	0.6	0.75	1.250000	
17	(Beef)	(Wine)	0.6	0.8	0.6	1.00	1.250000	
18	(Wine, Ice Cream)	(Beef)	0.6	0.6	0.6	1.00	1.666667	
19	(Ice Cream, Beef)	(Wine)	0.6	0.8	0.6	1.00	1.250000	
20	(Wine, Beef)	(Ice Cream)	0.6	0.6	0.6	1.00	1.666667	
21	(Ice Cream)	(Wine, Beef)	0.6	0.6	0.6	1.00	1.666667	
22	(Wine)	(Ice Cream, Beef)	0.8	0.6	0.6	0.75	1.250000	
23	(Beef)	(Wine, Ice Cream)	0.6	0.6	0.6	1.00	1.666667	

Part 1(c):

Examine the differences from the two sets of associations rules above. Select one rule that has low confidence and high lift value. Why is this rule interesting?

Answer:

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.

Part 1(d):

Describing results: A summary of results (number of rules, general description), and a selection of rules that you deem is interesting based on “interestingness" measure(s) of your choice. Describe your reasoning behind any choices you made.

Answer:

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.

Part 2:

In [5]:
```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 su
    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, frequenc

    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

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)
```

```python
        # 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()), k
    # Start with the lowest frequency
    for item in sortedItemList:
        # Pattern growth is achieved by the concatenation of suffix pattern w
        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, 
        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
```

In [6]:
```python
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)
        return freqItems, rules
```

Part 2(a):

What association rules can be found with the code provided in Part 2 with the minimum support 60% and the minimum confidence 70%? Comparing to the output from question 1(a), what do you think the cause is that leads to different outcome?

Answer:

The itemSet isn't properly sorted according to the headertable before it's used to construct the FPtree.

```
In [7]: freqItemSet, rules = fpgrowth(dataset, minSupRatio=0.6, minConf=0.7)
        freqItemSet, rules
```

```
Out[7]: ([{'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],
          [{'Chicken'}, {'Corn'}, 1.0],
          [{'Corn'}, {'Chicken'}, 1.0],
          [{'Light Cream'}, {'Chicken', 'Corn'}, 1.0],
          [{'Chicken'}, {'Corn', 'Light Cream'}, 1.0],
          [{'Corn'}, {'Chicken', 'Light Cream'}, 1.0],
          [{'Chicken', 'Light Cream'}, {'Corn'}, 1.0],
          [{'Corn', 'Light Cream'}, {'Chicken'}, 1.0],
          [{'Chicken', 'Corn'}, {'Light Cream'}, 1.0],
          [{'Wine'}, {'Ice Cream'}, 0.75],
          [{'Ice Cream'}, {'Wine'}, 1.0],
          [{'Ice Cream'}, {'Beef'}, 1.0],
          [{'Beef'}, {'Ice Cream'}, 1.0],
          [{'Wine'}, {'Beef', 'Ice Cream'}, 0.75],
          [{'Ice Cream'}, {'Beef', 'Wine'}, 1.0],
          [{'Beef'}, {'Ice Cream', 'Wine'}, 1.0],
          [{'Ice Cream', 'Wine'}, {'Beef'}, 1.0],
          [{'Beef', 'Wine'}, {'Ice Cream'}, 1.0],
          [{'Beef', 'Ice Cream'}, {'Wine'}, 1.0],
          [{'Wine'}, {'Beef'}, 0.75],
          [{'Beef'}, {'Wine'}, 1.0]])
```

Part 2(b):

Fix the problem in the Part 2 code and re-run your program on the given dataset to obtain the same results as in Part 1(a). What did you change?

Answer:

I can sequentially selected the items in the both headertable and itemset to generate an ordered itemsets.

```
In [8]:  freqItemSet, rules = fpgrowth(dataset, minSupRatio=0.6, minConf=0.7)
         freqItemSet, rules
```

```
Out[8]: ([{'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],
           [{'Chicken'}, {'Corn'}, 1.0],
           [{'Corn'}, {'Chicken'}, 1.0],
           [{'Light Cream'}, {'Chicken', 'Corn'}, 1.0],
           [{'Chicken'}, {'Corn', 'Light Cream'}, 1.0],
           [{'Corn'}, {'Chicken', 'Light Cream'}, 1.0],
           [{'Chicken', 'Light Cream'}, {'Corn'}, 1.0],
           [{'Corn', 'Light Cream'}, {'Chicken'}, 1.0],
           [{'Chicken', 'Corn'}, {'Light Cream'}, 1.0],
           [{'Wine'}, {'Ice Cream'}, 0.75],
           [{'Ice Cream'}, {'Wine'}, 1.0],
           [{'Ice Cream'}, {'Beef'}, 1.0],
           [{'Beef'}, {'Ice Cream'}, 1.0],
           [{'Wine'}, {'Beef', 'Ice Cream'}, 0.75],
           [{'Ice Cream'}, {'Beef', 'Wine'}, 1.0],
           [{'Beef'}, {'Ice Cream', 'Wine'}, 1.0],
           [{'Ice Cream', 'Wine'}, {'Beef'}, 1.0],
           [{'Beef', 'Wine'}, {'Ice Cream'}, 1.0],
           [{'Beef', 'Ice Cream'}, {'Wine'}, 1.0],
           [{'Wine'}, {'Beef'}, 0.75],
           [{'Beef'}, {'Wine'}, 1.0]])
```

Part 2(c):

Explain in your own words how the anti-monotonicity property of itemsets is maintained to reduce the search space in generating frequent itemsets.

Answer:

For conditional pattern tree on each item, we only consider the items from different paths that can be summed up to the support count to recursively grow the new tree.

Part 2(d):

Explain in your own words how the pattern growth is achieved in generating frequent itemsets.

Answer:

Conditioned on one item, we increase the count of items on the paths leading to that conditional item. Then we choose the qualified items (count > support count) to grow a new FPtree and then we mine the frequent itemsets from each note on the tree recursively until there's no items left to mine a new frequent itemset.

Part 2(e):

Explain in your own words how many data scan is required for FPgrowth compared to Apriori.

Answer:

For Apriori, we need one scan to count its support from all transactions for every candidate frequent itemset, so there're 2^k scans needed.

For FPgrowth, we only need two scans: the first one to count item frequencies in all transactions and sort them into a headertable, the second one is to use each transaction to grow FP tree leveraging the headertable we generate from the first step. After that, we only use FPgrowth to mine the frequent itemsets.

Part 2(f):

Explain in your own words how time complexity and memory consumption is reduced compared to Apriori.

Answer:

Time complexity:
For Apriori, it's a generate-and-test approach. Despite of anti-monotone, the workload still increases exponentially on the stage number k of qualified candidates.
For FPgrowth, we partition the database into a FPtree by two scans, then we keep them running in parallel and therefore time complexity is solely depend on the item number and tree size.

Memroy consumption:
For Apriori, we need to load the whole dataset to scan all transactions sequentially and make comparisons. For FPgrowth, in the best case, all transactions share the same set of 1 path, we can just need to store a relative small FP tree in the memory, in the worst case, all transactions go by different unique items, we get a relative big FPtree, then we need a lot more of memory resources as we need to take all the pointers and counters into account.

Extra Marks

Please provide a different item ordering (i.e., not descending order of frequency) in constructing FP-Tree and draw the corresponding FP-Tree ofthe same datasetas above. Do you believe that you

have the most compressed (or compact) FP-Tree representation in this particular case?

Answer:

Could the most compact one as I combine the most common items beef, wine, ice cream at the top of the header table.

If you would like to upload an image, use the following command (but exchange the filename): (Double-click on this cell to see the command)

Please make sure only to include high-quality and readable images! We can not grant full marks if we can't understand what you did.

Report

Throught the assignment, I gain a better understanding of Apriori, FPtree, FPgrowth. And also practice debugging skills - by displaying the FPtree to narrow down the scope and inspecting the itemset to fix the code.