CSC4008 Assignment 5

@arthor: JIANG, Jingxin 117020119

0. Macros

In [1]:
```python
import pandas as pd
import numpy as np
```

1. Data Preprocessing

1.1 Raw Data

In [2]:
```python
raw_supermarket = pd.read_excel("AS5 supermarket.xlsx")
```

1.2 Procedure

1.2.1 Replace

In [3]:
```python
supermarket = raw_supermarket.replace({"?":0,"high\n":"high","low\n":"low"})
```

1.2.2 Add column names

In [4]:
```python
names = pd.read_excel("AS5 supermarket_attribute_name.xlsx",header=None)[1]
names = [col.strip() for col in names]
supermarket.columns = names
```

1.2.3 Drop columns

drop department columns:

In [5]:
```python
department_col = [col for col in supermarket.columns if col.startswith('department')]
supermarket = supermarket.drop(columns=department_col)
```

drop non-frequent columns:

In [6]:
```python
supermarket = supermarket.drop(columns=supermarket.columns[supermarket.sum()==0])
```

drop `high` / `low` class column:

In [7]:
```python
supermarket = supermarket.drop(columns=['total'])
```

1.3 Data Info

In [8]:
```python
supermarket.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4627 entries, 0 to 4626
Data columns (total 100 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   grocery misc        4627 non-null   int64
 1   baby needs          4627 non-null   int64
 2   bread and cake      4627 non-null   int64
 3   baking needs        4627 non-null   int64
 4   juice-sat-cord-ms   4627 non-null   int64
 5   tea                 4627 non-null   int64
 6   biscuits            4627 non-null   int64
 7   canned fish-meat    4627 non-null   int64
 8   canned fruit        4627 non-null   int64
 9   canned vegetables   4627 non-null   int64
 10  breakfast food      4627 non-null   int64
 11  cigs-tobacco pkts   4627 non-null   int64
 12  cigarette cartons   4627 non-null   int64
 13  cleaners-polishers  4627 non-null   int64
 14  coffee              4627 non-null   int64
 15  sauces-gravy-pkle   4627 non-null   int64
 16  confectionary       4627 non-null   int64
 17  puddings-deserts    4627 non-null   int64
 18  dishcloths-scour    4627 non-null   int64
 19  deod-disinfectant   4627 non-null   int64
 20  frozen foods        4627 non-null   int64
 21  razor blades        4627 non-null   int64
 22  fuels-garden aids   4627 non-null   int64
 23  spices              4627 non-null   int64
 24  jams-spreads        4627 non-null   int64
 25  insecticides        4627 non-null   int64
 26  pet foods           4627 non-null   int64
 27  laundry needs       4627 non-null   int64
 28  party snack foods   4627 non-null   int64
 29  tissues-paper prd   4627 non-null   int64
 30  wrapping            4627 non-null   int64
 31  dried vegetables    4627 non-null   int64
 32  pkt-canned soup     4627 non-null   int64
 33  soft drinks         4627 non-null   int64
 34  health food other   4627 non-null   int64
 35  beverages hot       4627 non-null   int64
 36  health&beauty misc  4627 non-null   int64
 37  deodorants-soap     4627 non-null   int64
 38  mens toiletries     4627 non-null   int64
 39  medicines           4627 non-null   int64
 40  haircare            4627 non-null   int64
 41  dental needs        4627 non-null   int64
 42  lotions-creams      4627 non-null   int64
 43  sanitary pads       4627 non-null   int64
 44  cough-cold-pain     4627 non-null   int64
 45  meat misc           4627 non-null   int64
 46  cheese              4627 non-null   int64
 47  chickens            4627 non-null   int64
 48  milk-cream          4627 non-null   int64
 49  cold-meats          4627 non-null   int64
 50  deli gourmet        4627 non-null   int64
 51  margarine           4627 non-null   int64
 52  salads              4627 non-null   int64
 53  small goods         4627 non-null   int64
 54  dairy foods         4627 non-null   int64
 55  fruit drinks        4627 non-null   int64
 56  delicatessen misc   4627 non-null   int64
 57  beef                4627 non-null   int64
 58  hogget              4627 non-null   int64
 59  lamb                4627 non-null   int64
 60  pet food            4627 non-null   int64
 61  pork                4627 non-null   int64
 62  poultry             4627 non-null   int64
 63  veal                4627 non-null   int64
 64  gourmet meat        4627 non-null   int64
 65  produce misc        4627 non-null   int64
 66  fruit               4627 non-null   int64
 67  plants              4627 non-null   int64
 68  potatoes            4627 non-null   int64
 69  vegetables          4627 non-null   int64
 70  variety misc        4627 non-null   int64
 71  brushware           4627 non-null   int64
 72  electrical          4627 non-null   int64
 73  haberdashery        4627 non-null   int64
 74  kitchen             4627 non-null   int64
 75  manchester          4627 non-null   int64
 76  pantyhose           4627 non-null   int64
 77  plasticware         4627 non-null   int64
 78  stationary          4627 non-null   int64
 79  prepared meals      4627 non-null   int64
 80  preserving needs    4627 non-null   int64
 81  condiments          4627 non-null   int64
 82  cooking oils        4627 non-null   int64
 83  bake off products   4627 non-null   int64
 84  small goods2        4627 non-null   int64
 85  offal               4627 non-null   int64
 86  mutton              4627 non-null   int64
 87  trim pork           4627 non-null   int64
 88  trim lamb           4627 non-null   int64
 89  imported cheese     4627 non-null   int64
 90  casks white wine    4627 non-null   int64
 91  casks red wine      4627 non-null   int64
 92  750ml white nz      4627 non-null   int64
 93  750ml red nz        4627 non-null   int64
 94  750ml white imp     4627 non-null   int64
 95  750ml red imp       4627 non-null   int64
 96  sparkling nz        4627 non-null   int64
 97  sparkling imp       4627 non-null   int64
 98  port and sherry     4627 non-null   int64
 99  non host support    4627 non-null   int64
dtypes: int64(100)
memory usage: 3.5 MB
```

2. Data Mining

2.1 Apriori Algorithm

2.1.1 Apriori class

In [9]:
```python
class Apriori:
    def __init__(self, min_sup = 0.2):
        self.min_sup = min_sup # min support

    '''
    *Generating frequent 1-itemsets*
    in: data<pandas.core.frame.DataFrame>
    out: C<list>[<tuple>], D<dict>
    '''
    def _frequent_singleton(self, data):
        sup_count = data.sum()
        sup_count = sup_count[sup_count>=self.min_sup_count]
        sup_count.index = [(idx,) for idx in sup_count.index]
        C = sup_count.index
        D = sup_count.to_dict()
        return C, D

    '''
    *Generating candidates*
    in: l<list>[<tuple>], k<int>
    out: <list>[<tuple>]
    '''
    def _apriori_gen(self, l, k): # l:frequent (k-1)-itemsets
        C = set()
        for i in range(len(l)-1):
            for j in range(i+1,len(l)):
                c = tuple(sorted(set(l[i]).union(set(l[j]))))
                if (k == 2) or (len(c) == k) and not (self._has_infrequent_subset(c, l, k-1)):
                    C.add(c)
        return list(C)

    '''
    *Pruning*
    in: c<tuple>, l<list>[<tuple>], k<int>
    out: <boolean>
    '''
    def _has_infrequent_subset(self, c, l, k): # c: candidate k-itemset, L:frequent (k-1)-itemsets
        S = self._ksubsets(set(c), k)
        for s in S:
            if tuple(sorted(s)) not in l:
                return True
        return False

    '''
    *Generating all subsets of c with k elements*
    in: c<set>, k<int>
    out: <set>
    '''
    def _ksubsets(self, c, k):
        import itertools
        return set(itertools.combinations(c, k))

    '''
    *Mining all frequent itemsets*
    in: data<pandas.core.frame.DataFrame>
    out: L<list>[<list>[<tuple>]]
    '''
    def mine(self, data):
        import time
        start_time = time.time()
        self.min_sup_count = int(self.min_sup*data.shape[0])+1
        C, D= self._frequent_singleton(data)
        k = 1
        while len(C) > 0:
            C = self._apriori_gen(C, k+1)
            for i in range(data.shape[0]):
                transaction = data.loc[i]
                s = set(transaction[transaction>0].index)
                for c in C:
                    if set(c).issubset(s):
                        D[c] = D.get(c,0)+1
            C = [c for c in C if D.get(c,0) >= self.min_sup_count]
            k += 1
        D = {key:val for key,val in D.items() if val >= self.min_sup_count}
        elapsed_time = time.time() - start_time
        print('Time elapsed:', elapsed_time)
        return D

    '''
    *Showing top 10 most frequent 2,3,4-itemsets*
    in: D<dict>
    '''
    def display(self, D):
        for i in range(2,5):
            print('Top 10 most frequent ', i, '-itemset:',sep='')
            L = [item for item in D.items() if len(item[0])==i]
            L = sorted(L, key = lambda item: float(item[1]), reverse = True)
            for j in range(10):
                print(j+1,': ', L[j], sep='')
```

2.1.2 Pattern mining

In [10]:
```python
apriori = Apriori(min_sup=0.2)
D = apriori.mine(supermarket)
```

```
Time elapsed: 32.38124108314514
```

2.1.3 Frequent patterns

In [11]:
```python
apriori.display(D)
```

```
Top 10 most frequent 2-itemset:
1: (('bread and cake', 'milk-cream'), 2337)
2: (('bread and cake', 'fruit'), 2325)
3: (('bread and cake', 'vegetables'), 2298)
4: (('fruit', 'vegetables'), 2207)
5: (('baking needs', 'bread and cake'), 2191)
6: (('bread and cake', 'frozen foods'), 2129)
7: (('biscuits', 'bread and cake'), 2083)
8: (('fruit', 'milk-cream'), 2038)
9: (('milk-cream', 'vegetables'), 2025)
10: (('baking needs', 'vegetables'), 1949)
Top 10 most frequent 3-itemset:
1: (('bread and cake', 'fruit', 'vegetables'), 1791)
2: (('bread and cake', 'fruit', 'milk-cream'), 1684)
3: (('bread and cake', 'milk-cream', 'vegetables'), 1658)
4: (('baking needs', 'bread and cake', 'vegetables'), 1586)
5: (('baking needs', 'bread and cake', 'milk-cream'), 1580)
6: (('fruit', 'milk-cream', 'vegetables'), 1571)
7: (('baking needs', 'bread and cake', 'fruit'), 1564)
8: (('bread and cake', 'frozen foods', 'vegetables'), 1548)
9: (('bread and cake', 'frozen foods', 'fruit'), 1548)
10: (('biscuits', 'bread and cake', 'fruit'), 1541)
Top 10 most frequent 4-itemset:
1: (('bread and cake', 'fruit', 'milk-cream', 'vegetables'), 1311)
2: (('baking needs', 'bread and cake', 'fruit', 'vegetables'), 1255)
3: (('bread and cake', 'frozen foods', 'fruit', 'vegetables'), 1242)
4: (('biscuits', 'bread and cake', 'fruit', 'vegetables'), 1216)
5: (('baking needs', 'bread and cake', 'milk-cream', 'vegetables'), 1169)
6: (('baking needs', 'bread and cake', 'fruit', 'milk-cream'), 1161)
7: (('biscuits', 'bread and cake', 'frozen foods', 'fruit'), 1143)
8: (('bread and cake', 'frozen foods', 'fruit', 'milk-cream'), 1138)
9: (('bread and cake', 'frozen foods', 'milk-cream', 'vegetables'), 1131)
10: (('biscuits', 'bread and cake', 'fruit', 'milk-cream'), 1122)
```

2.2 FP-growth Algorithm

2.2.1 Header class

In [12]:
```python
class Node:
    def __init__(self, item, sup):
        self.item = item
        self.sup = sup
        self.adjacent = None
        self.tail = None
```

2.2.2 Tree class

In [13]:
```python
class treeNode:
    def __init__(self, item, count=1):
        self.item = item
        self.count = count
        self.parent = None
        self.next = None
        self.child = set()

    def single_path(self):
        temp = self
        while len(temp.child)>0:
            if len(temp.child)>1:
                return False
            temp = tuple(temp.child)[0]
        return True

    def insert_tree(self, header, transaction, i, count = 1):
        if i >= len(transaction):
            return
        head = None
        for node in header:
            if node.item == transaction[i]:
                head = node
                break
        if head == None:
            return
        for c in self.child:
            if c.item == transaction[i]:
                c.count += count
                c.insert_tree(header,transaction, i+1, count)
                return
        c = self.__class__(transaction[i],count)
        c.parent = self
        self.child.add(c)
        if head.adjacent == None:
            head.tail = head.adjacent = c
        else:
            head.tail.next = c
            head.tail = head.tail.next
        c.insert_tree(header,transaction, i+1, count)
        return
```

2.2.3 FP-tree class

In [14]:
```python
class FP_tree:
    def __init__(self,header, tree):
        self.header = header
        self.tree = tree

    '''
    *Generating conditional pattern base*
    in: item<str>
    out: pattern_base<dict>
    '''
    def _cond_pattern_base(self, item):
        pattern_base = dict()
        for node in self.header:
            if node.item == item:
                break
        head = node.adjacent
        while True:
            temp = head
            l = list()
            while temp.parent.item != '':
                l.append(temp.parent.item)
                temp = temp.parent
            if len(l) > 0:
                l.reverse()
                pattern_base[tuple(l)] = head.count
            if head.next == None:
                break
            head = head.next
        return pattern_base

    '''
    *Generating conditional FP-tree*
    in: item<str>, min_sup_count<int>
    out: <FP-tree>
    '''
    def cond_FP_tree(self, item, min_sup_count):
        cpb = self._cond_pattern_base(item)
        root = treeNode('',0)
        header = dict()
        for itemset in cpb:
            for item in itemset:
                header[item] = header.get(item,0) + cpb[itemset]
        header = [Node(key,val) for key,val in header.items() if val >= min_sup_count]
        for itemset in cpb:
            root.insert_tree(header, itemset, 0, cpb[itemset])
        return FP_tree(header, root)
```

2.2.4 FP-growth class

In [15]:
```python
class FP_growth:
    def __init__(self, min_sup = 0.2):
        self.min_sup = min_sup

    '''
    *Generating frequent 1-itemsets*
    in: data<pandas.core.frame.DataFrame>
    out: header<list>[<Node>], D<dict>
    '''
    def _frequent_item(self, data):
        sup_count = data.sum()
        sup_count = sup_count[sup_count >= self.min_sup_count]
        sup_count = sup_count.sort_values(ascending=False)
        header = [Node(item,sup_count[item]) for item in sup_count.index]
        sup_count.index = [(idx,) for idx in sup_count.index]
        D = sup_count.to_dict()
        return header, D

    '''
    *Constructing tree*
    in: header<list>[<Node>], data<pandas.core.frame.DataFrame>
    out: root<treeNode>
    '''
    def _construct_tree(self, header, data):
        root = treeNode('',0)
        for i in range(data.shape[0]):
            transaction = data.loc[i]
            transaction = [node.item for node in header if transaction[node.item] > 0]
            root.insert_tree(header, transaction, 0)
        return root

    '''
    *Implementing FP-growth*
    in: fp_tree<FP-tree>, pattern<list>
    out: D<dict>
    '''
    def _FP_growth(self, fp_tree, pattern):
        import itertools
        D = dict()
        if fp_tree.tree.single_path():
            temp = fp_tree.tree
            l = list()
            d = dict()
            while len(temp.child) > 0:
                c = tuple(temp.child)[0]
                l.append(c.item)
                d[c.item] = c.count
                temp = c
            for i in range(0,len(l)):
                combination = list(itertools.combinations(l,i+1))
                for itemset in combination:
                    sup_count = d[itemset[-1]]
                    itemset = list(itemset)+ pattern
                    D[tuple(itemset)] = sup_count
        else:
            for node in fp_tree.header:
                cfpt = fp_tree.cond_FP_tree(node.item, self.min_sup_count)
                D[tuple([node.item]+pattern)] = node.sup
                if len(cfpt.tree.child) > 0:
                    D = {**D,**self._FP_growth(cfpt, [node.item]+pattern)}
        return D

    '''
    *Mining all frequent itemsets*
    in: data<pandas.core.frame.DataFrame>
    out: <dict>
    '''
    def mine(self,data):
        import time
        start_time = time.time()
        self.min_sup_count = int(self.min_sup*data.shape[0])+1
        header, self.D = self._frequent_item(data) # D may be unnecessary
        tree = self._construct_tree(header,data)
        fp_tree = FP_tree(header, tree)
        self.D = {**self.D,**self._FP_growth(fp_tree,[])}
        elapsed_time = time.time() - start_time
        print('Time elapsed:', elapsed_time)
        return self.D

    '''
    *Showing top 10 most frequent 2,3,4-itemsets*
    in: D<dict>
    '''
    def display(self,D):
        for i in range(2,5):
            print('Top 10 most frequent ', i, '-itemset:',sep='')
            L = [(tuple(sorted(item[0])),item[1]) for item in D.items() if len(item[0])==i]
            L = sorted(L, key = lambda item: float(item[1]), reverse = True)
            for j in range(10):
                print(j+1,': ', L[j], sep='')
```

2.2.5 Pattern mining

In [16]:
```python
fp_growth = FP_growth(min_sup=0.2)
D = fp_growth.mine(supermarket)
```

```
Time elapsed: 5.522256135940552
```

2.2.6 Frequent patterns

In [17]:
```python
fp_growth.display(D)
```

```
Top 10 most frequent 2-itemset:
1: (('bread and cake', 'milk-cream'), 2337)
2: (('bread and cake', 'fruit'), 2325)
3: (('bread and cake', 'vegetables'), 2298)
4: (('fruit', 'vegetables'), 2207)
5: (('baking needs', 'bread and cake'), 2191)
6: (('bread and cake', 'frozen foods'), 2129)
7: (('biscuits', 'bread and cake'), 2083)
8: (('fruit', 'milk-cream'), 2038)
9: (('milk-cream', 'vegetables'), 2025)
10: (('baking needs', 'vegetables'), 1949)
Top 10 most frequent 3-itemset:
1: (('bread and cake', 'fruit', 'vegetables'), 1791)
2: (('bread and cake', 'fruit', 'milk-cream'), 1684)
3: (('bread and cake', 'milk-cream', 'vegetables'), 1658)
4: (('baking needs', 'bread and cake', 'vegetables'), 1586)
5: (('baking needs', 'bread and cake', 'milk-cream'), 1580)
6: (('fruit', 'milk-cream', 'vegetables'), 1571)
7: (('baking needs', 'bread and cake', 'fruit'), 1564)
8: (('bread and cake', 'frozen foods', 'fruit'), 1548)
9: (('bread and cake', 'frozen foods', 'vegetables'), 1548)
10: (('biscuits', 'bread and cake', 'fruit'), 1541)
Top 10 most frequent 4-itemset:
1: (('bread and cake', 'fruit', 'milk-cream', 'vegetables'), 1311)
2: (('baking needs', 'bread and cake', 'fruit', 'vegetables'), 1255)
3: (('bread and cake', 'frozen foods', 'fruit', 'vegetables'), 1242)
4: (('biscuits', 'bread and cake', 'fruit', 'vegetables'), 1216)
5: (('baking needs', 'bread and cake', 'milk-cream', 'vegetables'), 1169)
6: (('baking needs', 'bread and cake', 'fruit', 'milk-cream'), 1161)
7: (('biscuits', 'bread and cake', 'frozen foods', 'fruit'), 1143)
8: (('bread and cake', 'frozen foods', 'fruit', 'milk-cream'), 1138)
9: (('bread and cake', 'frozen foods', 'milk-cream', 'vegetables'), 1131)
10: (('biscuits', 'bread and cake', 'fruit', 'milk-cream'), 1122)
```

3. Comparison

under `min_sup` =0.2, FP-growth algorithm (~6s) runs almost five times faster than Apriori algorithm (>30s).

Different from the multiple scans of data in Apriori, FP-growth only scans the data twice and boosts the speed.

The end