ShriDattaMadhira_Apriori_Assignment_4

October 10, 2021

1 APRIORI ALGORITHM

[2]:
```
# (c) 2016 Everaldo Aguiar & Reid Johnson
#
# Modified from:
# Marcel Caraciolo (https://gist.github.com/marcelcaraciolo/1423287)
#
# Functions to compute and extract association rules from a given frequent␣
↪itemset
# generated by the Apriori algorithm.
#
# The Apriori algorithm is defined by Agrawal and Srikant in:
# Fast algorithms for mining association rules
# Proc. 20th int. conf. very large data bases, VLDB. Vol. 1215. 1994
import csv
import numpy as np

def load_dataset(filename):
    '''Loads an example of market basket transactions from a provided csv file.

    Returns: A list (database) of lists (transactions). Each element of a␣
↪transaction is
    an item.
    '''

    with open(filename,'r') as dest_f:
        data_iter = csv.reader(dest_f, delimiter = ',', quotechar = '"')
        data = [data for data in data_iter]
        data_array = np.asarray(data)

    return data_array

def apriori(dataset, min_support=0.5, verbose=False):
    """Implements the Apriori algorithm.

    The Apriori algorithm will iteratively generate new candidate
    k-itemsets using the frequent (k-1)-itemsets found in the previous
    iteration.

    Parameters
    ----------
    dataset : list
        The dataset (a list of transactions) from which to generate
        candidate itemsets.

    min_support : float
        The minimum support threshold. Defaults to 0.5.

    Returns
    -------
    F : list
        The list of frequent itemsets.

    support_data : dict
        The support data for all candidate itemsets.

    References
    ----------
    .. [1] R. Agrawal, R. Srikant, "Fast Algorithms for Mining Association
           Rules", 1994.

    """
    C1 = create_candidates(dataset)
    D = list(map(set, dataset))
    F1, support_data = support_prune(D, C1, min_support, verbose=False) # prune␣
↪candidate 1-itemsets
    F = [F1] # list of frequent itemsets; initialized to frequent 1-itemsets
    k = 2 # the itemset cardinality
    while (len(F[k - 2]) > 0):
        Ck = apriori_gen(F[k-2], k) # generate candidate itemsets
        Fk, supK = support_prune(D, Ck, min_support) # prune candidate itemsets
        support_data.update(supK) # update the support counts to reflect pruning
        F.append(Fk) # add the pruned candidate itemsets to the list of␣
↪frequent itemsets
        k += 1

    if verbose:
        # Print a list of all the frequent itemsets.
        for kset in F:
            for item in kset:
                print("" \
                    + "{" \
                    + "".join(str(i) + ", " for i in iter(item)).rstrip(', ') \
                    + "}" \
                    + ":  sup = " + str(round(support_data[item], 3)))

    return F, support_data

def create_candidates(dataset, verbose=False):
    """Creates a list of candidate 1-itemsets from a list of transactions.

    Parameters
    ----------
    dataset : list
        The dataset (a list of transactions) from which to generate candidate
        itemsets.

    Returns
    -------
    The list of candidate itemsets (c1) passed as a frozenset (a set that is
    immutable and hashable).
    """
    c1 = [] # list of all items in the database of transactions
    for transaction in dataset:
        for item in transaction:
            if not [item] in c1:
                c1.append([item])
    c1.sort()

    if verbose:
        # Print a list of all the candidate items.
        print("" \
            + "{" \
            + "".join(str(i[0]) + ", " for i in iter(c1)).rstrip(', ') \
            + "}")

    # Map c1 to a frozenset because it will be the key of a dictionary.
    return list(map(frozenset, c1))

def support_prune(dataset, candidates, min_support, verbose=False):
    """Returns all candidate itemsets that meet a minimum support threshold.

    By the apriori principle, if an itemset is frequent, then all of its
    subsets must also be frequent. As a result, we can perform support-based
    pruning to systematically control the exponential growth of candidate
    itemsets. Thus, itemsets that do not meet the minimum support level are
    pruned from the input list of itemsets (dataset).

    Parameters
    ----------
    dataset : list
```

```
        The dataset (a list of transactions) from which to generate candidate
        itemsets.

    candidates : frozenset
        The list of candidate itemsets.

    min_support : float
        The minimum support threshold.

    Returns
    -------
    retlist : list
        The list of frequent itemsets.

    support_data : dict
        The support data for all candidate itemsets.
    """
    sscnt = {} # set for support counts
    for tid in dataset:
        for can in candidates:
            if can.issubset(tid):
                sscnt.setdefault(can, 0)
                sscnt[can] += 1

    num_items = float(len(dataset)) # total number of transactions in the␣
↪dataset
    retlist = [] # array for unpruned itemsets
    support_data = {} # set for support data for corresponding itemsets
    for key in sscnt:
        # Calculate the support of itemset key.
        support = sscnt[key] / num_items
        if support >= min_support:
            retlist.insert(0, key)
        support_data[key] = support

    # Print a list of the pruned itemsets.
    if verbose:
        for kset in retlist:
            for item in kset:
                print("{" + str(item) + "}")
        print("")
        for key in sscnt:
            print("" \
                + "{" \
                + "".join([str(i) + ", " for i in iter(key)]).rstrip(', ') \
                + "}" \
                + ":  sup = " + str(support_data[key]))
    return retlist, support_data

def apriori_gen(freq_sets, k):
    """Generates candidate itemsets (via the F_k-1 x F_k-1 method).

    This operation generates new candidate k-itemsets based on the frequent
    (k-1)-itemsets found in the previous iteration. The candidate generation
    procedure merges a pair of frequent (k-1)-itemsets only if their first k-2
    items are identical.

    Parameters
    ----------
    freq_sets : list
        The list of frequent (k-1)-itemsets.

    k : integer
        The cardinality of the current itemsets being evaluated.

    Returns
    -------
    retlist : list
        The list of merged frequent itemsets.
    """
    retList = [] # list of merged frequent itemsets
    lenLk = len(freq_sets) # number of frequent itemsets
    for i in range(lenLk):
        for j in range(i+1, lenLk):
            a=list(freq_sets[i])
            b=list(freq_sets[j])
            a.sort()
            b.sort()
            F1 = a[:k-2] # first k-2 items of freq_sets[i]
            F2 = b[:k-2] # first k-2 items of freq_sets[j]

            if F1 == F2: # if the first k-2 items are identical
                # Merge the frequent itemsets.
                retList.append(freq_sets[i] | freq_sets[j])

    return retList

def rules_from_conseq(freq_set, H, support_data, rules, min_confidence=0.5,␣
 ↪verbose=False):
    """Generates a set of candidate rules.

    Parameters
    ----------
    freq_set : frozenset
        The complete list of frequent itemsets.

    H : list
        A list of frequent itemsets (of a particular length).

    support_data : dict
        The support data for all candidate itemsets.

    rules : list
        A potentially incomplete set of candidate rules above the minimum
        confidence threshold.

    min_confidence : float
        The minimum confidence threshold. Defaults to 0.5.
    """
    m = len(H[0])
    if m == 1:
        Hmp1 = calc_confidence(freq_set, H, support_data, rules,␣
↪min_confidence, verbose)
    if (len(freq_set) > (m+1)):
        Hmp1 = apriori_gen(H, m+1) # generate candidate itemsets
        Hmp1 = calc_confidence(freq_set, Hmp1, support_data, rules,␣
↪min_confidence, verbose)
        if len(Hmp1) > 1:
            # If there are candidate rules above the minimum confidence
            # threshold, recurse on the list of these candidate rules.
            rules_from_conseq(freq_set, Hmp1, support_data, rules,␣
↪min_confidence, verbose)

def calc_confidence(freq_set, H, support_data, rules, min_confidence=0.5,␣
↪verbose=False):
    """Evaluates the generated rules.

    One measurement for quantifying the goodness of association rules is
    confidence. The confidence for a rule 'P implies H' (P -> H) is defined as
    the support for P and H divided by the support for P
    (support (P|H) / support(P)), where the | symbol denotes the set union
    (thus P|H means all the items in set P or in set H).

    To calculate the confidence, we iterate through the frequent itemsets and
    associated support data. For each frequent itemset, we divide the support
    of the itemset by the support of the antecedent (left-hand-side of the
    rule).

    Parameters
    ----------
    freq_set : frozenset
        The complete list of frequent itemsets.

    H : list
        A list of frequent itemsets (of a particular length).

    min_support : float
        The minimum support threshold.

    rules : list
        A potentially incomplete set of candidate rules above the minimum
        confidence threshold.

    min_confidence : float
        The minimum confidence threshold. Defaults to 0.5.

    Returns
    -------
    pruned_H : list
        The list of candidate rules above the minimum confidence threshold.
    """
    pruned_H = [] # list of candidate rules above the minimum confidence␣
↪threshold
    for conseq in H: # iterate over the frequent itemsets
        conf = support_data[freq_set] / support_data[freq_set - conseq]
        if conf >= min_confidence:
            rules.append((freq_set - conseq, conseq, conf,␣
↪support_data[freq_set]))
            pruned_H.append(conseq)

            if verbose:
                print("" \
                    + "{" \
                    + "".join([str(i) + ", " for i in iter(freq_set-conseq)]).
↪rstrip(', ') \
                    + "}" \
                    + " ---> " \
                    + "{" \
                    + "".join([str(i) + ", " for i in iter(conseq)]).rstrip(',␣
↪') \
                    + "}" \
                    + ":  conf = " + str(round(conf, 3)) \
                    + ", sup = " + str(round(support_data[freq_set], 3)))

    return pruned_H

def generate_rules(F, support_data, min_confidence=0.5, verbose=True):
    """Generates a set of candidate rules from a list of frequent itemsets.

    For each frequent itemset, we calculate the confidence of using a
    particular item as the rule consequent (right-hand-side of the rule). By
    testing and merging the remaining rules, we recursively create a list of
    pruned rules.

    Parameters
    ----------
    F : list
        A list of frequent itemsets.

    support_data : dict
        The corresponding support data for the frequent itemsets (L).

    min_confidence : float
        The minimum confidence threshold. Defaults to 0.5.

    Returns
    -------
    rules : list
        The list of candidate rules above the minimum confidence threshold.
    """
    rules = []
    for i in range(1, len(F)):
        for freq_set in F[i]:
            H1 = [frozenset([itemset]) for itemset in freq_set]
            if (i > 1):
                rules_from_conseq(freq_set, H1, support_data, rules,␣
↪min_confidence, verbose)
            else:
                calc_confidence(freq_set, H1, support_data, rules,␣
↪min_confidence, verbose)

    return rules
```

[3]:
```
dataset = load_dataset('grocery.csv')
D = list(map(set, dataset))
```

```
/Users/shridatta/Downloads/ENTER/envs/cenv/lib/python3.9/site-
packages/numpy/core/_asarray.py:83: VisibleDeprecationWarning: Creating an
ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-
tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant
to do this, you must specify 'dtype=object' when creating the ndarray
  return array(a, dtype, copy=False, order=order)
```

[4]:
```
type(dataset)
```

```
[4]: numpy.ndarray
```

```
[5]: dataset.shape
```

```
[5]: (9835,)
```

```
[6]: dataset[:5]
```

```
[6]: array([list(['citrus fruit', 'semi-finished bread', 'margarine', 'ready
soups']),
       list(['tropical fruit', 'yogurt', 'coffee']), list(['whole milk']),
       list(['pip fruit', 'yogurt', 'cream cheese ', 'meat spreads']),
       list(['other vegetables', 'whole milk', 'condensed milk', 'long life
bakery product'])],
      dtype=object)
```

2 Complete the assignment below by making use of the provided funtions.

```
[7]: c1 = create_candidates(dataset, verbose=True) # 1-itemsets
```

```
{Instant food products, UHT-milk, abrasive cleaner, artif. sweetener, baby
cosmetics, baby food, bags, baking powder, bathroom cleaner, beef, berries,
beverages, bottled beer, bottled water, brandy, brown bread, butter, butter
milk, cake bar, candles, candy, canned beer, canned fish, canned fruit, canned
vegetables, cat food, cereals, chewing gum, chicken, chocolate, chocolate
marshmallow, citrus fruit, cleaner, cling film/bags, cocoa drinks, coffee,
condensed milk, cooking chocolate, cookware, cream, cream cheese , curd, curd
cheese, decalcifier, dental care, dessert, detergent, dish cleaner, dishes, dog
food, domestic eggs, female sanitary products, finished products, fish, flour,
flower (seeds), flower soil/fertilizer, frankfurter, frozen chicken, frozen
dessert, frozen fish, frozen fruits, frozen meals, frozen potato products,
frozen vegetables, fruit/vegetable juice, grapes, hair spray, ham, hamburger
meat, hard cheese, herbs, honey, house keeping products, hygiene articles, ice
cream, instant coffee, jam, ketchup, kitchen towels, kitchen utensil, light
bulbs, liqueur, liquor, liquor (appetizer), liver loaf, long life bakery
product, make up remover, male cosmetics, margarine, mayonnaise, meat, meat
spreads, misc. beverages, mustard, napkins, newspapers, nut snack, nuts/prunes,
oil, onions, organic products, organic sausage, other vegetables, packaged
fruit/vegetables, pasta, pastry, pet care, photo/film, pickled vegetables, pip
fruit, popcorn, pork, pot plants, potato products, preservation products,
processed cheese, prosecco, pudding powder, ready soups, red/blush wine, rice,
roll products , rolls/buns, root vegetables, rubbing alcohol, rum, salad
dressing, salt, salty snack, sauces, sausage, seasonal products, semi-finished
bread, shopping bags, skin care, sliced cheese, snack products, soap, soda, soft
cheese, softener, sound storage medium, soups, sparkling wine, specialty bar,
specialty cheese, specialty chocolate, specialty fat, specialty vegetables,
spices, spread cheese, sugar, sweet spreads, syrup, tea, tidbits, toilet
cleaner, tropical fruit, turkey, vinegar, waffles, whipped/sour cream, whisky,
white bread, white wine, whole milk, yogurt, zwieback}
```

[8]:
```
f1, data = support_prune(D, c1, 0.05, verbose=True)
```

```
{domestic eggs}
{whipped/sour cream}
{pork}
{napkins}
{shopping bags}
{brown bread}
{sausage}
{canned beer}
{root vegetables}
{pastry}
{newspapers}
{fruit/vegetable juice}
{soda}
{frankfurter}
{beef}
{curd}
{bottled water}
{bottled beer}
{rolls/buns}
{butter}
{other vegetables}
{pip fruit}
{whole milk}
{yogurt}
{tropical fruit}
{coffee}
{margarine}
{citrus fruit}

{citrus fruit}:  sup = 0.08276563294356888
{margarine}:  sup = 0.05856634468734113
{ready soups}:  sup = 0.0018301982714794102
{semi-finished bread}:  sup = 0.017691916624300967
{coffee}:  sup = 0.05805795627859685
{tropical fruit}:  sup = 0.10493136756481952
{yogurt}:  sup = 0.13950177935943062
{whole milk}:  sup = 0.25551601423487547
{cream cheese}:  sup = 0.03965429588205389
{meat spreads}:  sup = 0.004270462633451958
{pip fruit}:  sup = 0.07564819522114896
{condensed milk}:  sup = 0.010269445856634469
{long life bakery product}:  sup = 0.037417386883579054
```

```
{other vegetables}:  sup = 0.1934926283680732
{abrasive cleaner}:  sup = 0.0035587188612099642
{butter}:  sup = 0.05541433655312659
{rice}:  sup = 0.007625826131164209
{rolls/buns}:  sup = 0.18393492628368074
{UHT-milk}:  sup = 0.03345195729537367
{bottled beer}:  sup = 0.08052872394509406
{liquor (appetizer)}:  sup = 0.007930859176410779
{pot plants}:  sup = 0.01728520589730554
{cereals}:  sup = 0.0056939501779359435
{bottled water}:  sup = 0.11052364006100661
{chocolate}:  sup = 0.04961870869344179
{white bread}:  sup = 0.042094560244026434
{curd}:  sup = 0.05327910523640061
{dishes}:  sup = 0.01759023894255211
{flour}:  sup = 0.017386883579054397
{beef}:  sup = 0.05246568378240976
{frankfurter}:  sup = 0.058973055414336555
{soda}:  sup = 0.17437722419928825
{chicken}:  sup = 0.04290798169801729
{fruit/vegetable juice}:  sup = 0.0722928317234367
{newspapers}:  sup = 0.07981698017285206
{sugar}:  sup = 0.03385866802236909
{packaged fruit/vegetables}:  sup = 0.013014743263853584
{specialty bar}:  sup = 0.027351296390442297
{butter milk}:  sup = 0.027961362480935434
{pastry}:  sup = 0.08896797153024912
{detergent}:  sup = 0.019217081850533807
{processed cheese}:  sup = 0.016573462125063547
{bathroom cleaner}:  sup = 0.0027452974072191155
{candy}:  sup = 0.0298932384341637
{frozen dessert}:  sup = 0.010777834265378749
{root vegetables}:  sup = 0.10899847483477376
{salty snack}:  sup = 0.03782409761057448
{sweet spreads}:  sup = 0.009049313675648195
{waffles}:  sup = 0.038434163701067614
{canned beer}:  sup = 0.07768174885612608
{sausage}:  sup = 0.09395017793594305
{brown bread}:  sup = 0.06487036095577021
{shopping bags}:  sup = 0.09852567361464158
{beverages}:  sup = 0.026029486527707167
{hamburger meat}:  sup = 0.033248601931875954
{hygiene articles}:  sup = 0.03294356888662939
{napkins}:  sup = 0.05236400610066091
{spices}:  sup = 0.005185561769191663
{artif. sweetener}:  sup = 0.003253685815963396
{berries}:  sup = 0.033248601931875954
{pork}:  sup = 0.05765124555160142
```

```
{whipped/sour cream}:  sup = 0.07168276563294357
{grapes}:  sup = 0.022369089984748347
{dessert}:  sup = 0.03711235383833249
{zwieback}:  sup = 0.006914082358922217
{domestic eggs}:  sup = 0.06344687341128623
{spread cheese}:  sup = 0.011184544992374174
{misc. beverages}:  sup = 0.02836807320793086
{hard cheese}:  sup = 0.024504321301474327
{cat food}:  sup = 0.023284189120488054
{ham}:  sup = 0.026029486527707167
{baking powder}:  sup = 0.017691916624300967
{turkey}:  sup = 0.00813421453990849
{pickled vegetables}:  sup = 0.017895271987798677
{chewing gum}:  sup = 0.021047280122013217
{chocolate marshmallow}:  sup = 0.009049313675648195
{oil}:  sup = 0.02806304016268429
{ice cream}:  sup = 0.025012709710218607
{canned fish}:  sup = 0.015048296898830707
{frozen vegetables}:  sup = 0.04809354346720895
{seasonal products}:  sup = 0.014234875444839857
{curd cheese}:  sup = 0.005083884087442806
{red/blush wine}:  sup = 0.019217081850533807
{frozen potato products}:  sup = 0.008439247585155059
{candles}:  sup = 0.008947635993899338
{flower (seeds)}:  sup = 0.010371123538383325
{specialty chocolate}:  sup = 0.03040162684290798
{specialty fat}:  sup = 0.0036603965429588205
{sparkling wine}:  sup = 0.005592272496187087
{salt}:  sup = 0.010777834265378749
{frozen meals}:  sup = 0.02836807320793086
{canned vegetables}:  sup = 0.010777834265378749
{onions}:  sup = 0.031011692933401117
{herbs}:  sup = 0.01626842907981698
{white wine}:  sup = 0.019013726487036097
{brandy}:  sup = 0.004168784951703101
{photo/film}:  sup = 0.009252669039145907
{sliced cheese}:  sup = 0.024504321301474327
{pasta}:  sup = 0.015048296898830707
{softener}:  sup = 0.005490594814438231
{cling film/bags}:  sup = 0.011387900355871887
{fish}:  sup = 0.0029486527707168276
{male cosmetics}:  sup = 0.004575495678698526
{canned fruit}:  sup = 0.003253685815963396
{Instant food products}:  sup = 0.008032536858159633
{soft cheese}:  sup = 0.01708185053380783
{honey}:  sup = 0.001525165226232842
{dental care}:  sup = 0.005795627859684799
{popcorn}:  sup = 0.007219115404168785
```

```
{cake bar}:  sup = 0.013218098627351297
{snack products}:  sup = 0.003050330452465684
{flower soil/fertilizer}:  sup = 0.0019318759532282665
{specialty cheese}:  sup = 0.008540925266903915
{finished products}:  sup = 0.006507371631926792
{cocoa drinks}:  sup = 0.0022369089984748346
{dog food}:  sup = 0.008540925266903915
{prosecco}:  sup = 0.0020335536349771225
{frozen fish}:  sup = 0.011692933401118455
{make up remover}:  sup = 0.000813421453990849
{cleaner}:  sup = 0.005083884087442806
{female sanitary products}:  sup = 0.006100660904931368
{cookware}:  sup = 0.0027452974072191155
{dish cleaner}:  sup = 0.01047280122013218
{meat}:  sup = 0.025826131164209457
{tea}:  sup = 0.003863751906456533
{mustard}:  sup = 0.011997966446365024
{house keeping products}:  sup = 0.008337569903406202
{skin care}:  sup = 0.0035587188612099642
{potato products}:  sup = 0.0028469750889679717
{liquor}:  sup = 0.011082867310625319
{pet care}:  sup = 0.00945602440264362
{soups}:  sup = 0.00681240467717336
{rum}:  sup = 0.004473817996949669
{salad dressing}:  sup = 0.000813421453990849
{sauces}:  sup = 0.005490594814438231
{vinegar}:  sup = 0.006507371631926792
{soap}:  sup = 0.0026436197254702592
{hair spray}:  sup = 0.0011184544992374173
{instant coffee}:  sup = 0.007422470767666497
{roll products}:  sup = 0.010269445856634469
{mayonnaise}:  sup = 0.009150991357397052
{rubbing alcohol}:  sup = 0.0010167768174885613
{syrup}:  sup = 0.003253685815963396
{liver loaf}:  sup = 0.005083884087442806
{baby cosmetics}:  sup = 0.0006100660904931368
{organic products}:  sup = 0.001626842907981698
{nut snack}:  sup = 0.00315200813421454
{kitchen towels}:  sup = 0.005998983223182512
{frozen chicken}:  sup = 0.0006100660904931368
{light bulbs}:  sup = 0.004168784951703101
{ketchup}:  sup = 0.004270462633451958
{jam}:  sup = 0.005388917132689374
{decalcifier}:  sup = 0.001525165226232842
{nuts/prunes}:  sup = 0.003355363497712252
{liqueur}:  sup = 0.0009150991357397051
{organic sausage}:  sup = 0.0022369089984748346
{cream}:  sup = 0.0013218098627351296
```

```
{toilet cleaner}:  sup = 0.0007117437722419929
{specialty vegetables}:  sup = 0.0017285205897305542
{baby food}:  sup = 0.00010167768174885612
{pudding powder}:  sup = 0.002338586680223691
{tidbits}:  sup = 0.002338586680223691
{whisky}:  sup = 0.0008134214539908949
{frozen fruits}:  sup = 0.0012201321809862736
{bags}:  sup = 0.0004067107269954245
{cooking chocolate}:  sup = 0.002541942043721403
{sound storage medium}:  sup = 0.00010167768174885612
{kitchen utensil}:  sup = 0.0004067107269954245
{preservation products}:  sup = 0.00020335536349771224
```

[9]:
```
f, a_data = apriori(dataset, 0.02, verbose=True)
```

```
{meat}:  sup = 0.026
{sliced cheese}:  sup = 0.025
{onions}:  sup = 0.031
{frozen meals}:  sup = 0.028
{specialty chocolate}:  sup = 0.03
{frozen vegetables}:  sup = 0.048
{ice cream}:  sup = 0.025
{oil}:  sup = 0.028
{chewing gum}:  sup = 0.021
{ham}:  sup = 0.026
{cat food}:  sup = 0.023
{hard cheese}:  sup = 0.025
{misc. beverages}:  sup = 0.028
{domestic eggs}:  sup = 0.063
{dessert}:  sup = 0.037
{grapes}:  sup = 0.022
{whipped/sour cream}:  sup = 0.072
{pork}:  sup = 0.058
{berries}:  sup = 0.033
{napkins}:  sup = 0.052
{hygiene articles}:  sup = 0.033
{hamburger meat}:  sup = 0.033
{beverages}:  sup = 0.026
{shopping bags}:  sup = 0.099
{brown bread}:  sup = 0.065
{sausage}:  sup = 0.094
{canned beer}:  sup = 0.078
{waffles}:  sup = 0.038
{salty snack}:  sup = 0.038
{root vegetables}:  sup = 0.109
{candy}:  sup = 0.03
{pastry}:  sup = 0.089
{butter milk}:  sup = 0.028
```

```
{specialty bar}:  sup = 0.027
{sugar}:  sup = 0.034
{newspapers}:  sup = 0.08
{fruit/vegetable juice}:  sup = 0.072
{chicken}:  sup = 0.043
{soda}:  sup = 0.174
{frankfurter}:  sup = 0.059
{beef}:  sup = 0.052
{curd}:  sup = 0.053
{white bread}:  sup = 0.042
{chocolate}:  sup = 0.05
{bottled water}:  sup = 0.111
{bottled beer}:  sup = 0.081
{UHT-milk}:  sup = 0.033
{rolls/buns}:  sup = 0.184
{butter}:  sup = 0.055
{other vegetables}:  sup = 0.193
{long life bakery product}:  sup = 0.037
{pip fruit}:  sup = 0.076
{cream cheese}:  sup = 0.04
{whole milk}:  sup = 0.256
{yogurt}:  sup = 0.14
{tropical fruit}:  sup = 0.105
{coffee}:  sup = 0.058
{margarine}:  sup = 0.059
{citrus fruit}:  sup = 0.083
{yogurt, whipped/sour cream}:  sup = 0.021
{other vegetables, yogurt}:  sup = 0.043
{other vegetables, pip fruit}:  sup = 0.026
{other vegetables, pastry}:  sup = 0.023
{other vegetables, shopping bags}:  sup = 0.023
{other vegetables, sausage}:  sup = 0.027
{whole milk, bottled beer}:  sup = 0.02
{shopping bags, whole milk}:  sup = 0.025
{other vegetables, citrus fruit}:  sup = 0.029
{whole milk, fruit/vegetable juice}:  sup = 0.027
{whole milk, frankfurter}:  sup = 0.021
{newspapers, whole milk}:  sup = 0.027
{whole milk, margarine}:  sup = 0.024
{pip fruit, tropical fruit}:  sup = 0.02
{whole milk, pip fruit}:  sup = 0.03
{whole milk, rolls/buns}:  sup = 0.057
{whole milk, beef}:  sup = 0.021
{whole milk, sausage}:  sup = 0.03
{frozen vegetables, whole milk}:  sup = 0.02
{pastry, rolls/buns}:  sup = 0.021
{other vegetables, fruit/vegetable juice}:  sup = 0.021
{other vegetables, domestic eggs}:  sup = 0.022
```

```
{other vegetables, butter}:  sup = 0.02
{yogurt, rolls/buns}:  sup = 0.034
{soda, bottled water}:  sup = 0.029
{soda, tropical fruit}:  sup = 0.021
{soda, yogurt}:  sup = 0.027
{whole milk, pastry}:  sup = 0.033
{root vegetables, yogurt}:  sup = 0.026
{whole milk, brown bread}:  sup = 0.025
{domestic eggs, whole milk}:  sup = 0.03
{soda, pastry}:  sup = 0.021
{whole milk, soda}:  sup = 0.04
{other vegetables, soda}:  sup = 0.033
{whole milk, pork}:  sup = 0.022
{other vegetables, pork}:  sup = 0.022
{whole milk, whipped/sour cream}:  sup = 0.032
{other vegetables, whipped/sour cream}:  sup = 0.029
{whole milk, root vegetables}:  sup = 0.049
{bottled water, rolls/buns}:  sup = 0.024
{shopping bags, soda}:  sup = 0.025
{sausage, rolls/buns}:  sup = 0.031
{sausage, soda}:  sup = 0.024
{tropical fruit, rolls/buns}:  sup = 0.025
{root vegetables, tropical fruit}:  sup = 0.021
{other vegetables, root vegetables}:  sup = 0.047
{root vegetables, rolls/buns}:  sup = 0.024
{soda, rolls/buns}:  sup = 0.038
{citrus fruit, yogurt}:  sup = 0.022
{whole milk, citrus fruit}:  sup = 0.031
{whole milk, tropical fruit}:  sup = 0.042
{yogurt, bottled water}:  sup = 0.023
{whole milk, bottled water}:  sup = 0.034
{whole milk, curd}:  sup = 0.026
{other vegetables, tropical fruit}:  sup = 0.036
{other vegetables, bottled water}:  sup = 0.025
{other vegetables, rolls/buns}:  sup = 0.043
{whole milk, yogurt}:  sup = 0.056
{whole milk, butter}:  sup = 0.028
{other vegetables, whole milk}:  sup = 0.075
{tropical fruit, yogurt}:  sup = 0.029
{other vegetables, whole milk, yogurt}:  sup = 0.022
{other vegetables, whole milk, root vegetables}:  sup = 0.023
```

[10]:
```python
# generating rules based on minimum confidence.
ar = generate_rules(f, a_data, min_confidence=0.3, verbose=True)
```

```
{yogurt} ---> {other vegetables}:  conf = 0.311, sup = 0.043
{pip fruit} ---> {other vegetables}:  conf = 0.345, sup = 0.026
{citrus fruit} ---> {other vegetables}:  conf = 0.349, sup = 0.029
```

```
{fruit/vegetable juice} ---> {whole milk}:  conf = 0.368, sup = 0.027
{frankfurter} ---> {whole milk}:  conf = 0.348, sup = 0.021
{newspapers} ---> {whole milk}:  conf = 0.343, sup = 0.027
{margarine} ---> {whole milk}:  conf = 0.413, sup = 0.024
{pip fruit} ---> {whole milk}:  conf = 0.398, sup = 0.03
{rolls/buns} ---> {whole milk}:  conf = 0.308, sup = 0.057
{beef} ---> {whole milk}:  conf = 0.405, sup = 0.021
{sausage} ---> {whole milk}:  conf = 0.318, sup = 0.03
{frozen vegetables} ---> {whole milk}:  conf = 0.425, sup = 0.02
{domestic eggs} ---> {other vegetables}:  conf = 0.351, sup = 0.022
{butter} ---> {other vegetables}:  conf = 0.361, sup = 0.02
{pastry} ---> {whole milk}:  conf = 0.374, sup = 0.033
{brown bread} ---> {whole milk}:  conf = 0.389, sup = 0.025
{domestic eggs} ---> {whole milk}:  conf = 0.473, sup = 0.03
{pork} ---> {whole milk}:  conf = 0.384, sup = 0.022
{pork} ---> {other vegetables}:  conf = 0.376, sup = 0.022
{whipped/sour cream} ---> {whole milk}:  conf = 0.45, sup = 0.032
{whipped/sour cream} ---> {other vegetables}:  conf = 0.403, sup = 0.029
{root vegetables} ---> {whole milk}:  conf = 0.449, sup = 0.049
{sausage} ---> {rolls/buns}:  conf = 0.326, sup = 0.031
{root vegetables} ---> {other vegetables}:  conf = 0.435, sup = 0.047
{citrus fruit} ---> {whole milk}:  conf = 0.369, sup = 0.031
{tropical fruit} ---> {whole milk}:  conf = 0.403, sup = 0.042
{bottled water} ---> {whole milk}:  conf = 0.311, sup = 0.034
{curd} ---> {whole milk}:  conf = 0.49, sup = 0.026
{tropical fruit} ---> {other vegetables}:  conf = 0.342, sup = 0.036
{yogurt} ---> {whole milk}:  conf = 0.402, sup = 0.056
{butter} ---> {whole milk}:  conf = 0.497, sup = 0.028
{other vegetables} ---> {whole milk}:  conf = 0.387, sup = 0.075
{whole milk, yogurt} ---> {other vegetables}:  conf = 0.397, sup = 0.022
{other vegetables, yogurt} ---> {whole milk}:  conf = 0.513, sup = 0.022
{whole milk, root vegetables} ---> {other vegetables}:  conf = 0.474, sup =
0.023
{other vegetables, root vegetables} ---> {whole milk}:  conf = 0.489, sup =
0.023
{other vegetables, whole milk} ---> {root vegetables}:  conf = 0.31, sup = 0.023
```

3 Comments on the results of Apriori Algorithm.

Using the apriori algorithm we have generated some rules that have confidence more than the minimum confidence. These association rules describe that these items occur together. In other words, if I were to shop domestic eggs, then there is a 47.3% (conf = 0.473) chance of me also buying whole milk.

These types of association rules are used by Amazon, eBay, and other ecommerce websites to suggest what people bought with the item you are currently buying.

There are also other interesting uses. For example, IKEA uses this consumer data to build new stores' layout in a way that arranges the items with strong confidence in association rules near each other to instill the thought of buying in the customers.

This has hugely increased the profits for the company and is now incorporated by every store around the world.

4 PART - 3: INTEREST FACTOR

[12]:
```
int_factor = {}
for item in ar:
    print("A:", item[0], "B:", item[1], "S[AB]:", item[3], "S[A]:",␣
 ↪a_data[item[0]], "S[B]:", a_data[item[1]])
    int_factor[item[0]] = [item[1], round(item[3]/
 ↪(a_data[item[1]]*a_data[item[0]]), 3)]
# int_factor
```

```
A: frozenset({'yogurt'}) B: frozenset({'other vegetables'}) S[AB]:
0.04341637010676157 S[A]: 0.13950177935943062 S[B]: 0.1934926283680732
A: frozenset({'pip fruit'}) B: frozenset({'other vegetables'}) S[AB]:
0.026131164209456024 S[A]: 0.07564819522114896 S[B]: 0.1934926283680732
A: frozenset({'citrus fruit'}) B: frozenset({'other vegetables'}) S[AB]:
0.02887646161667514 S[A]: 0.08276563294356888 S[B]: 0.1934926283680732
A: frozenset({'fruit/vegetable juice'}) B: frozenset({'whole milk'}) S[AB]:
0.026639552618200304 S[A]: 0.0722928317234367 S[B]: 0.25551601423487547
A: frozenset({'frankfurter'}) B: frozenset({'whole milk'}) S[AB]:
0.020538891713268937 S[A]: 0.058973055414336555 S[B]: 0.25551601423487547
A: frozenset({'newspapers'}) B: frozenset({'whole milk'}) S[AB]:
0.027351296390442297 S[A]: 0.07981698017285206 S[B]: 0.25551601423487547
A: frozenset({'margarine'}) B: frozenset({'whole milk'}) S[AB]:
0.024199288256227757 S[A]: 0.05856634468734113 S[B]: 0.25551601423487547
A: frozenset({'pip fruit'}) B: frozenset({'whole milk'}) S[AB]:
0.030096593797661414 S[A]: 0.07564819522114896 S[B]: 0.25551601423487547
A: frozenset({'rolls/buns'}) B: frozenset({'whole milk'}) S[AB]:
0.05663446873411286 S[A]: 0.18393492628368074 S[B]: 0.25551601423487547
A: frozenset({'beef'}) B: frozenset({'whole milk'}) S[AB]: 0.021250635485551093
S[A]: 0.05246568378240976 S[B]: 0.25551601423487547
A: frozenset({'sausage'}) B: frozenset({'whole milk'}) S[AB]: 0.0298932384341637
S[A]: 0.09395017793594305 S[B]: 0.25551601423487547
A: frozenset({'frozen vegetables'}) B: frozenset({'whole milk'}) S[AB]:
0.02043721403152008 S[A]: 0.04809354346720895 S[B]: 0.25551601423487547
A: frozenset({'domestic eggs'}) B: frozenset({'other vegetables'}) S[AB]:
0.02226741230299949 S[A]: 0.06344687341128623 S[B]: 0.1934926283680732
A: frozenset({'butter'}) B: frozenset({'other vegetables'}) S[AB]:
0.020030503304524657 S[A]: 0.05541433655312659 S[B]: 0.1934926283680732
A: frozenset({'pastry'}) B: frozenset({'whole milk'}) S[AB]:
0.033248601931875954 S[A]: 0.08896797153024912 S[B]: 0.25551601423487547
A: frozenset({'brown bread'}) B: frozenset({'whole milk'}) S[AB]:
0.02521606507371632 S[A]: 0.06487036095577021 S[B]: 0.25551601423487547
```

```
A: frozenset({'domestic eggs'}) B: frozenset({'whole milk'}) S[AB]:
0.029994916115912557 S[A]: 0.06344687341128623 S[B]: 0.25551601423487547
A: frozenset({'pork'}) B: frozenset({'whole milk'}) S[AB]: 0.022165734621250637
S[A]: 0.05765124555160142 S[B]: 0.25551601423487547
A: frozenset({'pork'}) B: frozenset({'other vegetables'}) S[AB]:
0.021657346212506354 S[A]: 0.05765124555160142 S[B]: 0.1934926283680732
A: frozenset({'whipped/sour cream'}) B: frozenset({'whole milk'}) S[AB]:
0.032231825114387394 S[A]: 0.07168276563294357 S[B]: 0.25551601423487547
A: frozenset({'whipped/sour cream'}) B: frozenset({'other vegetables'}) S[AB]:
0.02887646161667514 S[A]: 0.07168276563294357 S[B]: 0.1934926283680732
A: frozenset({'root vegetables'}) B: frozenset({'whole milk'}) S[AB]:
0.048906964921199794 S[A]: 0.10899847483477376 S[B]: 0.25551601423487547
A: frozenset({'sausage'}) B: frozenset({'rolls/buns'}) S[AB]:
0.030604982206405694 S[A]: 0.09395017793594305 S[B]: 0.18393492628368074
A: frozenset({'root vegetables'}) B: frozenset({'other vegetables'}) S[AB]:
0.047381799694966954 S[A]: 0.10899847483477376 S[B]: 0.1934926283680732
A: frozenset({'citrus fruit'}) B: frozenset({'whole milk'}) S[AB]:
0.030503304524656837 S[A]: 0.08276563294356888 S[B]: 0.25551601423487547
A: frozenset({'tropical fruit'}) B: frozenset({'whole milk'}) S[AB]:
0.04229791560752415 S[A]: 0.10493136756481952 S[B]: 0.25551601423487547
A: frozenset({'bottled water'}) B: frozenset({'whole milk'}) S[AB]:
0.03436705643111337 S[A]: 0.11052364006100661 S[B]: 0.25551601423487547
A: frozenset({'curd'}) B: frozenset({'whole milk'}) S[AB]: 0.026131164209456024
S[A]: 0.05327910523640061 S[B]: 0.25551601423487547
A: frozenset({'tropical fruit'}) B: frozenset({'other vegetables'}) S[AB]:
0.035892221657346214 S[A]: 0.10493136756481952 S[B]: 0.1934926283680732
A: frozenset({'yogurt'}) B: frozenset({'whole milk'}) S[AB]: 0.05602440264361973
S[A]: 0.13950177935943062 S[B]: 0.25551601423487547
A: frozenset({'butter'}) B: frozenset({'whole milk'}) S[AB]: 0.02755465175394001
S[A]: 0.05541433655312659 S[B]: 0.25551601423487547
A: frozenset({'other vegetables'}) B: frozenset({'whole milk'}) S[AB]:
0.07483477376715811 S[A]: 0.1934926283680732 S[B]: 0.25551601423487547
A: frozenset({'whole milk', 'yogurt'}) B: frozenset({'other vegetables'}) S[AB]:
0.02226741230299949 S[A]: 0.05602440264361973 S[B]: 0.1934926283680732
A: frozenset({'other vegetables', 'yogurt'}) B: frozenset({'whole milk'}) S[AB]:
0.02226741230299949 S[A]: 0.04341637010676157 S[B]: 0.25551601423487547
A: frozenset({'whole milk', 'root vegetables'}) B: frozenset({'other
vegetables'}) S[AB]: 0.023182511438739197 S[A]: 0.048906964921199794 S[B]:
0.1934926283680732
A: frozenset({'other vegetables', 'root vegetables'}) B: frozenset({'whole
milk'}) S[AB]: 0.023182511438739197 S[A]: 0.047381799694966954 S[B]:
0.25551601423487547
A: frozenset({'other vegetables', 'whole milk'}) B: frozenset({'root
vegetables'}) S[AB]: 0.023182511438739197 S[A]: 0.07483477376715811 S[B]:
0.10899847483477376
```

[12]:
```
print("TOP-5 rules when sorted by Support: ")
ar_sorted_support = sorted(ar, key=lambda x: x[3], reverse=True)
for i in range(5):
    print(ar_sorted_support[i][0], "--->", ar_sorted_support[i][1], ": support␣
 ↪= ", round(ar_sorted_support[i][3], 3))


print("\n")

print("TOP-5 rules when sorted by Confidence: ")
ar_sorted = sorted(ar, key=lambda x: x[2], reverse=True)
for i in range(5):
    print(ar_sorted[i][0], "--->", ar_sorted[i][1], ": conf = ",␣
 ↪ar_sorted[i][2])

print("\n")

print("TOP-5 rules when sorted by Interest Factor: ")
int_factor_sorted = sorted(int_factor.items(), key=lambda x: x[1][1],␣
 ↪reverse=True)
# print(int_factor_sorted)
for i in range(5):
    print(int_factor_sorted[i][0], "--->", int_factor_sorted[i][1][0], ":␣
 ↪interest_factor = ", int_factor_sorted[i][1][1])
```

```
TOP-5 rules when sorted by Support:
frozenset({'other vegetables'}) ---> frozenset({'whole milk'}) : support =
0.075
frozenset({'rolls/buns'}) ---> frozenset({'whole milk'}) : support =  0.057
frozenset({'yogurt'}) ---> frozenset({'whole milk'}) : support =  0.056
frozenset({'root vegetables'}) ---> frozenset({'whole milk'}) : support =  0.049
frozenset({'root vegetables'}) ---> frozenset({'other vegetables'}) : support =
0.047


TOP-5 rules when sorted by Confidence:
frozenset({'other vegetables', 'yogurt'}) ---> frozenset({'whole milk'}) : conf
=  0.5128805620608898
frozenset({'butter'}) ---> frozenset({'whole milk'}) : conf =
0.4972477064220184
frozenset({'curd'}) ---> frozenset({'whole milk'}) : conf =  0.4904580152671756
frozenset({'root vegetables', 'other vegetables'}) ---> frozenset({'whole
milk'}) : conf =  0.4892703862660944
frozenset({'root vegetables', 'whole milk'}) ---> frozenset({'other
vegetables'}) : conf =  0.47401247401247404
```

```
TOP-5 rules when sorted by Interest Factor:
frozenset({'other vegetables', 'whole milk'}) ---> frozenset({'root
vegetables'}) : interest_factor =  2.842
frozenset({'root vegetables', 'whole milk'}) ---> frozenset({'other
vegetables'}) : interest_factor =  2.45
frozenset({'root vegetables'}) ---> frozenset({'other vegetables'}) :
interest_factor =  2.247
frozenset({'whipped/sour cream'}) ---> frozenset({'other vegetables'}) :
interest_factor =  2.082
frozenset({'whole milk', 'yogurt'}) ---> frozenset({'other vegetables'}) :
interest_factor =  2.054
```

In the above cell, we calculated the Top5 rules based on support, confidence, and interest factor.

The highest support is for 'other vegetables' and 'whole milk'. These exist as a part of the association rule in 3 out of the 5 highest confidence rules and in 2 out of 5 highest interest factor rules. This shows that if a person buys whole milk, they are very likely to buy other vegetables and vice versa.