# Introduction To Data Science Semester Project

**Sofiane OUAARI** — NEPTUNE CODE: **DFMYZK**

University Eötvös Lorànd University
`sofiane.ouaari@gmail.com`

**Abstract.** Credits loan is among the core services banks are offering to their various customers depending on the purpose and duration. Banks are competing for the best price-quality ratio to present it to their clients in order to be ahead in the market. However, lending money always hold a certain amount of risk since customers differ from each other in their nature and social status hence some might be bad customers in a way where they are not able to get back the loan, and such a case represents a loss to the bank. This report aims to use Data Science and Machine Learning to help banks make better decisions regarding credits loans.

## 1 Introduction

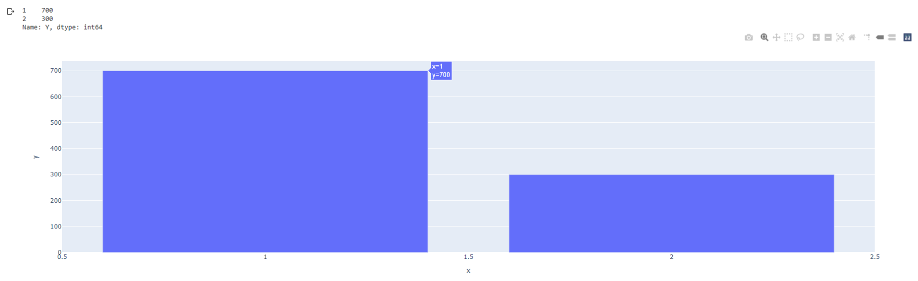
In this report, we will try to analyse past customer data from a bank and we will work on implementing an accurate binary classifier to predict either a customer is good or not for credit loan to allow the bank in the future to , based on the client data, decide whether to give or not a loan. Then we will perform some clustering to identify potential group of customers. In the end, we will work on identifying frequent pattern features present among good customers.

As a first step, we will be performing some analysis on the data by applying some **EDA** along with some data preprocessing to make the data ready to be fitted into a Machine Learning model.

## 2 Exploratory Data Analysis

### 2.1 Dataset

The dataset is composed of **1000** instances and **21** columns from which **20** of them are features in addition to a label column referring to the type of customer ( Good or Bad ).
From the below bar plot we can clearly observe the imbalance in the dataset in which the distribution is of **70% good customers** and **30% bad customers**, such an imbalance needs to be taken into consideration while building and evaluation the Machine Learning models later on in the report.

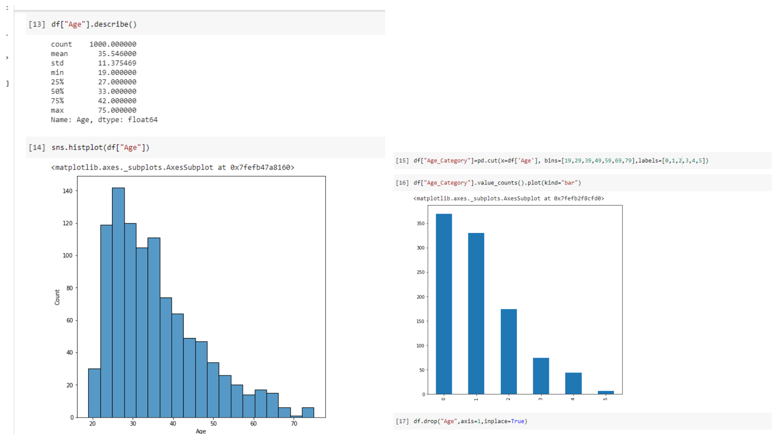


```
1    700
2    300
Name: Y, dtype: int64
```

**Fig. 1.** Dataset Imabalnce

For the sake of readability we will rename the columns into their actual significance.

Also our dataset is clean from null values which is in plenty of times an issue to put into consideration.
Let us learn more about the customers we are analyzing by learning more about the age range and spectrum. From the above distribution plot and from the general overview of the age feature, we see that the age distribution is mostly skewed towards the age range of [**19,30**] with a mean of **35** which is something quite understandable since most of the persons asking for a loan either for a house/car or starting a business is made by people starting their professional life hence young people. For this sake we splitted the age feature into bins following the distribution.



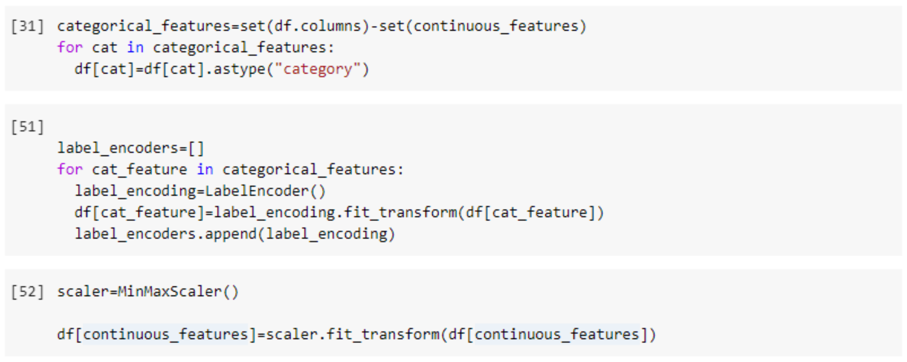
```
[13] df["Age"].describe()

     count    1000.000000
     mean       35.546000
     std        11.375469
     min        19.000000
     25%        27.000000
     50%        33.000000
     75%        42.000000
     max        75.000000
     Name: Age, dtype: float64

[14] sns.histplot(df["Age"])

     <matplotlib.axes._subplots.AxesSubplot at 0x7fefb47a8160>
```

```
[15] df["Age_Category"]=pd.cut(x=df['Age'], bins=[19,29,39,49,59,69,79],labels=[0,1,2,3,4,5])

[16] df["Age_Category"].value_counts().plot(kind="bar")

     <matplotlib.axes._subplots.AxesSubplot at 0x7fefb2f8cfd0>

[17] df.drop("Age",axis=1,inplace=True)
```

**Fig. 2.** Customers' Age Distribution

## 2.2 Data Preprocessing

In order to make our data ready for the classification task, we need to first to preprocess it. For our case we will first label encode the categorical features using the **sklearn's LabelEncoder()** class. In addition, we have normalized



```
[31] categorical_features=set(df.columns)-set(continuous_features)
     for cat in categorical_features:
       df[cat]=df[cat].astype("category")

[51]
     label_encoders=[]
     for cat_feature in categorical_features:
       label_encoding=LabelEncoder()
       df[cat_feature]=label_encoding.fit_transform(df[cat_feature])
       label_encoders.append(label_encoding)

[52] scaler=MinMaxScaler()

     df[continuous_features]=scaler.fit_transform(df[continuous_features])
```

**Fig. 3.** Label Encoding Of Categorical Features

our continuous data namely "Credit Amount" and "Duration of Credit" using the **MinMaxScaler** whose formula is:
$Xnorm = (X - Xmin)/(Xmax - Xmin)$

## 3 Building A Classifier

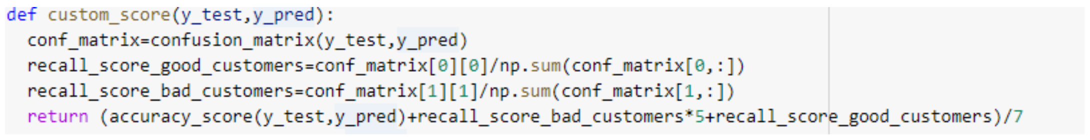
In this section of the report we will focus on building and comparing different machine learning models and fine-tune the best ones for this binary classification task of predicting either a customer is good or bad for credit loan.

### 3.1 Defining The Evaluation Metric

We have previously mentioned that the dataset is imbalanced where the number of good customers is bigger than the bad ones and since it is five times worse to predict a bad customer as a good one since it is riskier for the bank and the probability of losing money will get higher.
In order to take such a detail into consideration we will define a **Custom Evaluation Score** as a metric for the models' evaluation.

$$Custom\_Eval = (Accuracy\_Score + Recall\_Bad\_Cust * 5 + Recall\_Good\_Cust)/7$$

The used metric gives more weight to the recall score for bad customers hence penalizing more the cases where good customers were predicted as good ones. In addition, we added the overall accuracy score and the recall score for good customers to also give importance to the precise predictions related to good clients. Maximizing the recall is equivalent to minimizing the **False Negatives** rate. Since the recall is defined by: $Recall = TP/(TP + FN)$



```
def custom_score(y_test,y_pred):
  conf_matrix=confusion_matrix(y_test,y_pred)
  recall_score_good_customers=conf_matrix[0][0]/np.sum(conf_matrix[0,:])
  recall_score_bad_customers=conf_matrix[1][1]/np.sum(conf_matrix[1,:])
  return (accuracy_score(y_test,y_pred)+recall_score_bad_customers*5+recall_score_good_customers)/7
```

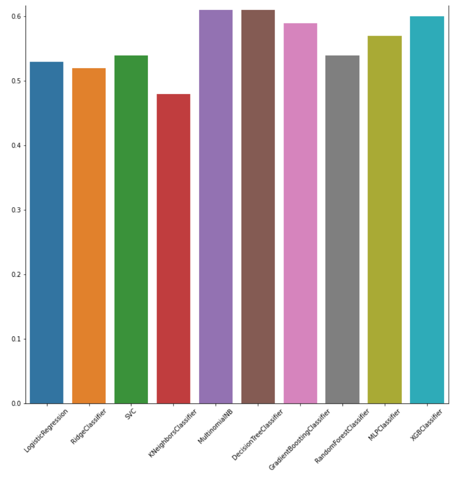
**Fig. 4.** Metric Eval Code Classification

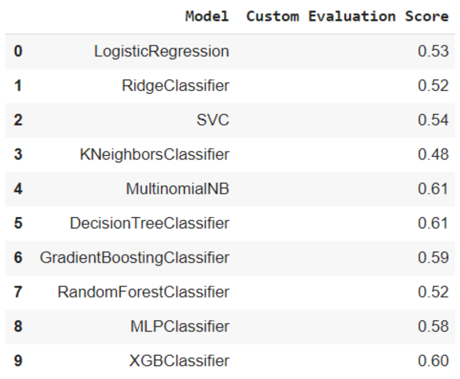
### 3.2 Using the default parameters

We will now build Machine Learning models and evaluate their predictions using the custom metric defined above and then pick the best ones to perform some hyper-parameters tuning.

Those are the models used for this report:
**LogisticRegression - RidgeClassifier -SVC-KNN- Multinomial Naive Bayes - DecisionTreeClassifier - GradientBoostingClassifier - RandomForestClassifier - MLPClassifier - XGBClassifier**

We will compare those machine learning models based on their default parameters and then we will fine-tune the best models who got the highest evaluation scores. From the results shown in the table and bar plot below, we can see that the best models reached scores ranging between **{59%-61%}**.
For the other ones like Logistic Regression and KNN, such score shows that they were directly affected by the imbalance hence resulting to a low recall score for bad customers.



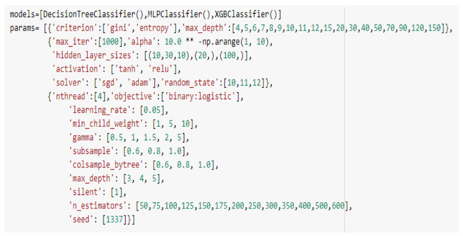


| | Model | Custom Evaluation Score |
|---|---|---|
| 0 | LogisticRegression | 0.53 |
| 1 | RidgeClassifier | 0.52 |
| 2 | SVC | 0.54 |
| 3 | KNeighborsClassifier | 0.48 |
| 4 | MultinomialNB | 0.61 |
| 5 | DecisionTreeClassifier | 0.61 |
| 6 | GradientBoostingClassifier | 0.59 |
| 7 | RandomForestClassifier | 0.52 |
| 8 | MLPClassifier | 0.58 |
| 9 | XGBClassifier | 0.60 |

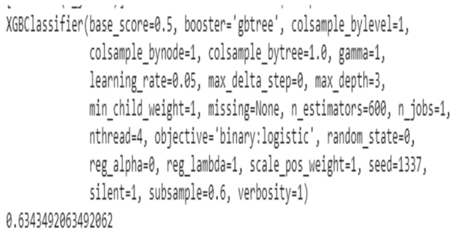
**Fig. 5.** Results of Default Machine Learning Models

## 3.3 Hyperparameter Tuning Using GridSearchCV

We will now perform grid search on 3 different models [**DecisionTreeClassifier - MLPClassifier - XGBClassifier**]



```
models=[DecisionTreeClassifier(),MLPClassifier(),XGBClassifier()]
params= [{'criterion':['gini','entropy'],'max_depth':[4,5,6,7,8,9,10,11,12,15,20,30,40,50,70,90,120,150]},
        {'max_iter':[1000],'alpha': 10.0 ** -np.arange(1, 10),
         'hidden_layer_sizes': [(10,30,10),(20,),(100,)],
         'activation': ['tanh', 'relu'],
         'solver': ['sgd', 'adam'],'random_state':[10,11,12]},
        {'nthread':[4],'objective':['binary:logistic'],
             'learning_rate': [0.05],
             'min_child_weight': [1, 5, 10],
             'gamma': [0.5, 1, 1.5, 2, 5],
             'subsample': [0.6, 0.8, 1.0],
             'colsample_bytree': [0.6, 0.8, 1.0],
             'max_depth': [3, 4, 5],
             'silent': [1],
             'n_estimators': [50,75,100,125,150,175,200,250,300,350,400,500,600],
             'seed': [1337]}]
```



```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1.0, gamma=1,
              learning_rate=0.05, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=600, n_jobs=1,
              nthread=4, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=1337,
              silent=1, subsample=0.6, verbosity=1)
0.6343492063492062
```

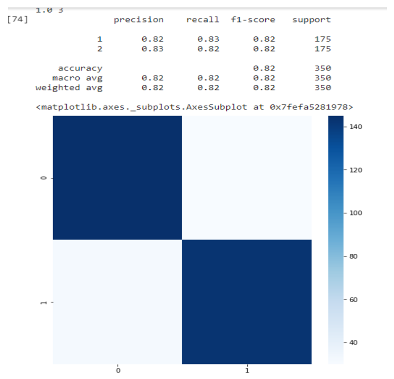
**Fig. 6.** Grid Search

After running the gridsearch, the model which reached the best score of **63%** is a fine-tuned version of XGBoost, it is worth mentioning that we explicitly added the Custom Evaluation Score ( weighted towards recall of bad customers ) within the GridSearch object and this by using sklearn's built-in method **make_scorer**.

### 3.4 Oversampling Model

To overcome the imbalance, we will use an oversampling algorithm called **Synthetic Minority Oversampling Technique -SMOTE-**. This technique consists of generating data points that are within the area between neighboring samples of the minority class.
**SMOTE** first selects a minority class instance at random and finds its k nearest minority class neighbors. The synthetic instance is then created by choosing one of the k nearest neighbors b at random and connecting a and b to form a line segment in the feature space. We will be using **imblearn** which is a python package mainly used for under/over-sampling a given dataset.

We will now train our fine-tuned version of XGBoost on the over-sampled dataset.



```
    1.0 3
[74]               precision    recall  f1-score   support

              1         0.82      0.83      0.82       175
              2         0.83      0.82      0.82       175

       accuracy                             0.82       350
      macro avg         0.82      0.82      0.82       350
   weighted avg         0.82      0.82      0.82       350

<matplotlib.axes._subplots.AxesSubplot at 0x7fefa5281978>
```

**Fig. 7.** Over Sampled Dataset Confusion Matrix

As we can clearly observe the score went from **63%** to **82%** which is a big leap!

### 3.5 Extracting Most Important Features

As a final step, and after reaching such an accuracy we will now know which features are impacting the most our model. XGBoost has this property to sort the features according to their impact on the classification. We have plotted both the accuracy score and the custom evaluation score to check both if the prediction is good for both types of customers as well as on bad customers. We can see that at 8 best features the model reached **80%** in both metrics which is not far from the final score **82%** when using all 20 features.

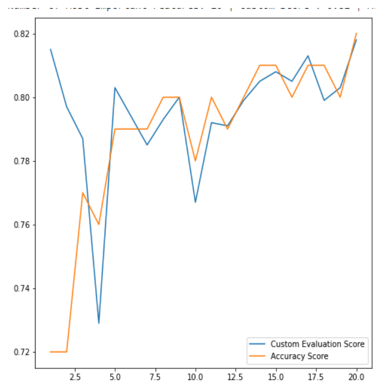


**Fig. 8.** Feature Importance Selection

## 4 Clustering

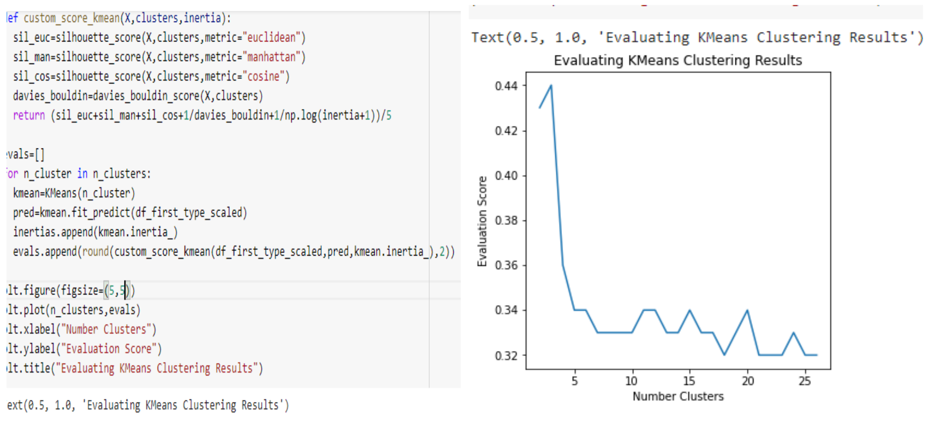
### 4.1 Evaluation Metrics

For this section we will be using an evaluation score, which will gather different type of clustering metrics together.
$Eval\_KMeans = (Sil\_euc+Sil\_manhat+Sil\_cos+1/(davies\_boudin)+1/log(1+inertia))/5$

The logarithm applied on the inertia is mainly for scaling reasons since in our case for example the number of clusters ranged between 2-25 is resulting in an inertia of a range [2000-9000]. We also added the **"+1"** just to mathematically avoid an infinity value in the perfect case where the number of clusters equals the number of samples. In addition we took the inverse of the logarithm of inertia and the value of the davies bouldin score since the smaller they are the better the clustering is. In this section we will try to learn more about the customers taking into consideration their ages, credit duration and amount

### 4.2 KMeans on 3 Features

In this section we will try to learn more about the customers taking into consideration their ages, credit duration and amount. We will try this out with different number of clusters and then we plot the scores according to them.



```
lef custom_score_kmean(X,clusters,inertia):
  sil_euc=silhouette_score(X,clusters,metric="euclidean")
  sil_man=silhouette_score(X,clusters,metric="manhattan")
  sil_cos=silhouette_score(X,clusters,metric="cosine")
  davies_bouldin=davies_bouldin_score(X,clusters)
  return (sil_euc+sil_man+sil_cos+1/davies_bouldin+1/np.log(inertia+1))/5

vals=[]
or n_cluster in n_clusters:
  kmean=KMeans(n_cluster)
  pred=kmean.fit_predict(df_first_type_scaled)
  inertias.append(kmean.inertia_)
  evals.append(round(custom_score_kmean(df_first_type_scaled,pred,kmean.inertia_),2))

lt.figure(figsize=(5,5))
lt.plot(n_clusters,evals)
lt.xlabel("Number Clusters")
lt.ylabel("Evaluation Score")
lt.title("Evaluating KMeans Clustering Results")
```

```
ext(0.5, 1.0, 'Evaluating KMeans Clustering Results')
```

```
Text(0.5, 1.0, 'Evaluating KMeans Clustering Results')
```

**Fig. 9.** Applying KMean on Age-Credit Amount-Duration

As we can observe **3** is the best number of clusters which reached the best score. Let us now visualize the clusters in a **3d-Scatter Plot** using **Plotly**. We can

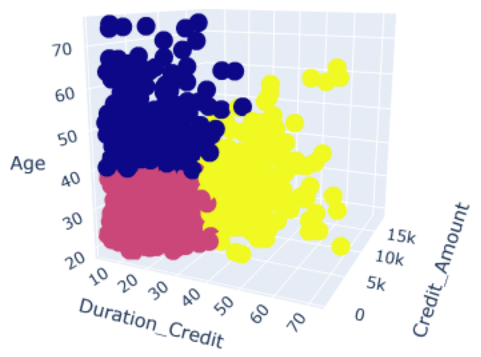


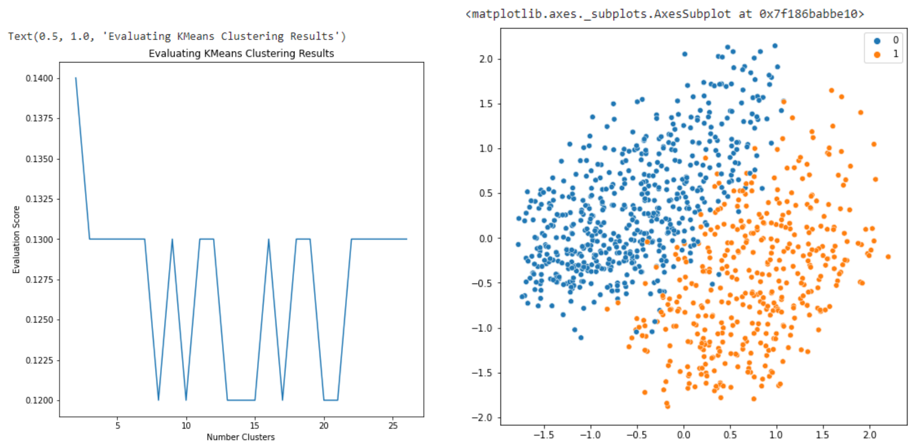
**Fig. 10.** 3D Scatter For Clustering Visualization

interpret this clustering this way:

1. **Red Group**: Youth starting small businesses and small projects which requires a small amount of time to accomplish.
2. **Blue Group**: Mostly composed of people aged more than 40 years old, but with the same criteria as the first group of small loans and small duration.
3. **Yellow Group**: Clients asking for bigger credit amount and taking a longer time. This age range is mostly within the range [20-50].

### 4.3 KMeans on the Whole Dataset

Now we will perform the same steps but this time on a **Hot-Encoded** version of the whole dataset using all the features.
We will then be using **Principal Component Analysis** to reduce the number of dimensions to 2 for visualization reasons. As you can see in **Fig.11**, The



```
Text(0.5, 1.0, 'Evaluating KMeans Clustering Results')
```

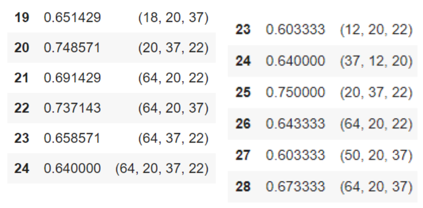
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f186babbe10>
```

**Fig. 11.** KMeans on the whole dataset

number of clusters which gave the best result is **2**. And when visualizing the reduced version of the dataset, we can clearly see the presence of 2 groups.

## 5 Frequent Pattern Mining

As mentioned earlier, all banks care about entrusting money to clients which will become a gain rather than a loss. Hence in this section, we will learn more about the set of features which frequently appear together among the good clients so that their presence might be a signal to label that client as a good customer. We will be using the **apriori** algorithm available in the python library **mlxtend**. We will focus only on the categorical features and turn them into a one-hot encoded vector which is required by the package, in case of the presence of 1s, the given itemset count will be incremented to calculate its support.
And here is the result received for a support threshold of **0.6**.
**The left results are from the good customers while the right ones are for the bad ones.**
As you can see we got a support of **0.64** for the itemset [**Is Not A Foreign Worker, Installment rate of 3, None Other Debtors, Number of People the customer being liable: 1**].
While for the bad customers we can observe that based on a support of **0.6**, the itemset composed of 4 items are not present.



| | | |
|---|---|---|
| 19 | 0.651429 | (18, 20, 37) |
| 20 | 0.748571 | (20, 37, 22) |
| 21 | 0.691429 | (64, 20, 22) |
| 22 | 0.737143 | (64, 20, 37) |
| 23 | 0.658571 | (64, 37, 22) |
| 24 | 0.640000 | (64, 20, 37, 22) |

| | | |
|---|---|---|
| 23 | 0.603333 | (12, 20, 22) |
| 24 | 0.640000 | (37, 12, 20) |
| 25 | 0.750000 | (20, 37, 22) |
| 26 | 0.643333 | (64, 20, 22) |
| 27 | 0.603333 | (50, 20, 37) |
| 28 | 0.673333 | (64, 20, 37) |

**Fig. 12.** Applying Apriori Algorithm (thresh=0.6 ) on Good/Bad Customers' Data

## 6 Conclusion

After trying multiple machine learning models to classify the customers depending on their type (good or bad), and after running/ applying some frequent pattern mining to look for repeated patterns present in the good clients. We conclude that we can adopt a **hybdrid** approach which consists of relying on the classification model in addition to consider those patterns as a layer to weaken or strengthen the prediction output.

## References

1. https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/
2. https://medium.com/swlh/feature-importance-hows-and-why-s-3678ede1e58f
3. http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/