Indian Institute of Technology Delhi

MTL782 Data Mining: Assignment

Deepak: 2019MT10685

Aakash Goel: 2019MT10668

Question 1

In this part, we have performed multi class classification using various classification techniques such as Decision Tree, Random Forest, Naive Bayes Classifier and KNN Classifier to classify the bitcoin transaction into different types of Ransomware or white(non-ransomware).

Data Description

We have used the Bitcoin Heist Ransomware Address Dataset Data Set from UCL Machine Learning Repository. This dataset contains the entire Bitcoin transaction graph (using a time interval of 24 hour and network edges greater than 80.3) from 2009 January to 2018 December. The dataset has a total of 29,16,697 instances and 10 attributes: address(String, Bitcoin address), year(Integer.Year), day(Integer,Day of the year: 1 is the first day, 365 is the last day), length(Integer), weight(Float), count(Integer), looped(Integer), neighbors(Integer), income(Integer. Satoshi amount), label(String, Name of the ransomware family (e.g., Cryptxxx, cryptolocker etc) or white (i.e., not known to be ransomware)).

Benefits we hope to get from Data Mining

- We hope to analyse this dataset and find patterns that may allow us to flag certain transactions as being ransomware and classify it into different ransomware families.
- This may aid law enforcement authorities to detect fraud and prevent cyber crime.
- Apply the data mining techniques for computing the best trained model with less computation cost and high testing accuracy.

Problems with the data

- The dataset has several redundant features which we need to drop of before training otherwise it would cause over fitting of model.
- Names and string data is not good for modelling purposes and needs to encoded for better efficiency.
- Attributes in the data are measured at different scales and therefore they may not contribute equally to the model fitting model learned function and this might end up creating a bias.

Appropriate response to quality Issues

- We have done feature selection to remove the redundant features(contain no information that is useful for the data mining task at hand) such as the attribute address in the dataset.
- We have used label encoding to handle the catagorical features.
- We have used Feature scaling (min max) to overcome the issue of variables being measued at different scales

Pre-processing the Data

We have applied several pre-processing techniques before feeding the data into the classifier

- **Handling missing values or duplicates :** The data did not contain any missing values or duplicates.
- **Normalisation and Feature Scaling :** It is essential to normalise or scale the data before feeding it into the classifier because without scaling, large values can introduce bias in the results. Thus the features are usually scaled down to the range [-1,1] or [0,1]. We used the normalise method from the preprocessing library of sklearn and applied MinMax scaling on top of it.Thus, the range off feature values was first brought down to [-1,1] and then to [0,1]
- **Label Encoding :** Inorder to handle the categorical features, we have used label encoding technique to convert them to integral values using the label encoder of sklearn preprocessing.
- **Feature Subset selection :** Since the address attribute was irrelevant to the type of Ransomware and including it would have caused overfitting of the model. Therefore, we have dropped this feature
- **K-fold Cross validation :** We have used the k-fold cross validation technique to split our data into training and test data and computed the model accuracy.

Decision Tree Classifier

Decision Tree is a Supervised Machine Learning Algorithm that uses a set of rules to make decisions. On using decision tree classifier to classify the bitcoin transactions, the training error accuracy was much greater as compared to testing accuracy. Therefore, overfitting has occurred.
Training Accuracy = 99.96 %
Testing Accuracy = 93.53 %

Random Forest Classifier

Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classifying Bitcoin transactions, optimal tuned random forest classifier has a maximum depth of 3 nodes and 1 random state
Training Accuracy = 98.51 %
Testing Accuracy = 98.53 %

Naive Bayes Classifier

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. We have used the Multinomial Naive Bayes classifier to classify the bitcoin transactions.
Training Accuracy = 98.58 %
Testing Accuracy = 98.58 %

KNN Classifier

KNN is a type of classification technique where the function is only approximated locally using the information obtained from its nearest neighbours. For classifying Bitcoin transactions,

optimal tuned KNN classifier works with 3 nearest neighbours.
Training Accuracy = 97.28 %
Testing Accuracy = 96.16 %

Question 2

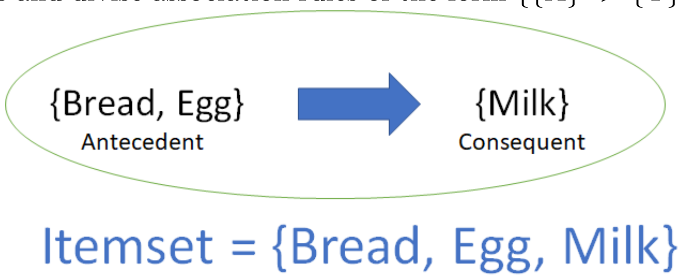
In the second question, we look for patterns in a transactional database, such as retail shopping statistics, to find correlation between different items.

Association Rule Mining

Association Rule Mining is one of the ways to find patterns in data. It finds:

- features (dimensions) which occur together
- features (dimensions) which are "correlated"

Apriori algorithm and FP-Growth algorithm are two of the methods to mine frequent itemsets and divise association rules of the form {{X} -> {Y}} from a given transactional database.



The measures of effectiveness of the rule are:

- Support : This measure gives an idea of how frequent an itemset is in all the transactions.

$$Support(\{X\} \rightarrow \{Y\}) = \frac{Transactions\ containing\ both\ X\ and\ Y}{Total\ number\ of\ transactions}$$

- Confidence : This measure defines the likeliness of occurrence of consequent on the cart given that the cart already has the antecedents.

$$Confidence(\{X\} \rightarrow \{Y\}) = \frac{Transactions\ containing\ both\ X\ and\ Y}{Transactions\ containing\ X}$$

- Lift : This is the rise in probability of having {Y} on the cart with the knowledge of {X} being present over the probability of having {Y} on the cart without any knowledge about presence of {X}.

$$Lift(\{X\} \rightarrow \{Y\}) = \frac{(Transactions\ containing\ both\ X\ and\ Y)/(Transactions\ containing\ X)}{Fraction\ of\ transactions\ containing\ Y}$$

Data Description

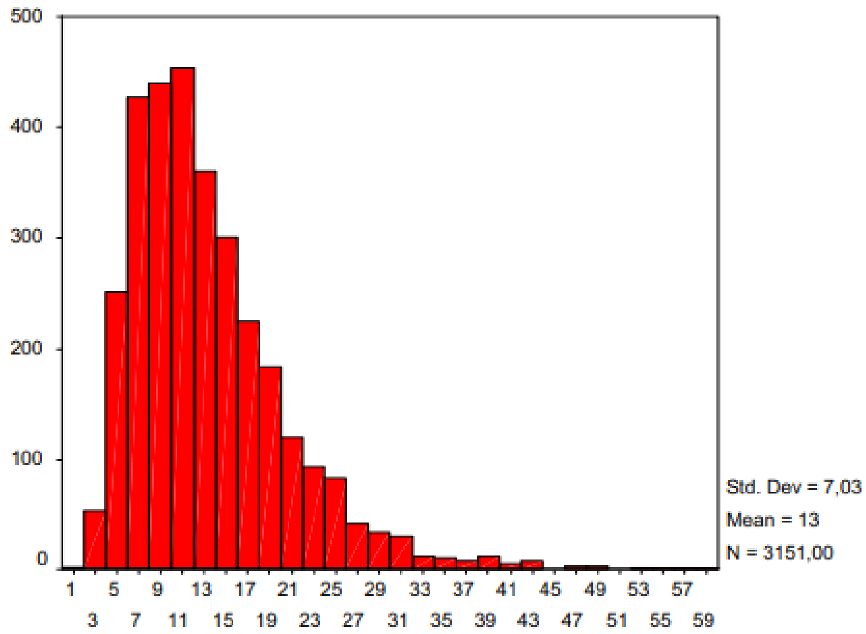
The association models are trained using the dataset from http://fimi.uantwerpen.be/data/retail.dat, which contains the (anonymized) retail market basket data from an anonymous Belgian retail store.

The data are provided 'as is'.

The dataset contains approximately 5 months of data. The total amount of receipts being collected equals 88,163.

Figure 1 shows the average number of distinct items purchased per shopping visit. The average number of distinct items (i.e. different products) purchased per shopping visit equals 13 and most customers buy between 7 and 11 items per shopping visit.

Figure 1: Average number of distinct items purchased per visit

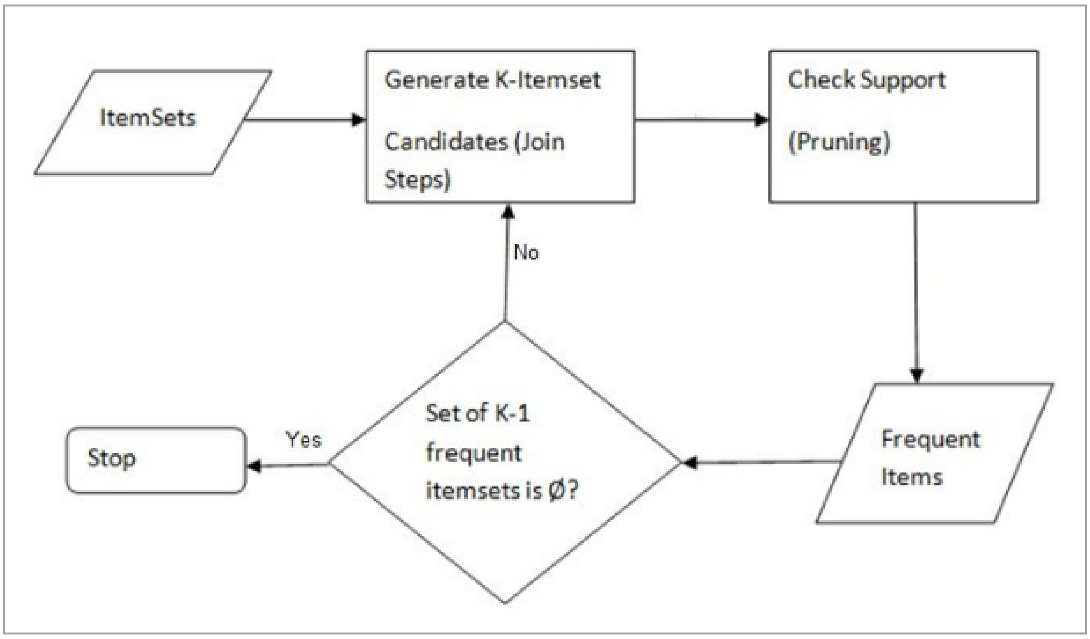


The data has already been pre-processed into labelled records, and can be fed directly to the frequency rule generator.

Apriori

The following are the main steps of the apriori algorithm:

- Calculate the support of item sets (of size k = 1) in the transactional database (note that support is the frequency of occurrence of an itemset). This is called generating the candidate set.
- Prune the candidate set by eliminating items with a support less than the given threshold.
- Join the frequent itemsets to form sets of size k + 1, and repeat the above sets until no more itemsets can be formed. This will happen when the set(s) formed have a support less than the given support.



Modifications

- Transaction reduction: Traditional Apriori algorithm involves the generation of many candidate item sets consisting of many infrequent and unnecessary item sets, and a large number of combinations. The algorithm was modified to disregard itemsets with frequency less than a threshold using minimum support and minimum confidence criteria, and only most frequent itemsets were considered for further rule generation, greatly increasing the efficiency of the algorithm.

- Sampling: The algorithm when run on randomly selected sample of the entire database gave similar results, as itemsets frequent in the entire database are likely to be frequent in the sample as well.

FP Growth

Frequent Pattern Tree is a tree-like structure that is made with the initial itemsets of the database. The purpose of the FP tree is to mine the most frequent pattern. Each node of the FP tree represents an item of the itemset.

The root node represents null while the lower nodes represent the itemsets. The association of the nodes with the lower nodes that is the itemsets with the other itemsets are maintained while forming the tree.

Running the algorithm

The functions apriori and fpgrowth each takes in the dataset and thresholds for support and confidence as parameters and returns the most frequent itemsets along with the association rules derived from those itemsets.

For hyperparameters min_support and min_confidence as 0.01 and 0.2 respectively, frequency sets having upto 5 unique items were discovered satisfying the bounds of support and confidence.

FP Growth was found to be much faster (by upto 20 times for the given dataset) even after making modifications to the apriori algorithm. This is most likely because the construction of fp-tree requires only one pass through the database as compared to multiple scans for apriori and is thus more efficient.

```
Scan  1
```

```
Size of freq set:  78
Scan  2
Size of freq set:  80
Scan  3
Size of freq set:  46
Scan  4
Size of freq set:  12
Scan  5
Size of freq set:  1
Scan  6
Size of freq set:  0
```

Question 1 Code (Jupyter Notebook):

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
from sklearn import tree
from sklearn.metrics import accuracy_score
import copy
```

Loading Dataset

In [2]:

```
input_data=pd.read_csv("data/BitcoinHeistData.csv")
```

In [3]:

```
print(input_data.head())
```

```
                              address  year  day  length    weight  count  \
0   111K8kZAEnJg245r2cM6y9zgJGHZtJPy6  2017   11      18  0.008333      1
1  1123pJv8jzeFQaCV4w644pzQJzVWay2zcA  2016  132      44  0.000244      1
2  112536im7hy6wtKbpH1qYDWtTyMRAcA2p7  2016  246       0  1.000000      1
3  1126eDRw2wqSkWosjTCre8cjjQW8sSeWH7  2016  322      72  0.003906      1
4  1129TSjKtx65E35GiUo4AYVeyo48twbrGX  2016  238     144  0.072848    456

   looped  neighbors       income            label
0       0          2  100050000.0  princetonCerber
1       0          1  100000000.0   princetonLocky
2       0          2  200000000.0  princetonCerber
3       0          2   71200000.0  princetonCerber
4       0          1  200000000.0   princetonLocky
```

Preprocessing

Checking for Duplicates

In [4]:

```
print('No of duplicates in the Input Data:',sum(input_data.duplicated()))
```

```
No of duplicates in the Input Data: 0
```

Checking for NaN/null values

In [5]:

```
print('No of NaN/Null values in Input Data:',input_data.isnull().values.sum())
```

```
No of NaN/Null values in Input Data: 0
```

Data Prepration

In [6]:

```
X = input_data.drop(['label'], axis = 1)
Y = input_data['label']

print(X.head())
```

```
                              address  year  day  length    weight  count  \
0   111K8kZAEnJg245r2cM6y9zgJGHZtJPy6  2017   11      18  0.008333      1
1  1123pJv8jzeFQaCV4w644pzQJzVWay2zcA  2016  132      44  0.000244      1
2  112536im7hy6wtKbpH1qYDWtTyMRAcA2p7  2016  246       0  1.000000      1
3  1126eDRw2wqSkWosjTCre8cjjQW8sSeWH7  2016  322      72  0.003906      1
4  1129TSjKtx65E35GiUo4AYVeyo48twbrGX  2016  238     144  0.072848    456

   looped  neighbors       income
0       0          2  100050000.0
1       0          1  100000000.0
2       0          2  200000000.0
3       0          2   71200000.0
4       0          1  200000000.0
```

Feature Subset Selection

In [7]:

```
e type of Ransomware and including it in X will cause overfitting of the model. Therefore, 
)
```

```
   year  day  length    weight  count  looped  neighbors       income
0  2017   11      18  0.008333      1       0          2  100050000.0
1  2016  132      44  0.000244      1       0          1  100000000.0
2  2016  246       0  1.000000      1       0          2  200000000.0
3  2016  322      72  0.003906      1       0          2   71200000.0
4  2016  238     144  0.072848    456       0          1  200000000.0
```

Label Encoding

In [8]:

```
# Transforming non-numerical value in Y to numerical value using label Encoder
le = preprocessing.LabelEncoder()
le.fit(Y)
Y = le.transform(Y)
print(le.classes_)
```

```
['montrealAPT' 'montrealComradeCircle' 'montrealCryptConsole'
 'montrealCryptXXX' 'montrealCryptoLocker' 'montrealCryptoTorLocker2015'
 'montrealDMALocker' 'montrealDMALockerv3' 'montrealEDA2' 'montrealFlyper'
 'montrealGlobe' 'montrealGlobeImposter' 'montrealGlobev3'
 'montrealJigSaw' 'montrealNoobCrypt' 'montrealRazy' 'montrealSam'
 'montrealSamSam' 'montrealVenusLocker' 'montrealWannaCry'
 'montrealXLocker' 'montrealXLockerv5.0' 'montrealXTPLocker'
 'paduaCryptoWall' 'paduaJigsaw' 'paduaKeRanger' 'princetonCerber'
 'princetonLocky' 'white']
```

Normalising the data

In [9]:

```
X_n = preprocessing.normalize(X)
```

Feature Scaling

In [10]:

```
# MinMaxScalar
from sklearn.preprocessing import MinMaxScaler
scaler1 = MinMaxScaler().fit(X_n)
X_mm = scaler1.transform(X_n)

# Standard Scaler
# from sklearn.preprocessing import StandardScaler
# scaler2 = StandardScaler().fit(X_n)
# X_st = scaler2.transform(X_n)
```

Training the model

Decision Tree

In [11]:

```
k=5
n=len(X_mm)//k
train_accuracy_scores=[]
test_accuracy_scores=[]
# Using K-fold cross validation
for i in range(k):
    X_dummy=copy.deepcopy(X_mm)
    Y_dummy=copy.deepcopy(Y)
    #Train-test split

    X_test=X_dummy[n*i:n*(i+1)]
    Y_test=Y_dummy[n*i:n*(i+1)]
    X_train=[]
    Y_train=[]
    if i==0:
        X_train=X_dummy[n:]
        Y_train=Y_dummy[n:]
    else:
        X_t=X_dummy[0:n*i]
        Y_t=Y_dummy[0:n*i]
        X_tt=X_dummy[n*(i+1):]
        Y_tt=Y_dummy[n*(i+1):]
        X_train=np.concatenate((X_t,X_tt))
        Y_train=np.concatenate((Y_t,Y_tt))

    # model training
    clf_D = tree.DecisionTreeClassifier()
    clf_D = clf_D.fit(X_train, Y_train)

    #Accuracy calculation
    Y_train_pred = clf_D.predict(X_train)
    Y_test_pred = clf_D.predict(X_test)
    train_accuracy_scores.append(accuracy_score(Y_train, Y_train_pred))
    test_accuracy_scores.append(accuracy_score(Y_test, Y_test_pred))
print('-------------------------------------------------------------------------------------
print('Accuracy Score on Training Data:',np.mean(train_accuracy_scores))
print('\n\n----------------------------------------------------------------------------------
print('Accuracy Score on Test Data:',np.mean(test_accuracy_scores))
print('\n------------------------------------------------------------------------------------
```

```
----------------------------------------------------------------------------
------------------------

Accuracy Score on Training Data: 0.9996392323852575


----------------------------------------------------------------------------
--------------------

Accuracy Score on Test Data: 0.9353830277077309

----------------------------------------------------------------------------
--------------------
```

Random Forest

In [12]:

```
from sklearn.ensemble import RandomForestClassifier
k=5
n=len(X_mm)//k
max_depths=[2,3,4,5]
random_states=[0,1]
for depth in max_depths:
    for state in random_states:
        print('\n\n\n------------------------------------------------------------------------
        print('-------------------------------------------------------------------------
        print('----------When max_depth =',depth,' and random state =',state,' ---------')
        train_accuracy_scores=[]
        test_accuracy_scores=[]
        # Using K-fold cross validation
        for i in range(k):
            X_dummy=copy.deepcopy(X_mm)
            Y_dummy=copy.deepcopy(Y)
            #Train-test split

            X_test=X_dummy[n*i:n*(i+1)]
            Y_test=Y_dummy[n*i:n*(i+1)]
            X_train
            Y_train
            if i==0:
                X_train=X_dummy[n:]
                Y_train=Y_dummy[n:]
            else:
                X_t=X_dummy[0:n*i]
                Y_t=Y_dummy[0:n*i]
                X_tt=X_dummy[n*(i+1):]
                Y_tt=Y_dummy[n*(i+1):]
                X_train=np.concatenate((X_t,X_tt))
                Y_train=np.concatenate((Y_t,Y_tt))

            # model training
            clf_R = RandomForestClassifier(max_depth=depth, random_state=state)
            clf_R = clf_R.fit(X_train, Y_train)

            #Accuracy calculation
            Y_train_pred = clf_R.predict(X_train)
            Y_test_pred = clf_R.predict(X_test)
            train_accuracy_scores.append(accuracy_score(Y_train, Y_train_pred))
            test_accuracy_scores.append(accuracy_score(Y_test, Y_test_pred))
        print('Accuracy Score on Training Data:',np.mean(train_accuracy_scores))
        print('Accuracy Score on Test Data:',np.mean(test_accuracy_scores))
```

```
----------------------------------------------------------------------------
------------------------

----------------------------------------------------------------------------
------------------------


----------When max_depth = 2  and random state = 0  ---------
Accuracy Score on Training Data: 0.9796468402483162
```

```
Accuracy Score on Test Data: 0.9730125886557665



----------------------------------------------------------------------------
------------------------

----------------------------------------------------------------------------
------------------------


----------When max_depth = 2  and random state = 1  ---------
Accuracy Score on Training Data: 0.9801899836023971
Accuracy Score on Test Data: 0.9772597068038396



----------------------------------------------------------------------------
------------------------

----------------------------------------------------------------------------
------------------------


----------When max_depth = 3  and random state = 0  ---------
Accuracy Score on Training Data: 0.9824401339759801
Accuracy Score on Test Data: 0.9804234950158263



----------------------------------------------------------------------------
------------------------

----------------------------------------------------------------------------
------------------------


----------When max_depth = 3  and random state = 1  ---------
Accuracy Score on Training Data: 0.9851944298786197
Accuracy Score on Test Data: 0.9853235242918484



----------------------------------------------------------------------------
------------------------

----------------------------------------------------------------------------
------------------------


----------When max_depth = 4  and random state = 0  ---------
Accuracy Score on Training Data: 0.9879082824229665
Accuracy Score on Test Data: 0.9811546719531687



----------------------------------------------------------------------------
------------------------

----------------------------------------------------------------------------
------------------------
```

```
----------When max_depth = 4  and random state = 1  ---------
Accuracy Score on Training Data: 0.989902151745714
Accuracy Score on Test Data: 0.9785577074769386



----------------------------------------------------------------------------
------------------------

----------------------------------------------------------------------------
------------------------


----------When max_depth = 5  and random state = 0  ---------
Accuracy Score on Training Data: 0.9907936810313196
Accuracy Score on Test Data: 0.9770535688778586



----------------------------------------------------------------------------
------------------------

----------------------------------------------------------------------------
------------------------


----------When max_depth = 5  and random state = 1  ---------
Accuracy Score on Training Data: 0.9914179092335152
Accuracy Score on Test Data: 0.9748013999402063
```

Naive Bayes Classifier

In [13]:

```
from sklearn.naive_bayes import MultinomialNB
k=5
n=len(X_mm)//k
train_accuracy_scores=[]
test_accuracy_scores=[]
# Using K-fold cross validation
for i in range(k):
    X_dummy=copy.deepcopy(X_mm)
    Y_dummy=copy.deepcopy(Y)
    #Train-test split

    X_test=X_dummy[n*i:n*(i+1)]
    Y_test=Y_dummy[n*i:n*(i+1)]
    X_train
    Y_train
    if i==0:
        X_train=X_dummy[n:]
        Y_train=Y_dummy[n:]
    else:
        X_t=X_dummy[0:n*i]
        Y_t=Y_dummy[0:n*i]
        X_tt=X_dummy[n*(i+1):]
        Y_tt=Y_dummy[n*(i+1):]
        X_train=np.concatenate((X_t,X_tt))
        Y_train=np.concatenate((Y_t,Y_tt))

    # model training
    clf_N = MultinomialNB()
    clf_N = clf_N.fit(X_train, Y_train)

    #Accuracy calculation
    Y_train_pred = clf_N.predict(X_train)
    Y_test_pred = clf_N.predict(X_test)
    train_accuracy_scores.append(accuracy_score(Y_train, Y_train_pred))
    test_accuracy_scores.append(accuracy_score(Y_test, Y_test_pred))
print('------------------------------------------------------------------------------------------
print('Accuracy Score on Training Data:',np.mean(train_accuracy_scores))
print('\n\n---------------------------------------------------------------------------------------
print('Accuracy Score on Test Data:',np.mean(test_accuracy_scores))
print('\n-----------------------------------------------------------------------------------------
```

```
----------------------------------------------------------------------------
------------------------

Accuracy Score on Training Data: 0.9858014072422663


----------------------------------------------------------------------------
--------------------

Accuracy Score on Test Data: 0.9858013950721622

----------------------------------------------------------------------------
--------------------
```

KNN classifier

In [14]:

```
from sklearn.neighbors import KNeighborsClassifier
k=2
n=len(X_mm)//k
nearest_neighbours=[2,3,4]
for neighbour in nearest_neighbours:
    print('\n\n\n-------------------------------------------------------------------------
    print('--------------------------------------------------------------------------------
    print('----------When we have ',neighbour,' nearest neighbours ---------')
    train_accuracy_scores=[]
    test_accuracy_scores=[]
    # Using K-fold cross validation
    for i in range(k):
        X_dummy=copy.deepcopy(X_mm)
        Y_dummy=copy.deepcopy(Y)
        #Train-test split

        X_test=X_dummy[n*i:n*(i+1)]
        Y_test=Y_dummy[n*i:n*(i+1)]
        X_train
        Y_train
        if i==0:
            X_train=X_dummy[n:]
            Y_train=Y_dummy[n:]
        else:
            X_t=X_dummy[0:n*i]
            Y_t=Y_dummy[0:n*i]
            X_tt=X_dummy[n*(i+1):]
            Y_tt=Y_dummy[n*(i+1):]
            X_train=np.concatenate((X_t,X_tt))
            Y_train=np.concatenate((Y_t,Y_tt))

        # model training
        clf_K = KNeighborsClassifier(n_neighbors=neighbour)
        clf_K = clf_K.fit(X_train, Y_train)

        #Accuracy calculation
        Y_train_pred = clf_K.predict(X_train)
        Y_test_pred = clf_K.predict(X_test)
        train_accuracy_scores.append(accuracy_score(Y_train, Y_train_pred))
        test_accuracy_scores.append(accuracy_score(Y_test, Y_test_pred))
    print('Accuracy Score on Training Data:',np.mean(train_accuracy_scores))
    print('Accuracy Score on Test Data:',np.mean(test_accuracy_scores))
```

```



--------------------------------------------------------------------------
-------------------------

--------------------------------------------------------------------------
-------------------------


----------When we have  2  nearest neighbours ---------
Accuracy Score on Training Data: 0.9632180389416014
Accuracy Score on Test Data: 0.9551857569571339
```

```
-------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------


----------When we have  3  nearest neighbours ---------
Accuracy Score on Training Data: 0.9728231856804006
Accuracy Score on Test Data: 0.9615643830943675



-------------------------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------------------------


----------When we have  4  nearest neighbours ---------
Accuracy Score on Training Data: 0.9733624540701200
Accuracy Score on Test Data: 0.9570979596664613
```

Question 2 Code (Jupyter Notebook):

In [1]:

```
import pandas as pd
import numpy as np
```

In [2]:

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

In [3]:

```
dataset_url = 'http://fimi.uantwerpen.be/data/retail.dat'
data = pd.read_table(dataset_url, header=None)
print(data.head())
```

```
                                                    0
0  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18...
1                                            30 31 32
2                                            33 34 35
3                      36 37 38 39 40 41 42 43 44 45 46
4                                         38 39 47 48
```

In [4]:

```
np_data = data.to_numpy()
itemSetList = []
for record in np_data:
    itemSetList.append(np.fromstring(record[0], dtype=int, sep=" "))
```

In [5]:

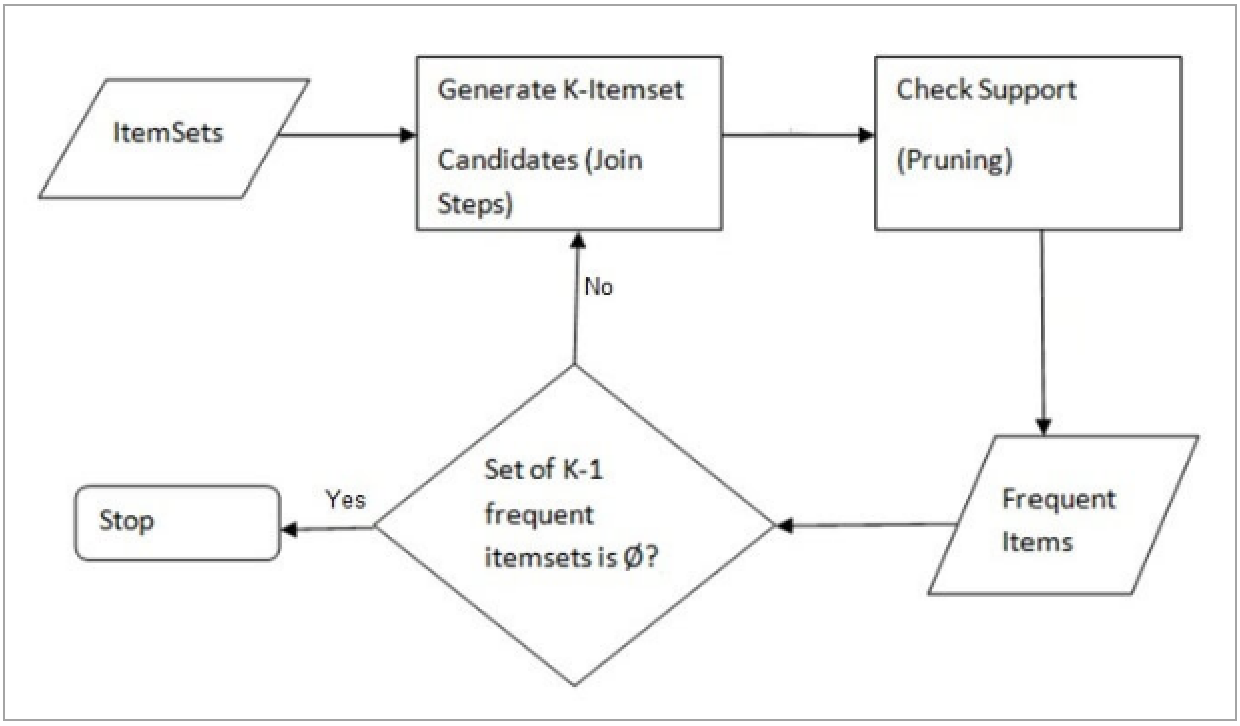
```
print(itemSetList[:5])
print(len(itemSetList))
```

```
[array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29]), array([30, 31,
32]), array([33, 34, 35]), array([36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 4
6]), array([38, 39, 47, 48])]
88162
```

Apriori

The following are the main steps of the apriori algorithm:

- Calculate the support of item sets (of size k = 1) in the transactional database (note that support is the frequency of occurrence of an itemset). This is called generating the candidate set.
- Prune the candidate set by eliminating items with a support less than the given threshold.
- Join the frequent itemsets to form sets of size k + 1, and repeat the above sets until no more itemsets can be formed. This will happen when the set(s) formed have a support less than the given support.



Modifications

- Transaction reduction: Traditional Apriori algorithm involves the generation of many candidate item sets consisting of many infrequent and unnecessary item sets, and a large number of combinations. The algorithm was modified to disregard itemsets with frequency less than a threshold using minimum support and minimum confidence criteria, and only most frequent itemsets were considered for further rule generation, greatly increasing the efficiency of the algorithm.
- Sampling: The algorithm when run on randomly selected sample of the entire database gave similar results, as itemsets frequent in the entire database are likely to be frequent in the sample as well.

In [6]:

```
def apriori(itemSetList, minSup, minConf):
    itemSet = getItemSetFromList(itemSetList)
    # Final result, global frequent itemset
    globalFreqItemSet = dict()
    # Storing global itemset with support count
    globalItemSetWithSup = defaultdict(int)

    print('Scan ',1)
    currentLSet = getAboveMinSup(itemSet, itemSetList, minSup, globalItemSetWithSup)
    print('Size of freq set: ', len(currentLSet))
    k = 2

    # Calculating frequent item set
    while(currentLSet):
        # Storing frequent itemset
        globalFreqItemSet[k-1] = currentLSet
        # Self-joining Lk
        candidateSet = getUnion(currentLSet, k)
        # Perform subset testing and remove pruned supersets
        candidateSet = pruning(candidateSet, currentLSet, k-1)
        # Scanning itemSet for counting support
        print('Scan ',k)
        currentLSet = getAboveMinSup(candidateSet, itemSetList, minSup, globalItemSetWithSu
        print('Size of freq set: ', len(currentLSet))
        k += 1

    rules = associationRuleAP(globalFreqItemSet, globalItemSetWithSup, minConf)
    rules.sort(key=lambda x: x[2])

    #print(globalFreqItemSet)

    return globalFreqItemSet, rules
```

In [7]:

```
def getItemSetFromList(itemSetList):
    tempItemSet = set()

    for itemSet in itemSetList:
        for item in itemSet:
            tempItemSet.add(frozenset([item]))

    return tempItemSet
```

In [8]:

```
def getUnion(itemSet, length):
    return set([i.union(j) for i in itemSet for j in itemSet if len(i.union(j)) == length])
```

In [9]:

```
def pruning(candidateSet, prevFreqSet, length):
    tempCandidateSet = candidateSet.copy()
    for item in candidateSet:
        subsets = combinations(item, length)
        for subset in subsets:
            # if the subset is not in previous K-frequent get, then remove the set
            if(frozenset(subset) not in prevFreqSet):
                tempCandidateSet.remove(item)
                break
    return tempCandidateSet
```

In [10]:

```
def getAboveMinSup(itemSet, itemSetList, minSup, globalItemSetWithSup):
    freqItemSet = set()
    localItemSetWithSup = defaultdict(int)

    for item in itemSet:
        for itemSet in itemSetList:
            if item.issubset(itemSet):
                globalItemSetWithSup[item] += 1
                localItemSetWithSup[item] += 1

    for item, supCount in localItemSetWithSup.items():
        support = float(supCount / len(itemSetList))
        if(support >= minSup):
            freqItemSet.add(item)

    return freqItemSet
```

In [11]:

```
def associationRuleAP(freqItemSet, itemSetWithSup, minConf):
    rules = []
    for k, itemSet in freqItemSet.items():
        for item in itemSet:
            subsets = powerset(item)
            for s in subsets:
                confidence = float(
                    itemSetWithSup[item] / itemSetWithSup[frozenset(s)])
                if(confidence > minConf):
                    rules.append([set(s), set(item.difference(s)), confidence])
    return rules
```

In [12]:

```
def powerset(s):
    return chain.from_iterable(combinations(s, r) for r in range(1, len(s)))
```

FP Growth

Frequent Pattern Tree is a tree-like structure that is made with the initial itemsets of the database. The purpose of the FP tree is to mine the most frequent pattern. Each node of the FP tree represents an item of the itemset.

The root node represents null while the lower nodes represent the itemsets. The association of the nodes with the lower nodes that is the itemsets with the other itemsets are maintained while forming the tree.

In [13]:

```
def fpgrowth(itemSetList, minSupRatio, minConf):
    frequency = getFrequencyFromList(itemSetList)
    minSup = len(itemSetList) * minSupRatio
    fpTree, headerTable = constructTree(itemSetList, frequency, minSup)
    if(fpTree == None):
        print('No frequent item set')
    else:
        freqItems = []
        mineTree(headerTable, minSup, set(), freqItems)
        rules = associationRuleFP(freqItems, itemSetList, minConf)
        return freqItems, rules
```

In [14]:

```
class Node:
    def __init__(self, itemName, frequency, parentNode):
        self.itemName = itemName
        self.count = frequency
        self.parent = parentNode
        self.children = {}
        self.next = None

    def increment(self, frequency):
        self.count += frequency

    def display(self, ind=1):
        print('  ' * ind, self.itemName, ' ', self.count)
        for child in list(self.children.values()):
            child.display(ind+1)
```

In [15]:

```
def constructTree(itemSetList, frequency, minSup):
    headerTable = defaultdict(int)
    # Counting frequency and create header table
    for idx, itemSet in enumerate(itemSetList):
        for item in itemSet:
            headerTable[item] += frequency[idx]

    # Deleting items below minSup
    headerTable = dict((item, sup) for item, sup in headerTable.items() if sup >= minSup)
    if(len(headerTable) == 0):
        return None, None

    # HeaderTable column [Item: [frequency, headNode]]
    for item in headerTable:
        headerTable[item] = [headerTable[item], None]

    # Init Null head node
    fpTree = Node('Null', 1, None)
    # Update FP tree for each cleaned and sorted itemSet
    for idx, itemSet in enumerate(itemSetList):
        itemSet = [item for item in itemSet if item in headerTable]
        itemSet.sort(key=lambda item: headerTable[item][0], reverse=True)
        # Traverse from root to leaf, update tree with given item
        currentNode = fpTree
        for item in itemSet:
            currentNode = updateTree(item, currentNode, headerTable, frequency[idx])

    return fpTree, headerTable
```

In [16]:

```
def updateHeaderTable(item, targetNode, headerTable):
    if(headerTable[item][1] == None):
        headerTable[item][1] = targetNode
    else:
        currentNode = headerTable[item][1]
        # Traverse to the last node then link it to the target
        while currentNode.next != None:
            currentNode = currentNode.next
        currentNode.next = targetNode

def updateTree(item, treeNode, headerTable, frequency):
    if item in treeNode.children:
        # If the item already exists, increment the count
        treeNode.children[item].increment(frequency)
    else:
        # Create a new branch
        newItemNode = Node(item, frequency, treeNode)
        treeNode.children[item] = newItemNode
        # Link the new branch to header table
        updateHeaderTable(item, newItemNode, headerTable)

    return treeNode.children[item]
```

In [17]:

```
def ascendFPtree(node, prefixPath):
    if node.parent != None:
        prefixPath.append(node.itemName)
        ascendFPtree(node.parent, prefixPath)

def findPrefixPath(basePat, headerTable):
    # First node in linked list
    treeNode = headerTable[basePat][1]
    condPats = []
    frequency = []
    while treeNode != None:
        prefixPath = []
        # From leaf node all the way to root
        ascendFPtree(treeNode, prefixPath)
        if len(prefixPath) > 1:
            # Storing the prefix path and it's corresponding count
            condPats.append(prefixPath[1:])
            frequency.append(treeNode.count)

        # Go to next node
        treeNode = treeNode.next
    return condPats, frequency

def mineTree(headerTable, minSup, preFix, freqItemList):
    # Sort the items with frequency and create a list
    sortedItemList = [item[0] for item in sorted(list(headerTable.items()), key=lambda p:p[
    # Start with the lowest frequency
    for item in sortedItemList:
        # Pattern growth is achieved by the concatenation of suffix pattern with frequent p
        newFreqSet = preFix.copy()
        newFreqSet.add(item)
        freqItemList.append(newFreqSet)
        # Find all prefix path, constrcut conditional pattern base
        conditionalPattBase, frequency = findPrefixPath(item, headerTable)
        # Construct conditonal FP Tree with conditional pattern base
        conditionalTree, newHeaderTable = constructTree(conditionalPattBase, frequency, min
        if newHeaderTable != None:
            # Mining recursively on the tree
            mineTree(newHeaderTable, minSup,
                       newFreqSet, freqItemList)
```

In [18]:

```
def getSupport(testSet, itemSetList):
    count = 0
    for itemSet in itemSetList:
        if(set(testSet).issubset(itemSet)):
            count += 1
    return count

def getFrequencyFromList(itemSetList):
    frequency = [1 for i in range(len(itemSetList))]
    return frequency
```

In [19]:

```
def associationRuleFP(freqItemSet, itemSetList, minConf):
    rules = []
    for itemSet in freqItemSet:
        subsets = powerset(itemSet)
        itemSetSup = getSupport(itemSet, itemSetList)
        for s in subsets:
            confidence = float(itemSetSup / getSupport(s, itemSetList))
            if(confidence > minConf):
                rules.append([set(s), set(itemSet.difference(s)), confidence])
    return rules
```

Test

The functions apriori and fpgrowth each takes in the dataset and thresholds for support and confidence as parameters and returns the most frequent itemsets along with the association rules derived from those itemsets.

In [20]:

```
(globalFreqItemSetAP, rulesAP) = apriori(itemSetList,0.01,0.2)
```

```
Scan  1
Size of freq set:  70
Scan  2
Size of freq set:  58
Scan  3
Size of freq set:  25
Scan  4
Size of freq set:  6
Scan  5
Size of freq set:  0
```

In [21]:

```
(globalFreqItemSetFP, rulesFP) = fpgrowth(itemSetList,0.01,0.2)
```

In [22]:

```
print(rulesAP)
print(globalFreqItemSetAP)
```

```
[[{48, 38, 39}, {32}, 0.2025565388397247], [{39}, {38}, 0.2041440552540700
6], [{41}, {38, 39}, 0.20414854466376714], [{32, 48}, {38}, 0.2048792631316
903], [{32, 48}, {41, 39}, 0.2048792631316903], [{41, 39}, {32}, 0.2066760119
1519186], [{48, 38}, {32}, 0.20720040281973817], [{48, 39}, {38}, 0.20938851
142680667], [{32}, {41}, 0.2107206435023406], [{41}, {32}, 0.213850786216125
8], [{48}, {41}, 0.2140263438946244], [{32, 39}, {38}, 0.21762270845653459],
[{48, 41}, {38, 39}, 0.22078066090042137], [{65}, {41}, 0.2224955277280858
7], [{48, 41, 39}, {32}, 0.22345913657344557], [{39}, {41}, 0.22523926985693
143], [{48, 41}, {32}, 0.22876469283654913], [{32, 48, 39}, {38}, 0.22880414
661236578], [{38}, {41}, 0.2498717619902539], [{48, 38}, {41, 39}, 0.2506294
058408862], [{48, 39}, {41}, 0.2527623361471416], [{32, 48}, {41}, 0.2567836
694050286], [{41}, {38}, 0.2607561057209769], [{48, 41}, {38}, 0.26325127522
73231], [{41, 39}, {38}, 0.26730331172244615], [{48, 41, 39}, {38}, 0.270295
9543850122], [{32, 39}, {41}, 0.27900650502661145], [{38, 39}, {41}, 0.29492
50845819236], [{48, 38}, {41}, 0.2988418932527694], [{32, 48, 39}, {41}, 0.3
047019622362088], [{48, 38, 39}, {41}, 0.32628646345460505], [{32}, {48, 3
9}, 0.35616799630777346], [{36}, {48, 38, 39}, 0.3678474114441417], [{110},
{48, 38, 39}, 0.3690050107372942], [{110}, {48, 39}, 0.37115246957766646],
[{110, 38}, {48, 39}, 0.378348623853211], [{36}, {48, 39}, 0.380108991825613
1], [{170}, {48, 38, 39}, 0.38496289125524363], [{36, 38}, {48, 39}, 0.38709
67741935484], [{170}, {48, 39}, 0.38915779283639884], [{38}, {48, 39}, 0.391
25416773531674], [{170, 38}, {48, 39}, 0.39359947212141205], [{65}, {48, 3
9}, 0.4018336314847943], [{237}, {48, 39}, 0.4102902374670185], [{101}, {48,
39}, 0.4228877961555655], [{225}, {48, 39}, 0.4298434141848327], [{32, 38},
{48, 39}, 0.4362866219555242], [{36}, {48, 38}, 0.46321525885558584], [{36},
{48}, 0.4822888283378747], [{110}, {48, 38}, 0.48711524695776665], [{36, 3
8}, {48}, 0.4874551971326165], [{41}, {48, 39}, 0.4928738708598193], [{110},
{48}, 0.49391553328561205], [{170}, {48, 38}, 0.4962891255243627], [{38, 11
0}, {48}, 0.49944954128440366], [{170}, {48}, 0.5024201355275896], [{475},
{48, 39}, 0.5039224734656207], [{170, 38}, {48}, 0.5074232926426921], [{38},
{48}, 0.5093613747114645], [{41, 38}, {48, 39}, 0.5109058249935848], [{32, 4
1}, {48, 39}, 0.5150187734668336], [{310}, {48, 39}, 0.5192752505782575],
[{271}, {48}, 0.5205348615090736], [{32}, {48}, 0.5297026438979363], [{36, 3
9}, {48, 38}, 0.5301914580265096], [{225}, {48}, 0.5330058335891925], [{132
7}, {48}, 0.541993281075028], [{36, 39}, {48}, 0.5478645066273933], [{438},
{48}, 0.5501878690284487], [{270}, {48}, 0.5519031141868512], [{89}, {48, 3
9}, 0.5538180870471723], [{237}, {48}, 0.5547493403693932], [{36, 38, 39},
{48}, 0.5552699228791774], [{2238}, {48}, 0.5568513119533528], [{32}, {39},
0.5574602755983386], [{79}, {48}, 0.558125], [{65}, {48}, 0.565518783542039
3], [{39}, {48}, 0.5750764676862358], [{170, 39}, {48, 38}, 0.57940747935891
21], [{32, 38}, {48}, 0.5810095305330039], [{147}, {48}, 0.582349634626194
5], [{170, 39}, {48}, 0.5857212238950947], [{101}, {48}, 0.586052749217702
2], [{110, 39}, {48, 38}, 0.5861284820920978], [{110, 39}, {48}, 0.589539511
0858442], [{38, 39}, {48}, 0.5898501691638472], [{170, 38, 39}, {48}, 0.5908
865775136206], [{110, 38, 39}, {48}, 0.5925287356321839], [{225, 39}, {48},
0.5954912803062526], [{413}, {39}, 0.601063829787234], [{41}, {48}, 0.603412
512546002], [{413}, {48}, 0.6037234042553191], [{41, 38}, {48}, 0.6091865537
59302], [{533}, {39}, 0.6200403496973773], [{110}, {38, 39}, 0.6227630637079
457], [{65}, {39}, 0.623211091234347], [{101}, {39}, 0.6258381761287438],
[{110}, {39}, 0.629563350035791], [{237}, {39}, 0.6362137203166227], [{38, 1
10}, {39}, 0.6385321100917432], [{32, 39}, {48}, 0.6389118864577173], [{14
7}, {39}, 0.6391231028667791], [{12925}, {39}, 0.6394001363326517], [{65, 3
9}, {48}, 0.6447793326157158], [{237, 39}, {48}, 0.6448937273198548], [{41,
39}, {48}, 0.6453478184685474], [{32, 41}, {48}, 0.64549436795995], [{1327},
{39}, 0.6472564389697648], [{32, 38}, {39}, 0.6494881750794211], [{170}, {3
```

```
8, 39}, 0.6515004840271055], [{310}, {48}, 0.6522744795682344], [{41, 38, 3
9}, {48}, 0.652572926909210l], [{475}, {48}, 0.6589755422242732], [{60}, {3
9}, 0.6601746138347885], [{36}, {38, 39}, 0.6624659400544959], [{38}, {39},
0.6633111054116441], [{170}, {39}, 0.6644078735075831], [{170, 38}, {39}, 0.
6661167931375783], [{32, 38, 39}, {48}, 0.6717391304347826], [{32, 48}, {3
9}, 0.6723923325865073], [{101, 39}, {48}, 0.6757142857142857], [{438}, {3
9}, 0.6763285024154589], [{271}, {39}, 0.6848137535816619], [{270}, {39}, 0.
6885813148788927], [{1146}, {39}, 0.6893408134642356], [{48}, {39}, 0.691634
0334638661], [{475}, {39}, 0.6922011998154131], [{36}, {39}, 0.6938010899182
562], [{79}, {39}, 0.694375], [{36, 38}, {39}, 0.6971326164874552], [{32, 4
1, 39}, {48}, 0.6977532852903773], [{48, 65}, {39}, 0.7105575326215896], [{3
10}, {39}, 0.7139552814186585], [{89}, {39}, 0.7164451394318478], [{255}, {3
9}, 0.7170963364993216], [{255}, {48}, 0.7170963364993216], [{48, 101}, {3
9}, 0.7215865751334859], [{225}, {39}, 0.7218299048203869], [{310, 39}, {4
8}, 0.7273218142548596], [{475, 39}, {48}, 0.728], [{89}, {48}, 0.7292155329
68465], [{32, 41}, {39}, 0.7381101376720901], [{48, 237}, {39}, 0.7395957193
816884], [{48, 110}, {38, 39}, 0.7471014492753624], [{2238}, {39}, 0.7504373
177842566], [{32, 48, 38}, {39}, 0.7509113001215066], [{48, 110}, {39}, 0.75
14492753623189], [{48, 110, 38}, {39}, 0.7575312270389419], [{48, 89}, {39},
0.7594710507505361], [{48, 36}, {38, 39}, 0.7627118644067796], [{41}, {39},
0.7637336901973905], [{48, 475}, {39}, 0.7647058823529411], [{48, 170}, {38,
39}, 0.7662170841361593], [{48, 38}, {39}, 0.7681268882175226], [{89, 39},
{48}, 0.7730083666787922], [{48, 170}, {39}, 0.7745664739884393], [{48, 170,
38}, {39}, 0.7756827048114434], [{41, 38}, {39}, 0.7829099307159353], [{48,
36}, {39}, 0.788135593220339], [{48, 36, 38}, {39}, 0.7941176470588235], [{4
8, 310}, {39}, 0.7960992907801419], [{32, 41, 48}, {39}, 0.797867183713039
2], [{48, 225}, {39}, 0.8064516129032258], [{48, 41}, {39}, 0.81681082279884
68], [{48, 41, 38}, {39}, 0.8386689132266217], [{286}, {38}, 0.9433643279797
126], [{36}, {38}, 0.9502724795640327], [{36, 39}, {38}, 0.954835542464408
5], [{48, 36}, {38}, 0.96045197740113], [{48, 36, 39}, {38}, 0.9677419354838
71], [{37}, {38}, 0.9739292364990689], [{110}, {38}, 0.9753042233357194],
[{170}, {38}, 0.9780574378831881], [{170, 39}, {38}, 0.9805730937348227],
[{48, 110}, {38}, 0.986231884057971], [{48, 170}, {38}, 0.9877970456005138],
[{110, 39}, {38}, 0.9891984081864695], [{48, 170, 39}, {38}, 0.9892205638474
295], [{48, 110, 39}, {38}, 0.9942140790742526]]
{1: {frozenset({32}), frozenset({264}), frozenset({49}), frozenset({170}), f
rozenset({677}), frozenset({956}), frozenset({89}), frozenset({237}), frozen
set({78}), frozenset({36}), frozenset({13041}), frozenset({79}), frozenset
({117}), frozenset({258}), frozenset({147}), frozenset({242}), frozenset({27
0}), frozenset({15832}), frozenset({225}), frozenset({548}), frozenset({139
3}), frozenset({161}), frozenset({1327}), frozenset({39}), frozenset({338}),
frozenset({179}), frozenset({271}), frozenset({1004}), frozenset({45}), froz
enset({1146}), frozenset({438}), frozenset({175}), frozenset({31}), frozense
t({41}), frozenset({16217}), frozenset({101}), frozenset({2238}), frozenset
({522}), frozenset({48}), frozenset({589}), frozenset({9}), frozenset({37}),
frozenset({3270}), frozenset({479}), frozenset({2958}), frozenset({824}), fr
ozenset({592}), frozenset({60}), frozenset({286}), frozenset({475}), frozens
et({14098}), frozenset({185}), frozenset({413}), frozenset({38}), frozenset
({310}), frozenset({110}), frozenset({301}), frozenset({783}), frozenset({24
9}), frozenset({10515}), frozenset({201}), frozenset({65}), frozenset({25
5}), frozenset({123}), frozenset({740}), frozenset({19}), frozenset({1292
5}), frozenset({16010}), frozenset({533}), frozenset({604})}, 2: {frozenset
({48, 413}), frozenset({48, 89}), frozenset({170, 39}), frozenset({48, 27
1}), frozenset({48, 237}), frozenset({48, 36}), frozenset({237, 39}), frozen
set({41, 38}), frozenset({225, 39}), frozenset({255, 39}), frozenset({39, 7
9}), frozenset({12925, 39}), frozenset({170, 38}), frozenset({271, 39}), fro
zenset({48, 147}), frozenset({38, 286}), frozenset({32, 48}), frozenset({32,
39}), frozenset({48, 1327}), frozenset({48, 38}), frozenset({48, 79}), froze
nset({310, 39}), frozenset({1146, 39}), frozenset({438, 39}), frozenset({14
7, 39}), frozenset({48, 438}), frozenset({101, 39}), frozenset({41, 39}), fr
ozenset({48, 101}), frozenset({48, 41}), frozenset({60, 39}), frozenset({48,
```

```
39}), frozenset({48, 475}), frozenset({39, 1327}), frozenset({475, 39}), fro
zenset({48, 310}), frozenset({38, 39}), frozenset({110, 38}), frozenset({37,
38}), frozenset({413, 39}), frozenset({36, 38}), frozenset({32, 41}), frozen
set({48, 65}), frozenset({2238, 39}), frozenset({89, 39}), frozenset({48, 22
38}), frozenset({48, 170}), frozenset({65, 39}), frozenset({270, 39}), froze
nset({36, 39}), frozenset({48, 270}), frozenset({110, 39}), frozenset({48, 1
10}), frozenset({533, 39}), frozenset({32, 38}), frozenset({65, 41}), frozen
set({48, 225}), frozenset({48, 255})}, 3: {frozenset({170, 38, 39}), frozens
et({48, 475, 39}), frozenset({32, 41, 48}), frozenset({48, 36, 38}), frozens
et({48, 225, 39}), frozenset({48, 65, 39}), frozenset({48, 170, 39}), frozen
set({48, 89, 39}), frozenset({48, 101, 39}), frozenset({48, 41, 39}), frozen
set({32, 48, 38}), frozenset({32, 38, 39}), frozenset({41, 38, 39}), frozens
et({48, 41, 38}), frozenset({48, 38, 39}), frozenset({48, 310, 39}), frozens
et({32, 48, 39}), frozenset({48, 170, 38}), frozenset({48, 36, 39}), frozens
et({36, 38, 39}), frozenset({32, 41, 39}), frozenset({110, 38, 39}), frozens
et({48, 110, 38}), frozenset({48, 110, 39}), frozenset({48, 237, 39})}, 4:
{frozenset({48, 38, 39, 41}), frozenset({48, 38, 39, 170}), frozenset({48, 3
6, 38, 39}), frozenset({32, 48, 39, 41}), frozenset({32, 48, 38, 39}), froze
nset({48, 38, 39, 110})}}
```

In [23]:

```
print(rulesFP)
print(globalFreqItemSetFP)
```

```
[[{37}, {38}, 0.9739292364990689], [{286}, {38}, 0.9433643279797126], [{114
6}, {39}, 0.6893408134642356], [{12925}, {39}, 0.6394001363326517], [{255},
{48}, 0.7170963364993216], [{255}, {39}, 0.7170963364993216], [{533}, {39},
0.6200403496973773], [{60}, {39}, 0.6601746138347885], [{79}, {48}, 0.55812
5], [{79}, {39}, 0.694375], [{2238}, {48}, 0.5568513119533528], [{2238}, {3
9}, 0.7504373177842566], [{270}, {48}, 0.5519031141868512], [{270}, {39}, 0.
6885813148788927], [{147}, {48}, 0.5823496346261945], [{147}, {39}, 0.639123
1028667791], [{1327}, {48}, 0.541993281075028], [{1327}, {39}, 0.64725643896
97648], [{438}, {48}, 0.5501878690284487], [{438}, {39}, 0.676328502415458
9], [{413}, {39}, 0.601063829787234], [{413}, {48}, 0.6037234042553191], [{2
71}, {48}, 0.5205348615090736], [{271}, {39}, 0.6848137535816619], [{475},
{48}, 0.6589755422242732], [{475}, {48, 39}, 0.5039224734656207], [{48, 47
5}, {39}, 0.7647058823529411], [{475, 39}, {48}, 0.728], [{475}, {39}, 0.692
2011998154131], [{101}, {48}, 0.5860527492177022], [{101}, {48, 39}, 0.42288
77961555655], [{48, 101}, {39}, 0.7215865751334859], [{101, 39}, {48}, 0.675
7142857142857], [{101}, {39}, 0.6258381761287438], [{310}, {48}, 0.652274479
5682344], [{310}, {48, 39}, 0.5192752505782575], [{48, 310}, {39}, 0.7960992
907801419], [{310, 39}, {48}, 0.7273218142548596], [{310}, {39}, 0.713955281
4186585], [{110}, {48}, 0.49391553328561205], [{110}, {48, 39}, 0.3711524695
7766646], [{48, 110}, {39}, 0.7514492753623189], [{110, 39}, {48}, 0.5895395
110858442], [{110}, {48, 38, 39}, 0.3690050107372942], [{48, 110}, {38, 39},
0.7471014492753624], [{110, 38}, {48, 39}, 0.378348623853211], [{110, 39},
{48, 38}, 0.5861284820920978], [{48, 110, 38}, {39}, 0.7575312270389419],
[{48, 110, 39}, {38}, 0.9942140790742526], [{110, 38, 39}, {48}, 0.592528735
6321839], [{110}, {48, 38}, 0.48711524695776665], [{48, 110}, {38}, 0.986231
884057971], [{110, 38}, {48}, 0.49944954128440366], [{110}, {39}, 0.62956335
0035791], [{110}, {38, 39}, 0.6227630637079457], [{110, 38}, {39}, 0.6385321
100917432], [{110, 39}, {38}, 0.9891984081864695], [{110}, {38}, 0.975304223
3357194], [{36}, {48}, 0.4822888283378747], [{36}, {48, 39}, 0.3801089918256
131], [{48, 36}, {39}, 0.788135593220339], [{36, 39}, {48}, 0.54786450662739
33], [{36}, {48, 38, 39}, 0.3678474114441417], [{48, 36}, {38, 39}, 0.762711
8644067796], [{36, 38}, {48, 39}, 0.3870967741935484], [{36, 39}, {48, 38},
0.5301914580265096], [{48, 36, 38}, {39}, 0.7941176470588235], [{48, 36, 3
9}, {38}, 0.967741935483871], [{36, 38, 39}, {48}, 0.5552699228791774], [{3
6}, {48, 38}, 0.46321525885558584], [{48, 36}, {38}, 0.96045197740113], [{3
6, 38}, {48}, 0.4874551971326165], [{36}, {39}, 0.6938010899182562], [{36},
{38, 39}, 0.6624659400544959], [{36, 38}, {39}, 0.6971326164874552], [{36, 3
9}, {38}, 0.9548355424644085], [{36}, {38}, 0.9502724795640327], [{237}, {4
8}, 0.5547493403693932], [{237}, {48, 39}, 0.4102902374670185], [{48, 237},
{39}, 0.7395957193816884], [{237, 39}, {48}, 0.6448937273198548], [{237}, {3
9}, 0.6362137203166227], [{170}, {48}, 0.5024201355275896], [{170}, {48, 3
9}, 0.38915779283639884], [{48, 170}, {39}, 0.7745664739884393], [{170, 39},
{48}, 0.5857212238950947], [{170}, {48, 38, 39}, 0.38496289125524363], [{48,
170}, {38, 39}, 0.7662170841361593], [{170, 38}, {48, 39}, 0.393599472121412
05], [{170, 39}, {48, 38}, 0.5794074793589121], [{48, 170, 38}, {39}, 0.7756
827048114434], [{48, 170, 39}, {38}, 0.9892205638474295], [{170, 38, 39}, {4
8}, 0.5908865775136206], [{170}, {48, 38}, 0.4962891255243627], [{48, 170},
{38}, 0.9877970456005138], [{170, 38}, {48}, 0.5074232926426921], [{170}, {3
9}, 0.6644078735075831], [{170}, {38, 39}, 0.6515004840271055], [{170, 38},
{39}, 0.6661167931375783], [{170, 39}, {38}, 0.9805730937348227], [{170}, {3
8}, 0.9780574378831881], [{225}, {48}, 0.5330058335891925], [{225}, {48, 3
9}, 0.4298434141848327], [{48, 225}, {39}, 0.8064516129032258], [{225, 39},
{48}, 0.5954912803062526], [{225}, {39}, 0.7218299048203869], [{89}, {39},
0.7164451394318478], [{89}, {48, 39}, 0.5538180870471723], [{48, 89}, {39},
0.7594710507505361], [{89, 39}, {48}, 0.7730083666787922], [{89}, {48}, 0.72
```

```
9215532968465], [{65}, {41}, 0.22249552772808587], [{65}, {48}, 0.5655187835
420393], [{65}, {48, 39}, 0.4018336314847943], [{48, 65}, {39}, 0.7105575326
215896], [{65, 39}, {48}, 0.6447793326157158], [{65}, {39}, 0.62321109123434
7], [{32}, {41}, 0.2107206435023406], [{41}, {32}, 0.2138507862161258], [{3
2, 41}, {48}, 0.64549436795995], [{32, 48}, {41}, 0.2567836694050286], [{48,
41}, {32}, 0.22876469283654913], [{32, 41}, {48, 39}, 0.5150187734668336],
[{32, 48}, {41, 39}, 0.2048792631316903], [{32, 41, 48}, {39}, 0.79786718371
30392], [{32, 41, 39}, {48}, 0.6977532852903773], [{32, 48, 39}, {41}, 0.304
7019622362088], [{48, 41, 39}, {32}, 0.22345913657344557], [{32, 41}, {39},
0.7381101376720901], [{32, 39}, {41}, 0.27900650502661145], [{41, 39}, {32},
0.20667601191519186], [{41}, {38}, 0.2607561057209769], [{38}, {41}, 0.24987
17619902539], [{48, 41}, {38}, 0.2632512752273231], [{48, 38}, {41}, 0.29884
18932527694], [{41, 38}, {48}, 0.609186553759302], [{48, 41}, {38, 39}, 0.22
078066090042137], [{48, 38}, {41, 39}, 0.2506294058408862], [{41, 38}, {48,
39}, 0.5109058249935848], [{48, 41, 38}, {39}, 0.8386689132266217], [{48, 4
1, 39}, {38}, 0.2702959543850122], [{48, 38, 39}, {41}, 0.3262864634546050
5], [{41, 38, 39}, {48}, 0.6525729269092101], [{41}, {38, 39}, 0.20414854466
376714], [{41, 38}, {39}, 0.7829099307159353], [{41, 39}, {38}, 0.2673033117
2244615], [{38, 39}, {41}, 0.2949250845819236], [{48}, {41}, 0.2140263438946
244], [{41}, {48}, 0.603412512546002], [{41}, {48, 39}, 0.4928738708598193],
[{48, 41}, {39}, 0.8168108227988468], [{48, 39}, {41}, 0.2527623361471416],
[{41, 39}, {48}, 0.6453478184685474], [{41}, {39}, 0.7637336901973905], [{3
9}, {41}, 0.22523926985693143], [{32, 48}, {38}, 0.2048792631316903], [{32,
38}, {48}, 0.5810095305330039], [{48, 38}, {32}, 0.20720040281973817], [{32,
38}, {48, 39}, 0.4362866219555242], [{32, 48, 38}, {39}, 0.750911300121506
6], [{32, 48, 39}, {38}, 0.22880414661236578], [{32, 38, 39}, {48}, 0.671739
1304347826], [{48, 38, 39}, {32}, 0.2025565388397247], [{32, 38}, {39}, 0.64
94881750794211], [{32, 39}, {38}, 0.21762270845653459], [{32}, {48}, 0.52970
26438979363], [{32}, {48, 39}, 0.35616799630777346], [{32, 48}, {39}, 0.6723
923325865073], [{32, 39}, {48}, 0.6389118864577173], [{32}, {39}, 0.55746027
55983386], [{38}, {48}, 0.5093613747114645], [{38}, {48, 39}, 0.391254167735
31674], [{48, 38}, {39}, 0.7681268882175226], [{48, 39}, {38}, 0.20938851142
680667], [{38, 39}, {48}, 0.5898501691638472], [{38}, {39}, 0.66331110541164
41], [{39}, {38}, 0.20414405525407006], [{48}, {39}, 0.6916340334638661],
[{39}, {48}, 0.5750764676862358]]
[{10515}, {264}, {2958}, {45}, {242}, {956}, {31}, {479}, {3270}, {783}, {17
5}, {522}, {258}, {179}, {19}, {161}, {117}, {13041}, {78}, {37}, {37, 38},
{1004}, {677}, {589}, {49}, {201}, {548}, {15832}, {249}, {1393}, {16217},
{740}, {286}, {38, 286}, {301}, {604}, {824}, {592}, {338}, {14098}, {123},
{16010}, {9}, {185}, {1146}, {1146, 39}, {12925}, {12925, 39}, {255}, {48, 2
55}, {39, 255}, {533}, {533, 39}, {60}, {60, 39}, {79}, {48, 79}, {39, 79},
{2238}, {48, 2238}, {2238, 39}, {270}, {48, 270}, {270, 39}, {147}, {48, 14
7}, {147, 39}, {1327}, {48, 1327}, {39, 1327}, {438}, {48, 438}, {438, 39},
{413}, {413, 39}, {48, 413}, {271}, {48, 271}, {39, 271}, {475}, {48, 475},
{48, 475, 39}, {475, 39}, {101}, {48, 101}, {48, 101, 39}, {101, 39}, {310},
{48, 310}, {48, 310, 39}, {310, 39}, {110}, {48, 110}, {48, 110, 39}, {48, 3
8, 110, 39}, {48, 38, 110}, {110, 39}, {38, 110, 39}, {38, 110}, {36}, {48,
36}, {48, 36, 39}, {48, 36, 38, 39}, {48, 36, 38}, {36, 39}, {36, 38, 39},
{36, 38}, {237}, {48, 237}, {48, 237, 39}, {237, 39}, {170}, {48, 170}, {48,
170, 39}, {48, 170, 38, 39}, {48, 170, 38}, {170, 39}, {170, 38, 39}, {170,
38}, {225}, {48, 225}, {48, 225, 39}, {225, 39}, {89}, {89, 39}, {48, 89, 3
9}, {48, 89}, {65}, {65, 41}, {48, 65}, {48, 65, 39}, {65, 39}, {41}, {32, 4
1}, {32, 41, 48}, {32, 41, 48, 39}, {32, 41, 39}, {41, 38}, {48, 41, 38}, {4
8, 41, 38, 39}, {41, 38, 39}, {48, 41}, {48, 41, 39}, {41, 39}, {32}, {32, 3
8}, {32, 48, 38}, {32, 48, 38, 39}, {32, 38, 39}, {32, 48}, {32, 48, 39}, {3
2, 39}, {38}, {48, 38}, {48, 38, 39}, {38, 39}, {48}, {48, 39}, {39}]
```

References

- https://towardsdatascience.com/association-rule-mining-be4122fc1793 (https://towardsdatascience.com/association-rule-mining-be4122fc1793)
- https://www.javatpoint.com/apriori-algorithm (https://www.javatpoint.com/apriori-algorithm)
- https://www.ijstr.org/final-print/aug2017/A-Modified-Apriori-Algorithm-For-Fast-And-Accurate-Generation-Of-Frequent-Item-Sets.pdf (https://www.ijstr.org/final-print/aug2017/A-Modified-Apriori-Algorithm-For-Fast-And-Accurate-Generation-Of-Frequent-Item-Sets.pdf)
- https://towardsdatascience.com/the-fp-growth-algorithm-1ffa20e839b8 (https://towardsdatascience.com/the-fp-growth-algorithm-1ffa20e839b8)
- https://towardsdatascience.com/association-rules-2-aa9a77241654 (https://towardsdatascience.com/association-rules-2-aa9a77241654)

In [ ]: