Market Basket Analysis using assocition rules - apriori technique in Two ways

Association rules analysis is a technique to uncover how items are associated to each other. There are three common ways to measure association.

Measure 1: Support. This says how popular an itemset is, it is number of times appear in total number of transaction. in other word we say frequency of item.

Measure 2: Confidence. This says how likely item Y is purchased when item X is purchased, expressed as {X -> Y}. This is measured by the proportion of transactions with item X, in which item Y also appears.

Measure 3: Lift. it is ratio of expected confidance to observed confidance. it is described as confidance of Y when item X was already known(x/y) to the confidance of Y when X item is unknown. in other words confidance of Y w.r.t. x and confiadnce of Y without X (means both are independent to each other).

support = occurance of item / total no of transaction.

confidance = support (X Union Y) / support(X).

lift = support (X Union Y)/ support(X) * support(Y) .

For more info report this link

[ ]:
```
#External package need to install
!pip install apyori
```

[ ]:
```
#import all required packages..
import pandas as pd
import numpy as np
from apyori import apriori
```

[ ]:
```
#loading market basket dataset..

df = pd.read_csv("../input/basket-optimisation/Market_Basket_Optimisation.
↪csv",header=None)
```

[ ]:
```
df.head()
```

[ ]:
```
#replacing empty value with 0.
df.fillna(0,inplace=True)
```

[ ]:
```
df.head()
```

[ ]:
```
#for using aprori need to convert data in list format..
#   transaction   =   [['apple','almonds'],['apple'],['banana','apple']]....
transactions = []
for i in range(0,len(df)):
    transactions.append([str(df.values[i,j]) for j in range(0,20) if str(df.
 ↪values[i,j])!="0"])
```

[ ]:
```
transactions[0]
```

[ ]:
```
#Call apriori function which requires minimum support, confidance and lift, min␣
 ↪length is combination of item default is 2".
rules = apriori(transactions,min_support=0.003,min_confidance=0.
 ↪2,min_lift=3,min_length=2)
```

[ ]:
```
#it generates a set of rules in a generator file...
rules
```

[ ]:
```
# all rules need to be converted in a list..
Results = list(rules)
Results
```

[ ]:
```
#convert result in a dataframe for further operation...
df_results = pd.DataFrame(Results)
```

[ ]:
```
# as we see order statistics itself a list so need to be converted in proper␣
 ↪format..
df_results.head()
```

[ ]:
```
#keep support in a separate data frame so we can use later..
support = df_results.support
```

[ ]:
```
'''
convert orderstatistic in a proper format.
order statistic has lhs => rhs as well rhs => lhs we can choose any one for␣
 ↪convience i choose first one which is␣
 ↪'df_results['ordered_statistics'][i][0]'
'''

#all four empty list which will contain lhs, rhs, confidance and lift␣
 ↪respectively.

first_values = []
second_values = []
third_values = []
```

```
fourth_value = []

# loop number of rows time and append 1 by 1 value in a separate list.. first␣
↪and second element was frozenset which need to be converted in list..
for i in range(df_results.shape[0]):
    single_list = df_results['ordered_statistics'][i][0]
    first_values.append(list(single_list[0]))
    second_values.append(list(single_list[1]))
    third_values.append(single_list[2])
    fourth_value.append(single_list[3])
```

[ ]:
```
#convert all four list into dataframe for further operation..
lhs = pd.DataFrame(first_values)
rhs= pd.DataFrame(second_values)
confidance=pd.DataFrame(third_values,columns=['Confidance'])
lift=pd.DataFrame(fourth_value,columns=['lift'])
```

[ ]:
```
#concat all list together in a single dataframe
df_final = pd.concat([lhs,rhs,support,confidance,lift], axis=1)
df_final
```

[ ]:
```
'''
 we have some of place only 1 item in lhs and some place 3 or more so we need␣
↪to a proper represenation for User to understand.
 removing none with ' ' extra so when we combine three column in 1 then only 1␣
↪item will be there with spaces which is proper rather than none.
 example : coffee,none,none which converted to coffee, ,
'''
df_final.fillna(value=' ', inplace=True)
```

[ ]:
```
#set column name
df_final.columns = ['lhs',1,2,'rhs','support','confidance','lift']
```

[ ]:
```
#add all three column because those where the lhs itemset only
df_final['lhs'] = df_final['lhs']+str(", ")+df_final[1]+str(", ")+df_final[2]
```

[ ]:
```
#drop those 1,2 column because now we already appended to lhs column..
df_final.drop(columns=[1,2],inplace=True)
```

[ ]:
```
#this is final output.. you can sort based on the support lift and confidance..
df_final.head()
```

Other way of doing Apriori in Python.

Why we doing it in this way -

1. Limitation of first approach was need to converted data in a list fomat. when we see real life a store has many thousands of sku in that case it is computationally expensive.

2. Apyori package is outdated. i mean there is no recent update from past few years.
3. Results are coming in improper format which need to represent properly and that need computational operation to perform.
4. mlxtend used two way based approach which generate frequent itemset and association rules over that. -check here for more info
5. mlxtend are proper and has community support.

[ ]:
```
'''
load apriori and association package from mlxtend.
Used different dataset because mlxtend need data in below format.

          itemname   apple banana  grapes

 we could have used above data as well but need to perform operation to bring␣
 ↪in this format instead of that used seperate data only.
'''


from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules

df1 = pd.read_csv("../input/ecommerce-data/data.csv", encoding="ISO-8859-1")
df1.head()
```

[ ]:
```
# data has many country choose any one for check..
df1.Country.value_counts().head(5)
```

[ ]:
```
#using only France country data for now can check for other as well..
df1 = df1[df1.Country == 'France']
```

[ ]:
```
# some spaces are there in description need to remove else later operation it␣
 ↪will create problem..
df1['Description'] = df1['Description'].str.strip()
```

[ ]:
```
#some of transaction quantity is negative which can not be possible remove that.
df1 = df1[df1.Quantity >0]
```

[ ]:
```
df1[df1.Country == 'France'].head(10)
```

[ ]:
```
#convert data in format which it require converting using pivot table and␣
 ↪Quantity sum as values. fill 0 if any nan values
```

```
basket = pd.
↪pivot_table(data=df1,index='InvoiceNo',columns='Description',values='Quantity',␣
↪\
                                        aggfunc='sum',fill_value=0)
```

[ ]:
```
basket.head()
```

[ ]:
```
#this to check correctness after binning it to 1 at below code..
basket['ALARM CLOCK BAKELIKE RED'].head(10)
```

[ ]:
```
# we dont need quantity sum we need either has taken or not so if user has␣
↪taken that item mark as 1 else he has not taken 0.

def convert_into_binary(x):
    if x > 0:
        return 1
    else:
        return 0
```

[ ]:
```
basket_sets = basket.applymap(convert_into_binary)
```

[ ]:
```
# above steps we can same item has quantity now converted to 1 or 0.
basket_sets['ALARM CLOCK BAKELIKE RED'].head()
```

[ ]:
```
#remove postage item as it is just a seal which almost all transaction contain.
basket_sets.drop(columns=['POSTAGE'],inplace=True)
```

[ ]:
```
#call apriori function and pass minimum support here we are passing 7%. means 7␣
↪times in total number of transaction that item was present.
frequent_itemsets = apriori(basket_sets, min_support=0.07, use_colnames=True)
```

[ ]:
```
#it will generate frequent itemsets using two step approch
frequent_itemsets
```

[ ]:
```
# we have association rules which need to put on frequent itemset. here we are␣
↪setting based on lift and has minimum lift as 1
rules_mlxtend = association_rules(frequent_itemsets, metric="lift",␣
↪min_threshold=1)
rules_mlxtend.head()
```

[ ]:
```
#     rules_mlxtend.rename(columns={'antecedents':'lhs','consequents':'rhs'})

# as based business use case we can sort based on confidance and lift.
rules_mlxtend[ (rules_mlxtend['lift'] >= 4) & (rules_mlxtend['confidence'] >= 0.
↪8) ]
```

Directed graph below is built for this rule and shown below. it will have always incoming and outcoming edges. Incoming edge(s) will represent antecedants and the stub (arrow) will be next to node.

```
[ ]: #plotting output in a graph plot.

     import networkx as nx
     import matplotlib.pyplot as plt

     def draw_graph(rules, rules_to_show):
         G1 = nx.DiGraph()
         color_map=[]
         N = 50
         colors = np.random.rand(N)
         strs=['R0', 'R1', 'R2', 'R3', 'R4', 'R5', 'R6', 'R7', 'R8', 'R9', 'R10',␣
      ↪'R11']

         for i in range(rules_to_show):
             G1.add_nodes_from(["R"+str(i)])
             for a in rules.iloc[i]['antecedents']:
                 G1.add_nodes_from([a])
                 G1.add_edge(a, "R"+str(i), color=colors[i] , weight = 2)
             for c in rules.iloc[i]['consequents']:
                 G1.add_nodes_from([c])
                 G1.add_edge("R"+str(i), c, color=colors[i],  weight=2)

         for node in G1:
             found_a_string = False
             for item in strs:
                 if node==item:
                     found_a_string = True
             if found_a_string:
                 color_map.append('yellow')
             else:
                 color_map.append('green')

         edges = G1.edges()
         colors = [G1[u][v]['color'] for u,v in edges]
         weights = [G1[u][v]['weight'] for u,v in edges]

         pos = nx.spring_layout(G1, k=16, scale=1)
         nx.draw(G1, pos, edges=edges, node_color = color_map, edge_color=colors,␣
      ↪width=weights, font_size=16,
                 with_labels=False)

         for p in pos:  # raise text positions
             pos[p][1] += 0.07
             nx.draw_networkx_labels(G1, pos)
```

```
          plt.show()

draw_graph (rules_mlxtend, 10)
```