# *CSB352: Data Mining*

Instructor : [Dr. Chandra Prakash]

## LAB Assignment 7: Association Mining

# Assigning Date: 15-Feb-2021

## Due Date: 21-Feb-2021

## Student Name: Rohit Byas

## Roll No: 181210043

```
try:
    from google.colab import drive
    %tensorflow_version 2.x
    COLAB = True
    print("Assignment 7 - Association Mining")
    print("Note: using Google CoLab")
except:
    print("Assignment 7 - Association Mining")
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Assignment 7 - Association Mining
Note: using Google CoLab
```

```
print("Name: Rohit Byas")
print("Roll Number : 181210043")
```

```
Name: Rohit Byas
Roll Number : 181210043
```

```
from datetime import date

today = date.today()
print("Current Date:", today)
```

```
Current Date: 2021-02-22
```

```
from datetime import datetime

now = datetime.now()
dt_string = now.strftime("%H:%M:%S")
print("Current Time:", dt_string)
```

```
Current Time: 04:16:27
```

### 4.0.1 Task 0: Getting to Know Your Data

## Read Dataset [ L7_Groceries.csv ] from the link from LAB 1

```
import itertools
import numpy as np
import pandas as pd
import pprint
pp = pprint.PrettyPrinter(indent=4)
```

```
#Downloading the dataset:
!gdown --id "1AbvTYu_UDv_IKqmh0CNLfib47aqqBt2Z"
```

```
Downloading...
From: https://drive.google.com/uc?id=1AbvTYu_UDv_IKqmh0CNLfib47aqqBt2Z
To: /content/L7_Groceries.csv
100% 501k/501k [00:00<00:00, 66.4MB/s]
```

```
#Storing all the transactions from the dataset in a list
list_of_transactions = []

for line in open("L7_Groceries.csv"):
    transaction = line.split(",")
    transaction[len(transaction)-1] = transaction[len(transaction)-1].replace("\n", "")
    list_of_transactions.append(transaction)
```

```
#Printing the list of transactions from the dataset:
pp.pprint(list_of_transactions)
```

```
             'sugar',
             'shopping bags'],
        [    'frankfurter',
             'tropical fruit',
             'other vegetables',
             'whole milk',
             'frozen meals',
             'rolls/buns',
             'detergent',
             'napkins',
             'newspapers'],
        [    'sausage',
             'butter',
```

```
            'butter',
            'rolls/buns',
            'pickled vegetables',
            'soda',
            'fruit/vegetable juice',
            'waffles'],
        [   'tropical fruit',
            'other vegetables',
            'domestic eggs',
            'zwieback',
            'ketchup',
            'soda',
            'dishes'],
        [   'sausage',
            'chicken',
            'beef',
            'hamburger meat',
            'citrus fruit',
            'grapes',
            'root vegetables',
            'whole milk',
            'butter',
            'whipped/sour cream',
            'flour',
            'coffee',
            'red/blush wine',
            'salty snack',
            'chocolate',
            'hygiene articles',
            'napkins'],
        ['cooking chocolate'],
        [   'chicken',
            'citrus fruit',
            'other vegetables',
            'butter',
            'yogurt',
            'frozen dessert',
            'domestic eggs',
            'rolls/buns',
            'rum',
            'cling film/bags'],
        ['semi-finished bread', 'bottled water', 'soda', 'bottled beer'],
        [   'chicken',
            'tropical fruit',
            'other vegetables',
            'vinegar',
            'shopping bags']]
```

```
#Function for storing the frequency count of each item in item_counts dictionary

def return_item_freq(list_of_transactions):
    item_count = {}

    for transaction in list_of_transactions:
        for item in transaction:
            if frozenset([item]) in item_count:
```

```
            if frozenset([item]) in item_count:
                item_count[frozenset([item])] = item_count[frozenset([item])]+1
            else:
                item_count[frozenset([item])] = 1

    return item_count


#Storing the frequency count in item_count
item_count = return_item_freq(list_of_transactions)


print("Number of unique items:", len(item_count))
```

```
Number of unique items: 169
```

```
#Displaying the itemset with its corresponding frequency (support count):
item_count
```

```
 frozenset({'snack products'}): 30,
 frozenset({'flower soil/fertilizer'}): 19,
 frozenset({'specialty cheese'}): 84,
 frozenset({'finished products'}): 64,
 frozenset({'cocoa drinks'}): 22,
 frozenset({'dog food'}): 84,
 frozenset({'prosecco'}): 20,
 frozenset({'frozen fish'}): 115,
 frozenset({'make up remover'}): 8,
 frozenset({'cleaner'}): 50,
 frozenset({'female sanitary products'}): 60,
 frozenset({'dish cleaner'}): 103,
 frozenset({'cookware'}): 27,
 frozenset({'meat'}): 254,
 frozenset({'tea'}): 38,
 frozenset({'mustard'}): 118,
 frozenset({'house keeping products'}): 82,
 frozenset({'skin care'}): 35,
 frozenset({'potato products'}): 28,
 frozenset({'liquor'}): 109,
 frozenset({'pet care'}): 93,
 frozenset({'soups'}): 67,
 frozenset({'rum'}): 44,
 frozenset({'salad dressing'}): 8,
 frozenset({'sauces'}): 54,
 frozenset({'vinegar'}): 64,
 frozenset({'soap'}): 26,
 frozenset({'hair spray'}): 11,
 frozenset({'instant coffee'}): 73,
 frozenset({'roll products '}): 101,
 frozenset({'mayonnaise'}): 90,
 frozenset({'rubbing alcohol'}): 10,
 frozenset({'syrup'}): 32,
 frozenset({'liver loaf'}): 50,
 frozenset({'baby cosmetics'}): 6,
 frozenset({'organic products'}): 16,
 frozenset({'nut snack'}): 31,
 frozenset({'kitchen towels'}): 59,
```

```
frozenset({'frozen chicken'}): 6,
frozenset({'light bulbs'}): 41,
frozenset({'ketchup'}): 42,
frozenset({'jam'}): 53,
frozenset({'decalcifier'}): 15,
frozenset({'nuts/prunes'}): 33,
frozenset({'liqueur'}): 9,
frozenset({'organic sausage'}): 22,
frozenset({'cream'}): 13,
frozenset({'toilet cleaner'}): 7,
frozenset({'specialty vegetables'}): 17,
frozenset({'baby food'}): 1,
frozenset({'pudding powder'}): 23,
frozenset({'tidbits'}): 23,
frozenset({'whisky'}): 8,
frozenset({'frozen fruits'}): 12,
frozenset({'bags'}): 4,
frozenset({'cooking chocolate'}): 25,
frozenset({'sound storage medium'}): 1,
frozenset({'kitchen utensil'}): 4,
frozenset({'preservation products'}): 2}
```

# 4.1 TASK 1. Apriori Algorithm

```
def calculate_support_count(item_set, list_of_transactions):
    count = 0
    for transaction in list_of_transactions:
        if set(item_set).issubset(set(transaction)):
            count += 1
    return count


def get_association_rules(freq_item_sets, dataset, min_confidence):
    association_rules = []
    for item_set in freq_item_sets.keys():
        s = list(item_set)
        subsets = list(itertools.chain.from_iterable(itertools.combinations(s, r) for r in ra
        for sub_item_set in subsets:
            support = calculate_support_count(item_set, dataset)/len(dataset)
            rhs = set(item_set).difference(sub_item_set)
            if(len(rhs)==0):
                break
            lhs = sub_item_set
            support_of_lhs = calculate_support_count(frozenset(lhs), dataset)/len(dataset)

            confidence = (freq_item_sets[item_set]/len(dataset))/support_of_lhs
            if(confidence>min_confidence):
                support_of_rhs = calculate_support_count(frozenset(rhs), dataset)/len(dataset
                lift = confidence/support_of_rhs
                rule = {}
                rule['LHS'] = set(lhs)
```

```
                rule['RHS'] = rhs
                rule['Confidence'] = confidence
                rule['Lift'] = lift
                rule['Support'] = support
                association_rules.append(rule)
    return association_rules


#Defining my own apriori algorithm
def my_apriori(dataset, min_support, min_confidence):
    min_support = min_support*len(dataset)
    #Extracting all the items with support count more than min_support
    #And also sorting the dictionary in descending order of their frequency
    item_count = dict(sorted([elem for elem in return_item_freq(dataset).items() if elem[1]>=
                             key=lambda item: item[1],
                             reverse=True))
    keys = [frozenset(elem[0]) for elem in item_count.items()]
    next_keys = []

    coupling_count = 2
    while True:
        next_keys = []
        for subset in itertools.combinations(keys, coupling_count):
            next_keys.append(frozenset(np.array([list(elem) for elem in subset]).flatten()))
        coupling_count+=1
        next_item_count = {}

        for key in next_keys:
            next_item_count[key] = calculate_support_count(key, dataset)

        next_item_count = dict(sorted([elem for elem in next_item_count.items() if elem[1]>=m
                             key=lambda item: item[1],
                             reverse=True))

        if len(next_item_count)==0:
            break
        item_count = next_item_count
        keys = [elem[0] for elem in item_count.items()]
    freq_item_sets = item_count
    print("Frequent Item Sets:")
    pp.pprint(freq_item_sets)

    #Association rules:
    association_rules = get_association_rules(freq_item_sets, dataset, min_confidence)

    return association_rules


list_of_rules = my_apriori(list_of_transactions, min_support=0.03, min_confidence=0.3)
```

```
Frequent Item Sets:
{   frozenset({'whole milk', 'other vegetables'}): 736,
```

```
    frozenset({'whole milk', 'rolls/buns'}): 557,
    frozenset({'yogurt', 'whole milk'}): 551,
    frozenset({'whole milk', 'root vegetables'}): 481,
    frozenset({'root vegetables', 'other vegetables'}): 466,
    frozenset({'yogurt', 'other vegetables'}): 427,
    frozenset({'rolls/buns', 'other vegetables'}): 419,
    frozenset({'tropical fruit', 'whole milk'}): 416,
    frozenset({'whole milk', 'soda'}): 394,
    frozenset({'rolls/buns', 'soda'}): 377,
    frozenset({'tropical fruit', 'other vegetables'}): 353,
    frozenset({'bottled water', 'whole milk'}): 338,
    frozenset({'yogurt', 'rolls/buns'}): 338,
    frozenset({'whole milk', 'pastry'}): 327,
    frozenset({'other vegetables', 'soda'}): 322,
    frozenset({'whipped/sour cream', 'whole milk'}): 317,
    frozenset({'rolls/buns', 'sausage'}): 301,
    frozenset({'citrus fruit', 'whole milk'}): 300,
    frozenset({'whole milk', 'pip fruit'}): 296}
```

```
df = pd.DataFrame(list_of_rules)
df = df.sort_values(by=['Lift'], ascending=False)
df = df.reset_index().drop(columns=['index'])
df
```

| | LHS | RHS | Confidence | Lift | Support |
|---|---|---|---|---|---|
| 0 | {root vegetables} | {other vegetables} | 0.434701 | 2.246605 | 0.047382 |
| 1 | {sausage} | {rolls/buns} | 0.325758 | 1.771048 | 0.030605 |
| 2 | {tropical fruit} | {other vegetables} | 0.342054 | 1.767790 | 0.035892 |
| 3 | {whipped/sour cream} | {whole milk} | 0.449645 | 1.759754 | 0.032232 |
| 4 | {root vegetables} | {whole milk} | 0.448694 | 1.756031 | 0.048907 |
| 5 | {yogurt} | {other vegetables} | 0.311224 | 1.608457 | 0.043416 |
| 6 | {tropical fruit} | {whole milk} | 0.403101 | 1.577595 | 0.042298 |
| 7 | {yogurt} | {whole milk} | 0.401603 | 1.571735 | 0.056024 |
| 8 | {pip fruit} | {whole milk} | 0.397849 | 1.557043 | 0.030097 |
| 9 | {other vegetables} | {whole milk} | 0.386758 | 1.513634 | 0.074835 |
| 10 | {pastry} | {whole milk} | 0.373714 | 1.462587 | 0.033249 |
| 11 | {citrus fruit} | {whole milk} | 0.368550 | 1.442377 | 0.030503 |
| 12 | {bottled water} | {whole milk} | 0.310948 | 1.216940 | 0.034367 |
| 13 | {rolls/buns} | {whole milk} | 0.307905 | 1.205032 | 0.056634 |

## 4.2 TASK 2. Frequent Pattern Growth Algorithm

```
#Creating a trie data structure:
class Tree:
    def __init__(self, item, parent):
        self.item = item
        self.parent = parent
        if item is not None:
            self.frequency = 1
        else:
            self.frequency = 0
        self.children = []

    def add_child(self, itemset):
        #itemset must be in decreasing order of their frequency
        if len(itemset)>0:
            current_item = itemset[0]
            if current_item == self.item:
                self.frequency+=1
                if len(itemset)>1:
                    self.add_child(itemset[1:])
            else:
                #Search for child == current_item
                found = False
                for child in self.children:
                    if child.item == current_item:
                        child.add_child(itemset)
                        found = True
                        break
                #If no such child, create a new child
                if not found:
                    new_child = Tree(current_item, self)
                    self.children.append(new_child)
                    if len(itemset)>1:
                            new_child.add_child(itemset[1:])

    def print_all_children(self):
        print("{item:", self.item, ", freq:",self.frequency, "}")
        if len(self.children)>0:
            for child in self.children:
                child.print_all_children()

    #wrong implementation
    def get_all_paths(self, item):
        set_of_paths = []
        if(self.item==item):
            freq = self.frequency
            #recursively get its path by traversing through its parents:
            path = []
            node = self.parent
            while node.item is not None:
                path.append(node.item)
                node = node.parent
            if(len(path)>0):
```

```
            if(len(path)>0):
                path.sort(reverse=True)
                set_of_paths.append({'path': path, 'frequency': freq})
        else:
            if len(self.children)>0:
                for child in self.children:
                    set_of_paths += child.get_all_paths(item)
        return set_of_paths


def get_sorted_dataset(dataset, min_support):
    item_count = dict(sorted([elem for elem in return_item_freq(dataset).items() if elem[1]>m
                             key=lambda item: item[1],
                             reverse=True))
    new_sorted_dataset = []
    for transaction in dataset:
        new_transaction = sorted([item for item in transaction if frozenset({item}) in item_c
        if len(new_transaction)>0:
            new_sorted_dataset.append(new_transaction)
    return new_sorted_dataset


def my_FPG(dataset, min_support, min_confidence):
    min_support = min_support*len(dataset)
    new_sorted_dataset = get_sorted_dataset(dataset, min_support)
    count = 0
    root = Tree(None, None)
    for transaction in new_sorted_dataset:
        root.add_child(transaction)
    #root.print_all_children()
    item_count = dict(sorted([elem for elem in return_item_freq(dataset).items() if elem[1]>m
                             key=lambda item: item[1],
                             reverse=True))
    #print(item_count)
    list_of_all_paths = []
    items = [list(elem)[0] for elem in list(item_count.keys())]
    #pp.pprint(item_count)
    for item in items:
        paths = root.get_all_paths(item)
        if(len(paths)>0):
            list_of_all_paths.append({
                "item": item,
                "paths": paths
            })
    pp.pprint(list_of_all_paths)


my_FPG(list_of_transactions, 0.03, 0.3)
```

```
                                          'root vegetables',
                                          'pork',
                                          'napkins']},
                              {   'frequency': 1,
                                  'path': [   'root vegetables',
```

```
                 'fruit/vegetable juice',
                 'frozen vegetables',
                 'beef']},
{   'frequency': 1,
    'path': ['root vegetables', 'chicken', 'UHT-milk']},
{   'frequency': 1,
    'path': ['salty snack', 'root vegetables']},
{'frequency': 1, 'path': ['frozen vegetables']},
{'frequency': 1, 'path': ['whipped/sour cream']},
{   'frequency': 1,
    'path': ['whipped/sour cream', 'waffles']},
{'frequency': 1, 'path': ['curd']},
{'frequency': 1, 'path': ['shopping bags', 'pip fruit']},
{   'frequency': 1,
    'path': [   'waffles',
                'shopping bags',

                'pastry',
                'long life bakery product',
                'domestic eggs']},
{   'frequency': 1,
    'path': [   'shopping bags',
                'chocolate',
                'bottled beer']},
{'frequency': 2, 'path': ['shopping bags', 'canned beer']},
{   'frequency': 1,
    'path': [   'shopping bags',
                'napkins',
                'hygiene articles',
                'canned beer']},
{'frequency': 1, 'path': ['shopping bags', 'brown bread']},
{   'frequency': 1,
    'path': [   'shopping bags',
                'onions',
                'napkins',
                'beef']},
{   'frequency': 1,
    'path': ['shopping bags', 'salty snack', 'chocolate']},
{   'frequency': 1,
    'path': [   'waffles',
                'shopping bags',
                'long life bakery product',
                'fruit/vegetable juice']},
{'frequency': 2, 'path': ['bottled beer']},
{'frequency': 3, 'path': ['dessert']},
{'frequency': 1, 'path': ['pip fruit', 'frankfurter']},
{   'frequency': 1,
    'path': ['pork', 'butter', 'brown bread']},
{'frequency': 1, 'path': ['white bread', 'cream cheese ']},
{'frequency': 2, 'path': ['salty snack']},
{'frequency': 1, 'path': ['domestic eggs']},
{'frequency': 2, 'path': ['long life bakery product']},
{'frequency': 2, 'path': ['waffles']},
{'frequency': 1, 'path': ['waffles', 'berries']},
{'frequency': 1, 'path': ['cream cheese ']}]}]
```

## 4.3 TASK 3: Compare the results of your funcitons for both algorithm with the inbuild/pre-build packages respectively.

```
from mlxtend.frequent_patterns import apriori,association_rules
from mlxtend.preprocessing import TransactionEncoder


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


frequent_itemsets = apriori(df, min_support=0.03, use_colnames=True)

rules = association_rules(frequent_itemsets, metric ="confidence", min_threshold = 0.3)
rules = rules.sort_values('lift', ascending =False)
rules
```

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | le |
|---|---|---|---|---|---|---|---|---|
| **2** | (root vegetables) | (other vegetables) | 0.108998 | 0.193493 | 0.047382 | 0.434701 | 2.246605 | 0. |
| **8** | (sausage) | (rolls/buns) | 0.093950 | 0.183935 | 0.030605 | 0.325758 | 1.771048 | 0. |
| **3** | (tropical fruit) | (other vegetables) | 0.104931 | 0.193493 | 0.035892 | 0.342054 | 1.767790 | 0. |
| **12** | (whipped/sour cream) | (whole milk) | 0.071683 | 0.255516 | 0.032232 | 0.449645 | 1.759754 | 0. |
| **10** | (root vegetables) | (whole milk) | 0.108998 | 0.255516 | 0.048907 | 0.448694 | 1.756031 | 0. |
| **5** | (yogurt) | (other vegetables) | 0.139502 | 0.193493 | 0.043416 | 0.311224 | 1.608457 | 0. |
| **11** | (tropical fruit) | (whole milk) | 0.104931 | 0.255516 | 0.042298 | 0.403101 | 1.577595 | 0. |
| **13** | (yogurt) | (whole milk) | 0.139502 | 0.255516 | 0.056024 | 0.401603 | 1.571735 | 0. |
| **7** | (pip fruit) | (whole milk) | 0.075648 | 0.255516 | 0.030097 | 0.397849 | 1.557043 | 0. |
| **4** | (other vegetables) | (whole milk) | 0.193493 | 0.255516 | 0.074835 | 0.386758 | 1.513634 | 0. |

## Your Learning and observation

### Observation

we came to know that people buy milk products very frequently and together i.e.

1. milk, yogurt, cheese together
2. onion with vegetables.
3. soda with margarine etc.

**Learning**

In this assignment, I have learnt to create functions to implement these algorithms practically.

After creating my functions, I also compared the results with the in-build python packages to understand better