Web Mining Lab - 11

Aadhitya Swarnesh

[redacted]

- 20 May 2021

Question 1

Use the Hungarian News Clickstream dataset and perform the Associative Rule Mining using Apriori Algorithm and identify the most clicked upon site. Form Strong Rules of prediction with support and confidence threshold scores of greater than 50%

We will first obtain the dataset, we use the **Hungarian Clickstream Datase**t. We can directly import this and begin with our Rule Mining procedure, We will first observe the dataset, and as it is in the form of a **Comma Separated File** format, so we import it easily.

For the pre-processing part, we had observed that there were a few instance where "110" was marked as "0 11" and "116" as "0 6 11", so we replace these terms by their correct values in the dataset, and now it is ready for applying the algorithm.

Now that the dataset is ready, we will proceed to the actual Rule Mining Process, we will in this lab use the **Apriori Algorithm** to perform the Associated Rule Mining.

The implementation of these algorithms is done in the **Python Programming Language**. We have utilised both the "apyori" python library, and as well as implemented the whole algorithm manually without the use of any library implementations.

After the model has trained on the data both using the library and the custom implementations, and the results look promising. They have been shown below in the results section. For the custom code, we have also implemented a few additional utitity function which are a part of the apriori algorithm like — "frequency", "support" and "confidence", and many others as visible in the code section below.

For the custom code implementation, we have first read the file, then taken the items and formed a item set using all the transactions in the dataset, then we have filtered these item sets by their support value which should be greater than the threshold set in the question. We then extend it to form rules, we then filter these rules by their confidence scores. We perform the same process repeatedly to get higher order item sets and more complex rules which are extracted from the dataset. After training we here print the total number of rules and item sets, and also print a few rules to get a general idea about how rules look.

Result :

As specified before, we are asked to find the most frequently occurring news items. We do just the same and the results of this are as follows. These are the top 10 most frequently occurring news items in the dataset :

```
The most Frequent News Items are :
34 occuring 413 times
85 occuring 413 times
86 occuring 413 times
90 occuring 413 times
76 occuring 412 times
67 occuring 410 times
59 occuring 399 times
93 occuring 378 times
2 occuring 370 times
36 occuring 345 times
```

We now apply the Algorithm using the **apyori** library of python. It returns the item sets and the rules. We have displayed a few of those rules here.

```
Rule : (2 , 34 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)   =>   (67 , 36),   support=0.6319612590799032, confidence=1.0, lift=1.2076023391812867
Rule : (2 , 34 , 67 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 93)   =>   (59 , 36),   support=0.6319612590799032, confidence=1.0, lift=1.1971014492753624
Rule : (2 , 34 , 67 , 39 , 76 , 85 , 86 , 23 , 90 , 59 , 93)   =>   (36 , 52),   support=0.6319612590799032, confidence=1.0, lift=1.2076023391812867
Rule : (2 , 34 , 67 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)   =>   (36 , 39),   support=0.6319612590799032, confidence=1.0, lift=1.5643939393939394
Rule : (2 , 67 , 36 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 59)   =>   (34 , 93),   support=0.6319612590799032, confidence=1.0, lift=1.0925925925925926
Rule : (2 , 67 , 36 , 39 , 76 , 52 , 85 , 86 , 23 , 59 , 93)   =>   (34 , 90),   support=0.6319612590799032, confidence=1.0, lift=1.0
Rule : (2 , 67 , 36 , 39 , 76 , 52 , 85 , 23 , 90 , 59 , 93)   =>   (34 , 86),   support=0.6319612590799032, confidence=1.0, lift=1.0
Rule : (2 , 67 , 36 , 39 , 76 , 52 , 86 , 23 , 90 , 59 , 93)   =>   (34 , 85),   support=0.6319612590799032, confidence=1.0, lift=1.0
Rule : (2 , 67 , 36 , 39 , 52 , 85 , 86 , 23 , 90 , 59 , 93)   =>   (34 , 76),   support=0.6319612590799032, confidence=1.0, lift=1.0024271844660195
Rule : (2 , 36 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)   =>   (34 , 67),   support=0.6319612590799032, confidence=1.0, lift=1.0073170731707317
Rule : (2 , 67 , 36 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 93)   =>   (34 , 59),   support=0.6319612590799032, confidence=1.0, lift=1.0350877192982457
Rule : (2 , 67 , 36 , 39 , 76 , 85 , 86 , 23 , 90 , 59 , 93)   =>   (34 , 52),   support=0.6319612590799032, confidence=1.0, lift=1.2076023391812867
Rule : (2 , 67 , 36 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)   =>   (34 , 39),   support=0.6319612590799032, confidence=1.0, lift=1.3812709030100334
Rule : (2 , 67 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)   =>   (34 , 36),   support=0.6319612590799032, confidence=1.0, lift=1.1971014492753624
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 52 , 85 , 86 , 90 , 59)   =>   (93 , 23),   support=0.6319612590799032, confidence=1.0, lift=1.214705882352941
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 52 , 85 , 86 , 59 , 93)   =>   (90 , 23),   support=0.6319612590799032, confidence=1.0, lift=1.214705882352941
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 52 , 85 , 90 , 59 , 93)   =>   (86 , 23),   support=0.6319612590799032, confidence=1.0, lift=1.214705882352941
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 52 , 86 , 90 , 59 , 93)   =>   (85 , 23),   support=0.6319612590799032, confidence=1.0, lift=1.214705882352941
Rule : (2 , 34 , 36 , 67 , 39 , 52 , 85 , 86 , 90 , 59 , 93)   =>   (76 , 23),   support=0.6319612590799032, confidence=1.0, lift=1.2182890855457227
Rule : (2 , 34 , 36 , 39 , 76 , 52 , 85 , 86 , 90 , 59 , 93)   =>   (67 , 23),   support=0.6319612590799032, confidence=1.0, lift=1.225519287833828
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 52 , 85 , 86 , 90 , 93)   =>   (59 , 23),   support=0.6319612590799032, confidence=1.0, lift=1.214705882352941
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 85 , 86 , 90 , 59 , 93)   =>   (52 , 23),   support=0.6319612590799032, confidence=1.0, lift=1.3585526315789473
```

We have obtained a total of around 2 lakh rule and 12000 item sets using this algorithm.

```
We have a total of 2268881 strong rules whose support and confidence matches our threshold conditions.
```

```
We have a total of 12159 sets, which are the different combinations which have the above said support of 50.0 %.
```

As specified before, we have also implemented the whole Apriori algorithm without the use of any libraries, and its results are as follows :

The following diagram shows a few rule and item sets we have obtained with their support and confidence scores respectively :

```
------------------------ Top Item Sets ------------------------
item: ('59', '67', '3') , 0.501
item: ('59', '67', '3', '34') , 0.501
item: ('59', '67', '85', '3') , 0.501
item: ('59', '67', '3', '86') , 0.501
item: ('59', '67', '3', '90') , 0.501
item: ('59', '76', '67', '3') , 0.501
item: ('59', '85', '3', '67', '90') , 0.501
item: ('59', '85', '3', '34', '67') , 0.501
item: ('59', '76', '85', '3', '67') , 0.501
item: ('59', '3', '34', '67', '90') , 0.501

------------------------ Top RULES ------------------------
Rule: ('34',) ==> ('59', '67', '3') , 0.501
Rule: ('85',) ==> ('59', '67', '3') , 0.501
Rule: ('86',) ==> ('59', '67', '3') , 0.501
Rule: ('90',) ==> ('59', '67', '3') , 0.501
Rule: ('85',) ==> ('59', '67', '90', '3') , 0.501
Rule: ('90',) ==> ('59', '67', '85', '3') , 0.501
Rule: ('90', '85') ==> ('59', '67', '3') , 0.501
Rule: ('85',) ==> ('59', '67', '3', '34') , 0.501
Rule: ('34',) ==> ('59', '67', '85', '3') , 0.501
Rule: ('85', '34') ==> ('59', '67', '3') , 0.501
```

We have extracted around 22 lakh rule sets using this implementation

```
We have a total of 2256722 strong rules whose support and confidence matches our threshold conditions.
```

It is important to note here that the number of rules obtained by the library and the algorithm's direct implementation have resulted in almost same number of **22lakhs**. This proves the validity of our implementation of the algorithm.

Code :

We have attached the complete code of both these sections.

The **code** that has delivered these results are as follows —

Apriori Algorithm for Associated Rule Mining

Question : Find the frequent news itemsets using apriori algorithm and generate strong rules with

- Support = 50%
- Confidence = 50%

In [ ]:

```
# Set the conditions given

minSupport = 0.5
minConfidence = 0.5
```

1. Using Apyriori Library

1.1 Import Libraries and dataset

In [ ]:

```
# Install this library first

!pip install apyori
```

```
Collecting apyori
  Downloading https://files.pythonhosted.org/packages/5e/62/5ffde5c473ea4b033
490617ec5caa80d59804875ad3c3c57c0976533a21a/apyori-1.1.2.tar.gz
Building wheels for collected packages: apyori
  Building wheel for apyori (setup.py) ... done
  Created wheel for apyori: filename=apyori-1.1.2-cp37-none-any.whl size=5975
sha256=6d56c2bf7814b2efecdd1aed130a65e3a4c128171ff4de91887263683eb29173
  Stored in directory: /root/.cache/pip/wheels/5d/92/bb/474bbadbc8c0062b9eb16
8f69982a0443263f8ab1711a8cad0
Successfully built apyori
Installing collected packages: apyori
Successfully installed apyori-1.1.2
```

In [ ]:

```
# Import the necessary libraries

import numpy as np
import pandas as pd
from apyori import apriori, load_transactions
import csv

from collections import Counter
```

In [ ]:

```
# Import the dataset

df = pd.read_csv("HungarianNewsItems_indexedforARM.csv", header=None)
df.head()
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
0	1	3	9	13	23	25	34	36	38	40	52	54	59	63	67	76	85	86	90	93	98	107	113
1	2	3	9	14	23	26	34	36	39	40	52	55	59	63	67	76	85	86	90	93	99	108	114
2	2	4	9	15	23	27	34	36	39	41	52	55	59	63	67	76	85	86	90	93	99	108	115
3	1	3	10	15	23	25	34	36	38	41	52	54	59	63	67	76	85	86	90	93	98	107	113
4	2	3	9	16	24	28	34	37	39	40	53	54	59	63	67	76	85	86	90	94	99	109	114

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 413 entries, 0 to 412
Data columns (total 23 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       413 non-null    int64 
 1   1       413 non-null    int64 
 2   2       413 non-null    int64 
 3   3       413 non-null    int64 
 4   4       413 non-null    int64 
 5   5       413 non-null    int64 
 6   6       413 non-null    int64 
 7   7       413 non-null    int64 
 8   8       413 non-null    int64 
 9   9       413 non-null    int64 
 10  10      413 non-null    int64 
 11  11      413 non-null    int64 
 12  12      413 non-null    int64 
 13  13      413 non-null    int64 
 14  14      413 non-null    int64 
 15  15      413 non-null    int64 
 16  16      413 non-null    int64 
 17  17      413 non-null    int64 
 18  18      413 non-null    int64 
 19  19      413 non-null    int64 
 20  20      413 non-null    int64 
 21  21      413 non-null    object
 22  22      413 non-null    object
dtypes: int64(21), object(2)
memory usage: 74.3+ KB
```

1.2 Pre-processing the Dataset

In [ ]:

```
# Convert these outliers into integers

df[21] = df[21].replace('0 11', 110)
df[22] = df[22].replace('0 116', 110)
```

In [ ]:

```
# FOrce all the data types to be integers

df = df.astype("int64")
```

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 413 entries, 0 to 412
Data columns (total 23 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       413 non-null    int64
 1   1       413 non-null    int64
 2   2       413 non-null    int64
 3   3       413 non-null    int64
 4   4       413 non-null    int64
 5   5       413 non-null    int64
 6   6       413 non-null    int64
 7   7       413 non-null    int64
 8   8       413 non-null    int64
 9   9       413 non-null    int64
 10  10      413 non-null    int64
 11  11      413 non-null    int64
 12  12      413 non-null    int64
 13  13      413 non-null    int64
 14  14      413 non-null    int64
 15  15      413 non-null    int64
 16  16      413 non-null    int64
 17  17      413 non-null    int64
 18  18      413 non-null    int64
 19  19      413 non-null    int64
 20  20      413 non-null    int64
 21  21      413 non-null    int64
 22  22      413 non-null    int64
dtypes: int64(23)
memory usage: 74.3 KB
```

In [ ]:

```
# Save the pre-processed dataset
df.to_csv("HungarianNewsItems_preprocessed.csv", header=False, index=False)
```

1.2 Find Most Frequent News Items

In [ ]:

```
# Extract the data from the dataframe and save them as numpy arrays

data = df.to_numpy()
```

In [ ]:

```
# Sort the news items by their frequency of occurance

# Find the unique news items and their corresponding frequencies of occurance
news_items, news_frequencies = np.unique(data.flatten(), return_counts=True)
# Convert them into lists for them to be zipped together
news_items = list(news_items)
news_frequencies = list(news_frequencies)
# Zip the news items and their counts into a single list
news_items_zip = [i for i in zip(news_items, news_frequencies)]

# Sort them by the frequency counts in the decreasing order of frequency
news_items_zip.sort(key = lambda i : i[1], reverse=True)
```

In [ ]:

```
# Print the most frequent news items

print("The most Frequent News Items are : ")
for (news_temp, freq_temp) in news_items_zip[:10] :
    print(f"{news_temp} occuring {freq_temp} times")
```

```
The most Frequent News Items are :
34 occuring 413 times
85 occuring 413 times
86 occuring 413 times
90 occuring 413 times
76 occuring 412 times
67 occuring 410 times
59 occuring 399 times
93 occuring 378 times
2 occuring 370 times
36 occuring 345 times
```

1.3 Find the Rules with the given support and Confidence Thresholds

In [ ]:

```
# Find the rules based on the given support and confidence thresholds

rules = apriori(data, min_support = minSupport, min_confidence = minConfidence)
association_results = list(rules)
```

In [ ]:

```
print(f"We have a total of {len(association_results)} sets, which are the different combi
nations which have the above said support of {minSupport * 100} %.")
```

```
We have a total of 12159 sets, which are the different combinations which hav
e the above said support of 50.0 %.
```

This is the format of eery record of the `association_results`.

```
RelationRecord(items=frozenset({'beer', 'nuts'}),
               support=0.5,
               ordered_statistics=[OrderedStatistic(items_base=frozenset({'beer'
}),
                                                    items_add=frozenset({'nuts'}),
                                                    confidence=0.5,
                                                    lift=1.0),
                                   OrderedStatistic(items_base=frozenset({'nuts'
}),
                                                    items_add=frozenset({'beer'}),
                                                    confidence=1.0,
                                                    lift=1.0)])
```

In [ ]:

```
# Print a few rules

for relationRecord in association_results[-1:] :    # We print the rules for only the las
t rule set
    cur_support = relationRecord[1]
    for orderedStatistic in relationRecord[2][-50:] :   # We print only the last 50 rules
        cur_base_items = " , ".join([str(i) for i in list(orderedStatistic[0])])
        cur_add_items = " , ".join([str(i) for i in list(orderedStatistic[1])])
        cur_confidence = orderedStatistic[2]
        cur_lift = orderedStatistic[3]

        print(f"Rule : ({cur_base_items})   =>   ({cur_add_items}),   support={cur_suppor
t}, confidence={cur_confidence}, lift={cur_lift}")
```

Rule : (2 , 34 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)   =>   (67 , 3
6),   support=0.6319612590799032, confidence=1.0, lift=1.2076023391812867
Rule : (2 , 34 , 67 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 93)   =>   (59 , 3
6),   support=0.6319612590799032, confidence=1.0, lift=1.1971014492753624
Rule : (2 , 34 , 67 , 39 , 76 , 85 , 86 , 23 , 90 , 59 , 93)   =>   (36 , 5
2),   support=0.6319612590799032, confidence=1.0, lift=1.2076023391812867
Rule : (2 , 34 , 67 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)   =>   (36 , 3
9),   support=0.6319612590799032, confidence=1.0, lift=1.5643939393939394
Rule : (2 , 67 , 36 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 59)   =>   (34 , 9
3),   support=0.6319612590799032, confidence=1.0, lift=1.0925925925925926
Rule : (2 , 67 , 36 , 39 , 76 , 52 , 85 , 86 , 23 , 59 , 93)   =>   (34 , 9
0),   support=0.6319612590799032, confidence=1.0, lift=1.0
Rule : (2 , 67 , 36 , 39 , 76 , 52 , 85 , 23 , 90 , 59 , 93)   =>   (34 , 8
6),   support=0.6319612590799032, confidence=1.0, lift=1.0
Rule : (2 , 67 , 36 , 39 , 76 , 52 , 86 , 23 , 90 , 59 , 93)   =>   (34 , 8
5),   support=0.6319612590799032, confidence=1.0, lift=1.0
Rule : (2 , 67 , 36 , 39 , 52 , 85 , 86 , 23 , 90 , 59 , 93)   =>   (34 , 7
6),   support=0.6319612590799032, confidence=1.0, lift=1.0024271844660195
Rule : (2 , 36 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)   =>   (34 , 6
7),   support=0.6319612590799032, confidence=1.0, lift=1.0073170731707317
Rule : (2 , 67 , 36 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 93)   =>   (34 , 5
9),   support=0.6319612590799032, confidence=1.0, lift=1.0350877192982457
Rule : (2 , 67 , 36 , 39 , 76 , 85 , 86 , 23 , 90 , 59 , 93)   =>   (34 , 5
2),   support=0.6319612590799032, confidence=1.0, lift=1.2076023391812867
Rule : (2 , 67 , 36 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)   =>   (34 , 3
9),   support=0.6319612590799032, confidence=1.0, lift=1.3812709030100334
Rule : (2 , 67 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)   =>   (34 , 3
6),   support=0.6319612590799032, confidence=1.0, lift=1.1971014492753624
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 52 , 85 , 86 , 90 , 59)   =>   (93 , 2
3),   support=0.6319612590799032, confidence=1.0, lift=1.214705882352941
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 52 , 85 , 86 , 59 , 93)   =>   (90 , 2
3),   support=0.6319612590799032, confidence=1.0, lift=1.214705882352941
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 52 , 85 , 90 , 59 , 93)   =>   (86 , 2
3),   support=0.6319612590799032, confidence=1.0, lift=1.214705882352941
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 52 , 86 , 90 , 59 , 93)   =>   (85 , 2
3),   support=0.6319612590799032, confidence=1.0, lift=1.214705882352941
Rule : (2 , 34 , 36 , 67 , 39 , 52 , 85 , 86 , 90 , 59 , 93)   =>   (76 , 2
3),   support=0.6319612590799032, confidence=1.0, lift=1.2182890855457227
Rule : (2 , 34 , 36 , 39 , 76 , 52 , 85 , 86 , 90 , 59 , 93)   =>   (67 , 2
3),   support=0.6319612590799032, confidence=1.0, lift=1.225519287833828
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 52 , 85 , 86 , 90 , 93)   =>   (59 , 2
3),   support=0.6319612590799032, confidence=1.0, lift=1.214705882352941
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 85 , 86 , 90 , 59 , 93)   =>   (52 , 2
3),   support=0.6319612590799032, confidence=1.0, lift=1.3585526315789473
Rule : (2 , 34 , 36 , 67 , 76 , 52 , 85 , 86 , 90 , 59 , 93)   =>   (23 , 3
9),   support=0.6319612590799032, confidence=0.8729096989966556, lift=1.36557
4642748556
Rule : (2 , 34 , 67 , 39 , 76 , 52 , 85 , 86 , 90 , 59 , 93)   =>   (36 , 2
3),   support=0.6319612590799032, confidence=1.0, lift=1.3452768729641693
Rule : (2 , 67 , 36 , 39 , 76 , 52 , 85 , 86 , 90 , 59 , 93)   =>   (34 , 2
3),   support=0.6319612590799032, confidence=1.0, lift=1.214705882352941
Rule : (34 , 67 , 36 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 59)   =>   (2 , 9
3),   support=0.6319612590799032, confidence=1.0, lift=1.2328358208955223
Rule : (34 , 67 , 36 , 39 , 76 , 52 , 85 , 86 , 23 , 59 , 93)   =>   (2 , 9
0),   support=0.6319612590799032, confidence=1.0, lift=1.1162162162162161
Rule : (34 , 67 , 36 , 39 , 76 , 52 , 85 , 23 , 90 , 59 , 93)   =>   (2 , 8
6),   support=0.6319612590799032, confidence=1.0, lift=1.1162162162162161
Rule : (34 , 67 , 36 , 39 , 76 , 52 , 86 , 23 , 90 , 59 , 93)   =>   (2 , 8
5),   support=0.6319612590799032, confidence=1.0, lift=1.1162162162162161
Rule : (34 , 67 , 36 , 39 , 52 , 85 , 86 , 23 , 90 , 59 , 93)   =>   (2 , 7
6),   support=0.6319612590799032, confidence=1.0, lift=1.119241192411924
Rule : (34 , 36 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)   =>   (2 , 6
7),   support=0.6319612590799032, confidence=1.0, lift=1.125340599455041
Rule : (34 , 67 , 36 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 93)   =>   (2 , 5
9),   support=0.6319612590799032, confidence=1.0, lift=1.1601123595505618

```
Rule : (34 , 67 , 36 , 39 , 76 , 85 , 86 , 23 , 90 , 59 , 93)    =>    (2 , 5
2),   support=0.6319612590799032, confidence=1.0, lift=1.3812709030100334
Rule : (34 , 67 , 36 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)    =>    (2 , 3
9),   support=0.6319612590799032, confidence=0.8585526315789475, lift=1.18589
37687026934
Rule : (34 , 67 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)    =>    (2 , 3
6),   support=0.6319612590799032, confidence=1.0, lift=1.3675496688741722
Rule : (67 , 36 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)    =>    (2 , 3
4),   support=0.6319612590799032, confidence=1.0, lift=1.1162162162162161
Rule : (34 , 67 , 36 , 39 , 76 , 52 , 85 , 86 , 90 , 59 , 93)    =>    (2 , 2
3),   support=0.6319612590799032, confidence=1.0, lift=1.3905723905723906
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 59)    =>    (9
3),   support=0.6319612590799032, confidence=1.0, lift=1.0925925925925926
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 52 , 85 , 86 , 23 , 59 , 93)    =>    (9
0),   support=0.6319612590799032, confidence=1.0, lift=1.0
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 52 , 85 , 23 , 90 , 59 , 93)    =>    (8
6),   support=0.6319612590799032, confidence=1.0, lift=1.0
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 52 , 86 , 23 , 90 , 59 , 93)    =>    (8
5),   support=0.6319612590799032, confidence=1.0, lift=1.0
Rule : (2 , 34 , 36 , 67 , 39 , 52 , 85 , 86 , 23 , 90 , 59 , 93)    =>    (7
6),   support=0.6319612590799032, confidence=1.0, lift=1.0024271844660195
Rule : (2 , 34 , 36 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)    =>    (6
7),   support=0.6319612590799032, confidence=1.0, lift=1.0073170731707317
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 93)    =>    (5
9),   support=0.6319612590799032, confidence=1.0, lift=1.0350877192982457
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 85 , 86 , 23 , 90 , 59 , 93)    =>    (5
2),   support=0.6319612590799032, confidence=1.0, lift=1.2076023391812867
Rule : (2 , 34 , 36 , 67 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)    =>    (3
9),   support=0.6319612590799032, confidence=1.0, lift=1.3812709030100334
Rule : (2 , 34 , 67 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)    =>    (3
6),   support=0.6319612590799032, confidence=1.0, lift=1.1971014492753624
Rule : (2 , 67 , 36 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)    =>    (3
4),   support=0.6319612590799032, confidence=1.0, lift=1.0
Rule : (2 , 34 , 36 , 67 , 39 , 76 , 52 , 85 , 86 , 90 , 59 , 93)    =>    (2
3),   support=0.6319612590799032, confidence=1.0, lift=1.214705882352941
Rule : (34 , 67 , 36 , 39 , 76 , 52 , 85 , 86 , 23 , 90 , 59 , 93)    =>
(2),   support=0.6319612590799032, confidence=1.0, lift=1.1162162162162161
```

In [ ]:

```
# Count the total number of rules

total_rule_count = 0
for relationRecord in association_results :
    for orderedStatistic in relationRecord[2] :
        total_rule_count += 1
```

In [ ]:

```
print(f"We have a total of {total_rule_count} strong rules whose support and confidence m
atches our threshold conditions.")
```

```
We have a total of 2268881 strong rules whose support and confidence matches
our threshold conditions.
```

2. Apriori Algorithm

In this section, we write our own version of the Apriori algorithm without making use of any libraries.

2.1 Import necessary Libraries

In [ ]:

```python
import sys

from itertools import chain, combinations
from collections import defaultdict
```

2.2 Utility Functions

In [ ]:

```python
# To find the non-empty subsets

def subsets(arr):
    """ Returns non empty subsets of arr"""
    return chain(*[combinations(arr, i + 1) for i, a in enumerate(arr)])
```

In [ ]:

```python
# Find and return item sets which satisfies the given support threshold conditions

def returnItemsWithMinSupport(itemSet, transactionList, minSupport, freqSet):
    """calculates the support for items in the itemSet and returns a subset
    of the itemSet each of whose elements satisfies the minimum support"""
    _itemSet = set()
    localSet = defaultdict(int)

    for item in itemSet:
        for transaction in transactionList:
            if item.issubset(transaction):
                freqSet[item] += 1
                localSet[item] += 1

    for item, count in localSet.items():
        support = float(count) / len(transactionList)

        if support >= minSupport:
            _itemSet.add(item)

    return _itemSet
```

In [ ]:

```python
def joinSet(itemSet, length):
    """Join a set with itself and returns the n-element itemsets"""
    return set(
        [i.union(j) for i in itemSet for j in itemSet if len(i.union(j)) == length]
    )
```

In [ ]:

```python
# Get the item sets from a given transaction list

def getItemSetTransactionList(data_iterator):
    transactionList = list()
    itemSet = set()
    for record in data_iterator:
        transaction = frozenset(record)
        transactionList.append(transaction)
        for item in transaction:
            itemSet.add(frozenset([item]))  # Generate 1-itemSets
    return itemSet, transactionList
```

In [ ]:

```python
# The main runner function

def runApriori(data_iter, minSupport, minConfidence):
    """
    run the apriori algorithm. data_iter is a record iterator
    Return both:
     - items (tuple, support)
     - rules ((pretuple, posttuple), confidence)
    """
    itemSet, transactionList = getItemSetTransactionList(data_iter)

    freqSet = defaultdict(int)
    largeSet = dict()
    # Global dictionary which stores (key=n-itemSets,value=support)
    # which satisfy minSupport

    assocRules = dict()
    # Dictionary which stores Association Rules

    oneCSet = returnItemsWithMinSupport(itemSet, transactionList, minSupport, freqSet)

    currentLSet = oneCSet
    k = 2
    while currentLSet != set([]):
        largeSet[k - 1] = currentLSet
        currentLSet = joinSet(currentLSet, k)
        currentCSet = returnItemsWithMinSupport(
            currentLSet, transactionList, minSupport, freqSet
        )
        currentLSet = currentCSet
        k = k + 1

    def getSupport(item):
        """local function which Returns the support of an item"""
        return float(freqSet[item]) / len(transactionList)

    toRetItems = []
    for key, value in largeSet.items():
        toRetItems.extend([(tuple(item), getSupport(item)) for item in value])

    toRetRules = []
    for key, value in list(largeSet.items())[1:]:
        for item in value:
            _subsets = map(frozenset, [x for x in subsets(item)])
            for element in _subsets:
                remain = item.difference(element)
                if len(remain) > 0:
                    confidence = getSupport(item) / getSupport(element)
                    if confidence >= minConfidence:
                        toRetRules.append(((tuple(element), tuple(remain)), confidence))
    return toRetItems, toRetRules
```

In [ ]:

```
# Print the rules

def printResults(items, rules):
    """prints the generated itemsets sorted by support and the confidence rules sorted by
confidence"""
    print("\n------------------------ Top Item Sets ------------------------")
    for item, support in sorted(items, key=lambda x: x[1])[:10]:      # added[:10] after
 sorted() to reduce the output
        print("item: %s , %.3f" % (str(item), support))
    print("\n------------------------ Top RULES ------------------------")
    for rule, confidence in sorted(rules, key=lambda x: x[1])[:10]:   # added[:10] after
 sorted() to reduce the output
        pre, post = rule
        print("Rule: %s ==> %s , %.3f" % (str(pre), str(post), confidence))
```

In [ ]:

```
# Reads data from a file

def dataFromFile(fname):
    """Function which reads from the file and yields a generator"""
    with open(fname, "r") as file_iter:
        for line in file_iter:
            line = line.strip().rstrip(",")  # Remove trailing comma
            record = frozenset(line.split(","))
            yield record
```

2.3 Implementing the Algorithm

In [ ]:

```
# Load the data from file

inFile = dataFromFile("HungarianNewsItems_indexedforARM.csv")
```

In [ ]:

```
# Find the top item sets and rules

items, rules = runApriori(inFile, minSupport, minConfidence)
```

In [ ]:

```
# Print the rules

printResults(items, rules)
```

```
------------------------ Top Item Sets ------------------------
item: ('59', '67', '3') , 0.501
item: ('59', '67', '3', '34') , 0.501
item: ('59', '67', '85', '3') , 0.501
item: ('59', '67', '3', '86') , 0.501
item: ('59', '67', '3', '90') , 0.501
item: ('59', '76', '67', '3') , 0.501
item: ('59', '85', '3', '67', '90') , 0.501
item: ('59', '85', '3', '34', '67') , 0.501
item: ('59', '76', '85', '3', '67') , 0.501
item: ('59', '3', '34', '67', '90') , 0.501

------------------------ Top RULES ------------------------
Rule: ('34',) ==> ('59', '67', '3') , 0.501
Rule: ('85',) ==> ('59', '67', '3') , 0.501
Rule: ('86',) ==> ('59', '67', '3') , 0.501
Rule: ('90',) ==> ('59', '67', '3') , 0.501
Rule: ('85',) ==> ('59', '67', '90', '3') , 0.501
Rule: ('90',) ==> ('59', '67', '85', '3') , 0.501
Rule: ('90', '85') ==> ('59', '67', '3') , 0.501
Rule: ('85',) ==> ('59', '67', '3', '34') , 0.501
Rule: ('34',) ==> ('59', '67', '85', '3') , 0.501
Rule: ('85', '34') ==> ('59', '67', '3') , 0.501
```

In [ ]:

```
print(f"We have a total of {len(rules)} strong rules whose support and confidence matches
our threshold conditions.")
```

```
We have a total of 2256722 strong rules whose support and confidence matches
our threshold conditions.
```

3. References

I have referred to the following blogs and codes while implementing the above code.

- [https://pypi.org/project/apyori/ (https://pypi.org/project/apyori/)](https://pypi.org/project/apyori/)
- [https://zaxrosenberg.com/unofficial-apyori-documentation/ (https://zaxrosenberg.com/unofficial-apyori-documentation/)](https://zaxrosenberg.com/unofficial-apyori-documentation/)
- Agrawal, Rakesh, and Ramakrishnan Srikant. "Fast algorithms for mining association rules." Proc. 20th int. conf. very large data bases, VLDB. Vol. 1215. 1994.

Prevent random disconnects

This cell runs JS code to automatic reconnect to runtime.

This is to be used when this notebook is being run in a Google Colab Environment

In [ ]:

```
import IPython
from google.colab import output

display(IPython.display.Javascript('''
 function ClickConnect(){
   btn = document.querySelector("colab-connect-button")
   if (btn != null){
     console.log("Click colab-connect-button");
     btn.click()
     }

   btn = document.getElementById('ok')
   if (btn != null){
     console.log("Click reconnect");
     btn.click()
     }
  }

setInterval(ClickConnect,60000)
'''))

print("Done.")
```

```
Done.
```