
# Credit Card Fraud Detection by Neural network in Keras Framework

27.03.2020 - Houssam AlRachid

Houssam AlRachid

# Introduction

Due to the theatrical increase of fraud which results in loss of dollars worldwide each year, several modern techniques in detecting fraud are persistently evolved and applied to many business fields. Fraud detection involves monitoring the activities of populations of users in order to estimate, perceive or avoid undesirable behavior. Undesirable behavior is a broad term including delinquency, fraud, intrusion, and account defaulting. Credit Card Fraud is one of the biggest faced issues and the amount of money involved in this is generally enormous.

The major aspect of this project is to develop a best suited algorithm to find the frauds in case of credit cards. We will implement several **Machine Learning and Deep Learning algorithms and compare them and choose the best algorithm**. We will implement algorithms like:

1. Artificial Neural Network;
2. Logistic Regression;
3. K-Means.

# Exploratory Data Analysis

We used an existing online dataset . The dataset contains information about transactions in September 2013 by European cardholders. The dataset consists of around **284,807 records** out of which there are only **492 fraudsters**. So, this shows that the dataset is highly imbalanced as the positive class or frauds are only **0.172%** of all transactions. It contains only numeric input variables which are the result of a **PCA transformation**. Unfortunately, due to confidentiality issues, the original features and more background information about the data cannot be provided. Features columns **V1, V2, V3 … V28** are obtained as a result of **PCA transformation.** Hence their value ranges from **-1 to 1**. The only features which have not been transformed with PCA are **'Time'** and **'Amount'**. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction amount, this feature can be used for example-dependent cost-sensitive learning. Feature **'Class'** column is the classification variable which contains value **0 (Normal Case)** and **1 (Fraud)**.

The dataset can be downloaded for free on Kaggle website:

https://www.kaggle.com/mlg-ulb/creditcardfraud

These transactions occurred in two days::

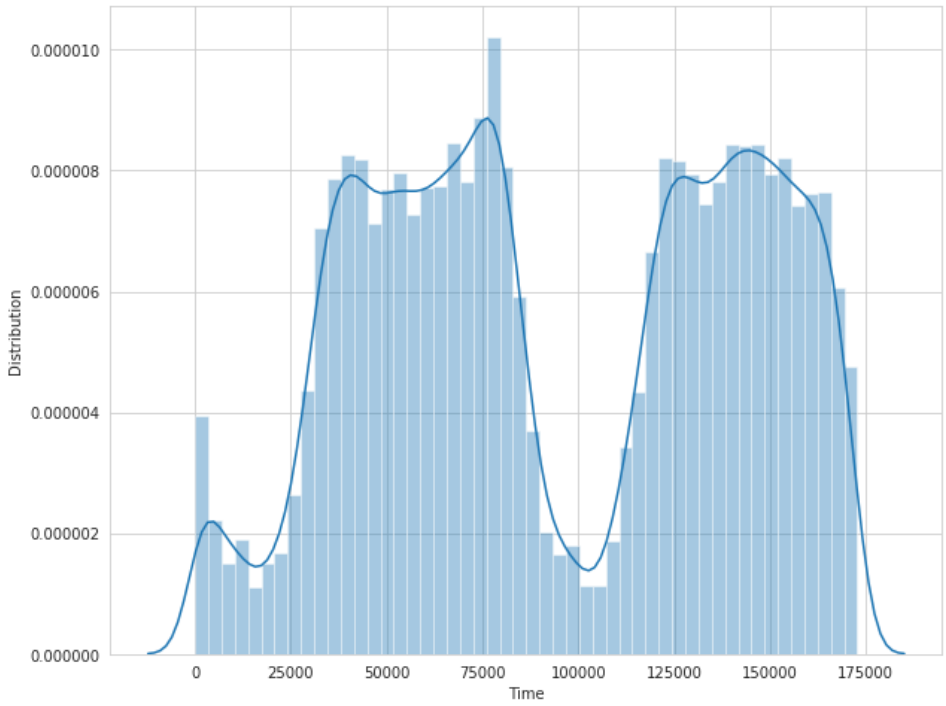
| V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | ... | V21 | V22 | V23 | V24 | V25 | V26 | V27 | V28 | Amount | Class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -1.359807 | -0.072781 | 2.536347 | 1.378155 | -0.338321 | 0.462388 | 0.239599 | 0.098698 | 0.363787 | ... | -0.018307 | 0.277838 | -0.110474 | 0.066928 | 0.128539 | -0.189115 | 0.133558 | -0.021053 | 149.62 | 0 |
| 1.191857 | 0.266151 | 0.166480 | 0.448154 | 0.060018 | -0.082361 | -0.078803 | 0.085102 | -0.255425 | ... | -0.225775 | -0.638672 | 0.101288 | -0.339846 | 0.167170 | 0.125895 | -0.008983 | 0.014724 | 2.69 | 0 |
| -1.358354 | -1.340163 | 1.773209 | 0.379780 | -0.503198 | 1.800499 | 0.791461 | 0.247676 | -1.514654 | ... | 0.247998 | 0.771679 | 0.909412 | -0.689281 | -0.327642 | -0.139097 | -0.055353 | -0.059752 | 378.66 | 0 |
| -0.966272 | -0.185226 | 1.792993 | -0.863291 | -0.010309 | 1.247203 | 0.237609 | 0.377436 | -1.387024 | ... | -0.108300 | 0.005274 | -0.190321 | -1.175575 | 0.647376 | -0.221929 | 0.062723 | 0.061458 | 123.50 | 0 |
| -1.158233 | 0.877737 | 1.548718 | 0.403034 | -0.407193 | 0.095921 | 0.592941 | -0.270533 | 0.817739 | ... | -0.009431 | 0.798278 | -0.137458 | 0.141267 | -0.206010 | 0.502292 | 0.219422 | 0.215153 | 69.99 | 0 |

The data set features can be summarized as follow:

- **V1, V2, ... V28**: principal components obtained with PCA;
- **Time**: seconds elapsed between each transaction and the first transaction in the dataset;
- **Amount**: transaction Amount;
- **Class**: response variable: 1 (fraud) or 0 (normal).

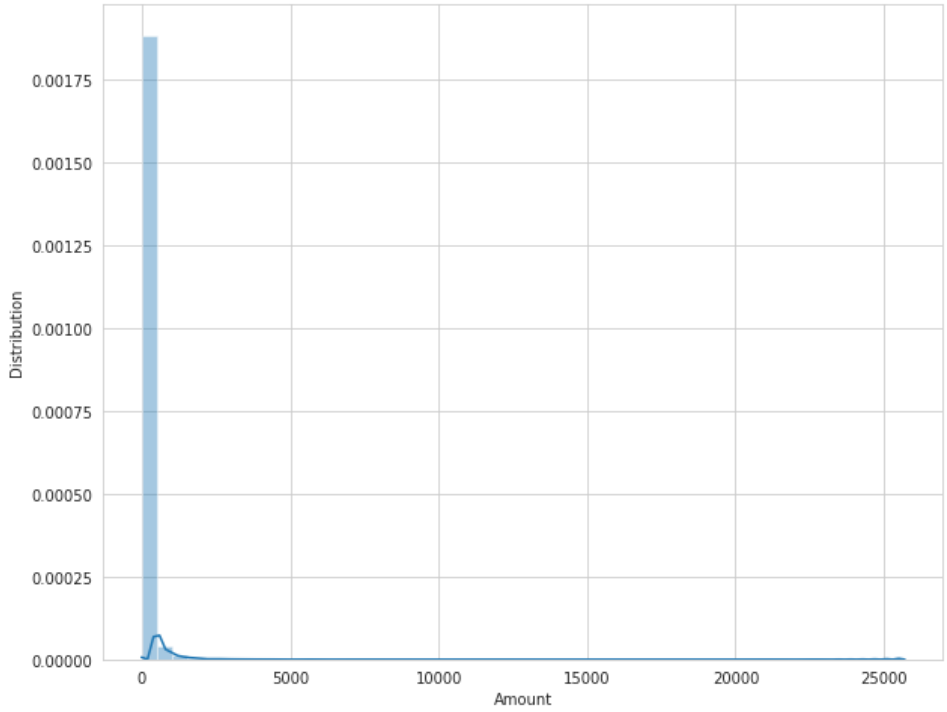
Now that we have the data, let's run a few initial comparisons between Time, Amount, some $V_i$ Class.

1. **Time:**



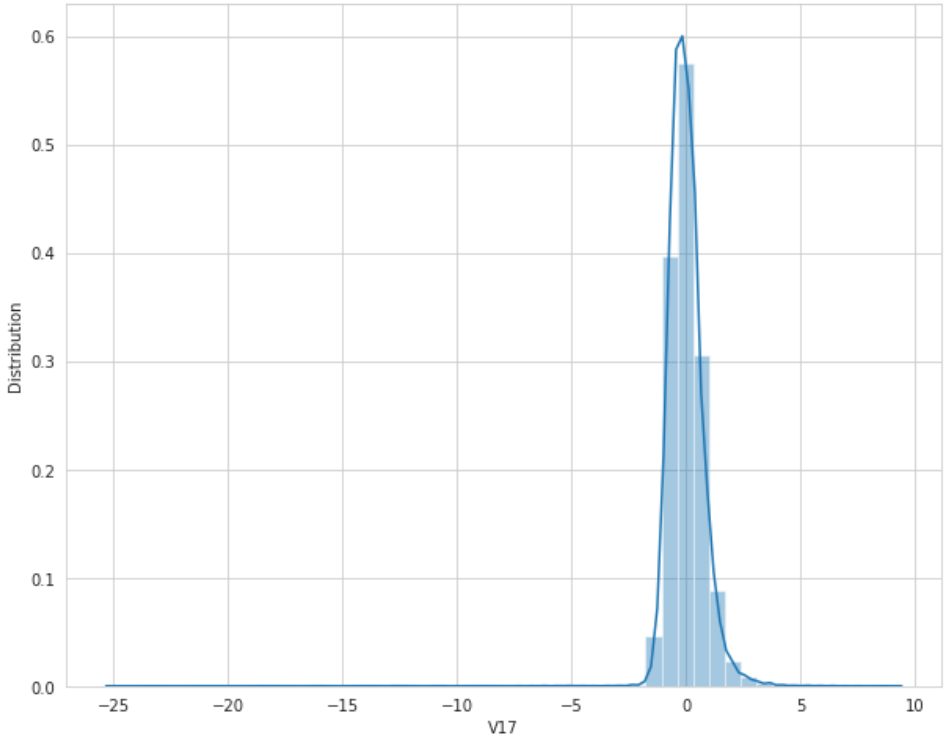

Since this distribution represents two days of data, it would follow the trend I expected to see for normal consumers. Most purchases are made during the day and when people leave work / school and go home, purchases decrease till the next day.

2. **Amount:**



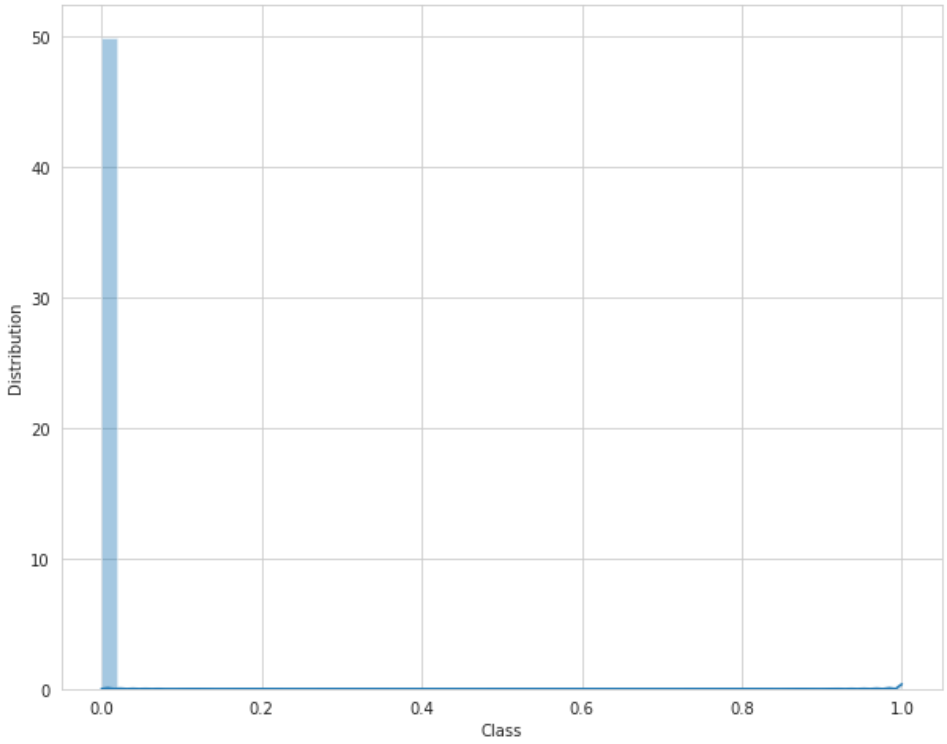

As expected, the vast majority of transactions are very small (most are < 50$), but this is probably where most fraudulent transactions also occur.

**3.** $V_{17}$ **:**



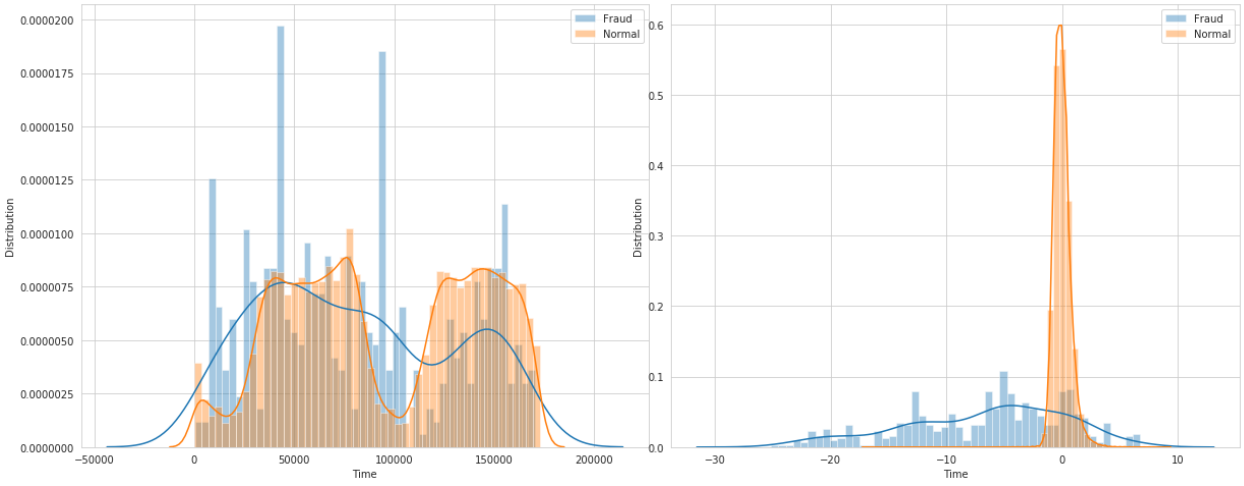

Since $V_{17}$ is a linear combination of some correlated features, we cannot comment on the meaning of this distribution.

## 4. Class:




In this dataset, there are only 492 fraudulent transactions. This only represents 0.173% of all transactions in this dataset, which explain the pace of the histogram.

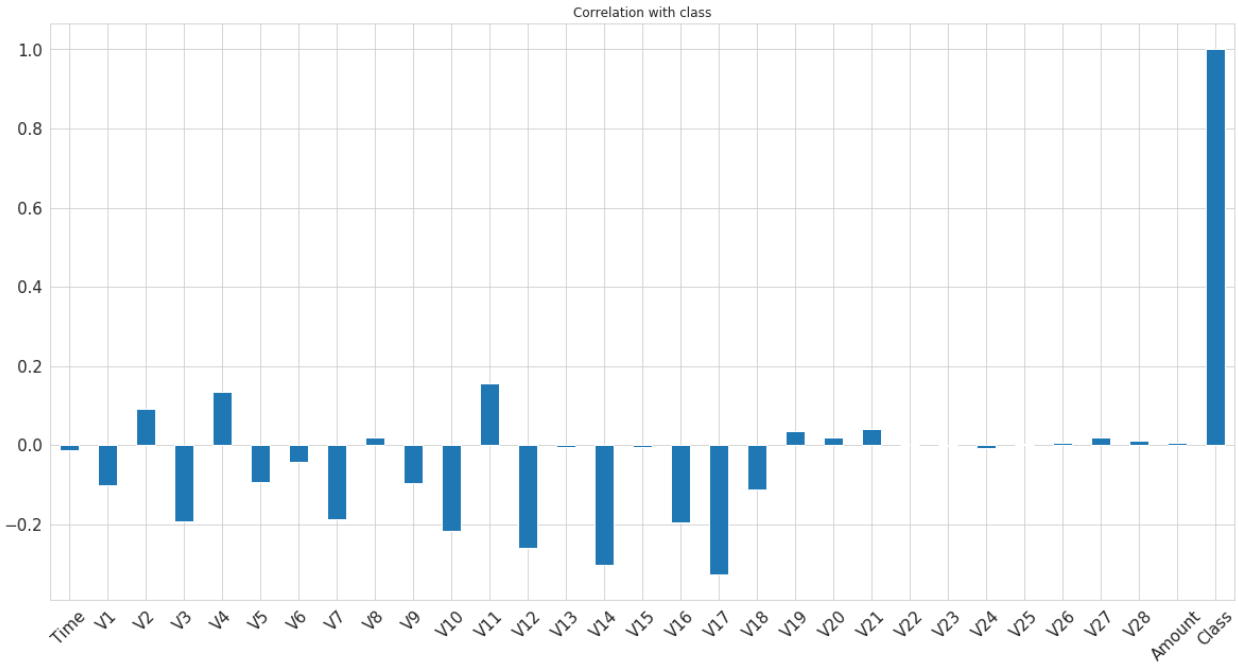
We now illustrate the distributions of Time and $V_{17}$ variables with respect to the Class : 0/1



# Feature Engineering and ETL

## Correlation:

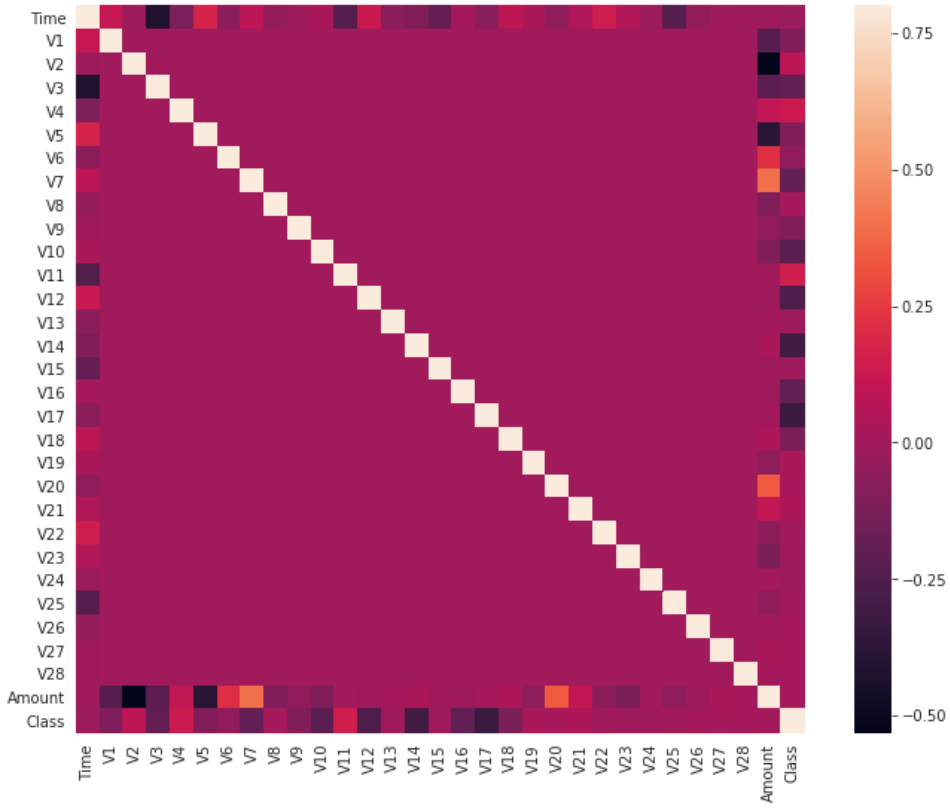
Covariance between each feature and the prediction independent variable can help us predict what are the features that are most relevant for the prediction.




We can see positive/negative correlation and nul correlations.

Similarly, correlation matrices graphically give us an idea of how features correlate with respect to the prediction independent variable and can help us predict what are the features that are most relevant for the prediction.

In the following HeatMap we can clearly see that most of the features do not correlate to other features but there are some features that either have a positive or a negative correlation with each other. For example “V2” and “V5” are highly negatively correlated with the feature called “Amount”. We also see that “V7” and “V20” are positively correlated with “Amount”. This gives us a deeper understanding of the Data available to us.




## Scaling:

The majority of machine learning algorithms behave much better using **standardization**, we center the feature columns at mean 0 with standard deviation 1 so that the feature columns take the form of a normal distribution, which makes it easier to learn the weights. Furthermore, standardization maintains useful information about outliers and makes the algorithm less sensitive to them in contrast to min-max scaling, which scales the data to a limited range of values. Since we are using Neural Network for prediction, it will be helpful to scale the data.

# Methodology

In this present project, we will focus on developing a best suited algorithm to detect fraudulent credit cards transactions. We will implement several Machine Learning and Deep Learning algorithms (prediction/classification/clustering) and compare them and choose the best algorithm.

The methodology can be resumed in the following steps:

1. We have collected and manipulated the database in the best possible way, we then divide it (using Skicit learn) between training and testing (20%) sets;
2. We will implement algorithms like:

   a. Artificial Neural Network;
   b. Logistic Regression;
   c. K-Means.
3. At the end of each algorithm, we evaluate it by computing accuracy using several indicators such as : Score, F1-score, Precision, Recall and Confusion matrix;
4. We Class the algorithms and we choose the most suited one.

# Analysis

## Artificial Neural Network (ANN)

### Building the model

Fraud detection methods based on neural networks are the most popular ones. ANNs are a concept of deep learning that we will implement using Keras framework. ANNs are made up of neurons. The first layer or input layer is the input neuron which includes the transaction and the amount of each client. The hidden layer consists of weight, bias and activation function. We can add as many hidden layers to adjust performance. In this case, we are using **5 layers,** with **16, 24, 20, 24 and 1 neurons** respectively, with a total number of **1,917** parameters to be trained. The output layer is the final layer where we get the classified output. The output will either be 1 indicating "Fraud" or 0 indicating a "Normal" transaction.

```
# Initialising the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(units =16 , kernel_initializer = 'uniform', activation = 'relu', input_dim = 29))

# Adding the second hidden layer
classifier.add(Dense(units = 24, kernel_initializer = 'uniform', activation = 'relu'))

# Dropout
Dropout(0.5)

# Adding the third hidden layer
classifier.add(Dense(units = 20, kernel_initializer = 'uniform', activation = 'relu'))

# Adding the fourth hidden layer
classifier.add(Dense(units = 24, kernel_initializer = 'uniform', activation = 'relu'))

# Adding the output layer
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
```

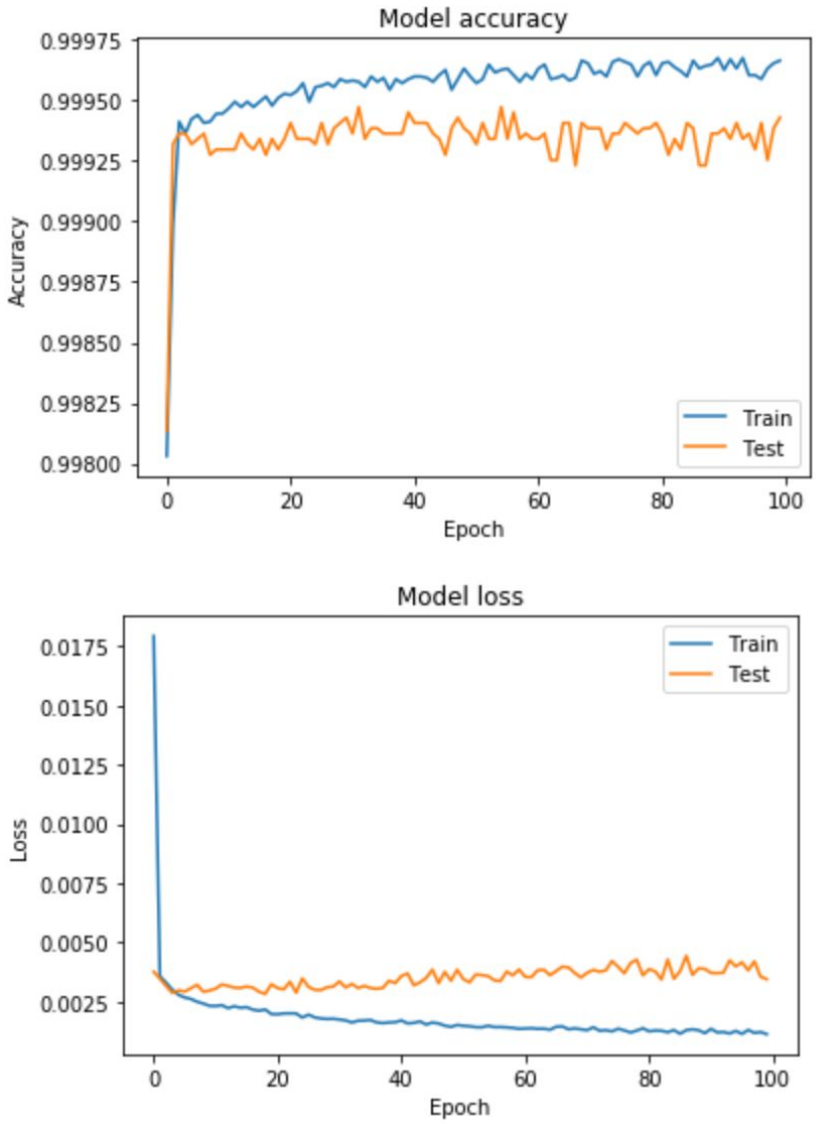
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 16)                480
_________________________________________________________________
dense_7 (Dense)              (None, 24)                408
_________________________________________________________________
dense_8 (Dense)              (None, 20)                500
_________________________________________________________________
dense_9 (Dense)              (None, 24)                504
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 25
=================================================================
Total params: 1,917
Trainable params: 1,917
Non-trainable params: 0
```

Let's train our model for **100 epochs** with a **batch size of 50** samples and save the best performing model to a file. The ModelCheckpoint provided by Keras is really handy for such tasks. The ModelCheckpoint callback class allows you to define where to checkpoint the model weights, how the file should be named and under what circumstances to make a checkpoint of the model. Please note that in the fit function we added a **validation set** (20% of training set). That allows us to evaluate the validation loss and accuracy.

```
# Compiling model
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

# Saves the model weights after each epoch if the validation loss decreased
checkpointer = ModelCheckpoint(filepath = "model.h5", verbose = 1, save_best_only = True)

# Fitting the model to the Training set
history = classifier.fit(X_train, y_train, validation_split =0.2, batch_size = 50, epochs = 100, callbacks = [checkpointer])

# list all data in history
print(history.history.keys())
```

## Evaluation

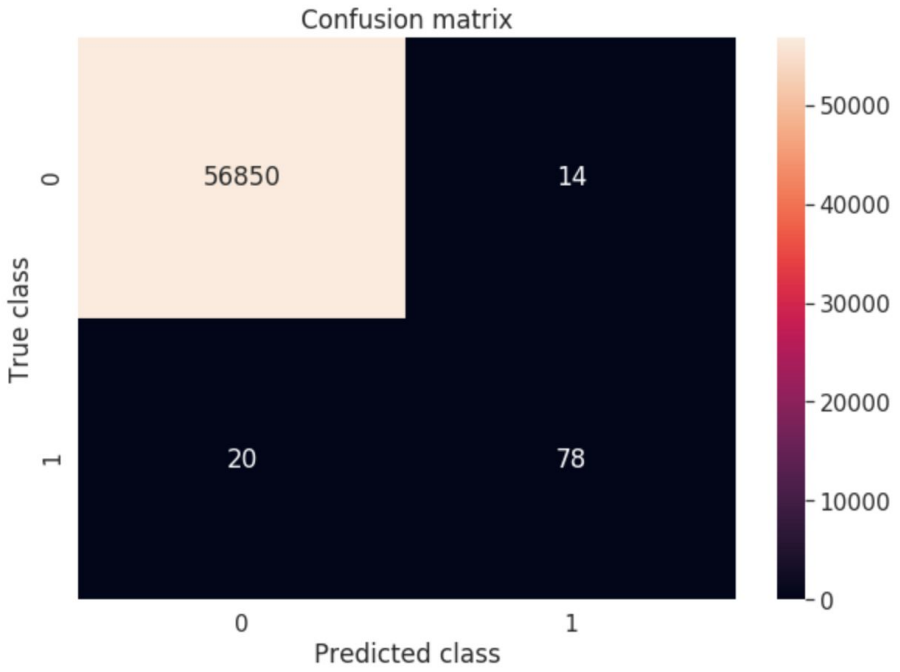
Analysing the evolution of model loss and model accuracy on training and validation sets, the model seems to be performing well enough. The loss of our current model seems to be converging and so more training epochs are not likely going to help. Let's explore this visually to confirm.




Now let's evaluate the model using the test set, we obtain a score of order **0.9994**. It is pretty good value. Now let's look at F1 vs. recall vs. precision to see the trade-off between the three. We can see that the model is well converged regarding these values.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 56864 |
| 1 | 0.85 | 0.80 | 0.82 | 98 |
| micro avg | 1.00 | 1.00 | 1.00 | 56962 |
| macro avg | 0.92 | 0.90 | 0.91 | 56962 |
| weighted avg | 1.00 | 1.00 | 1.00 | 56962 |

Finally let's have a look at the confusion matrix:




Our model seems to catch a lot of the fraudulent cases. With this out-of-sample test set, we capture 78 of the 98 fraudulent transactions, and mistakenly mark 14 transactions as fraudulent.

## Logistic Regression

### Building the model

Logistic Regression is a statistical model that tries to minimize the cost of how wrong a prediction is. It's a classification algorithm that is used where the response variable is categorical. The idea of Logistic Regression is to find a relationship between features and probability of particular outcome. In this part, we will train a Logistic Regression model with Scikit-learn.

```
# Logistic Regression model
LR = LogisticRegression(C=0.01, solver='liblinear').fit(X_train,y_train)
```

### Evaluation

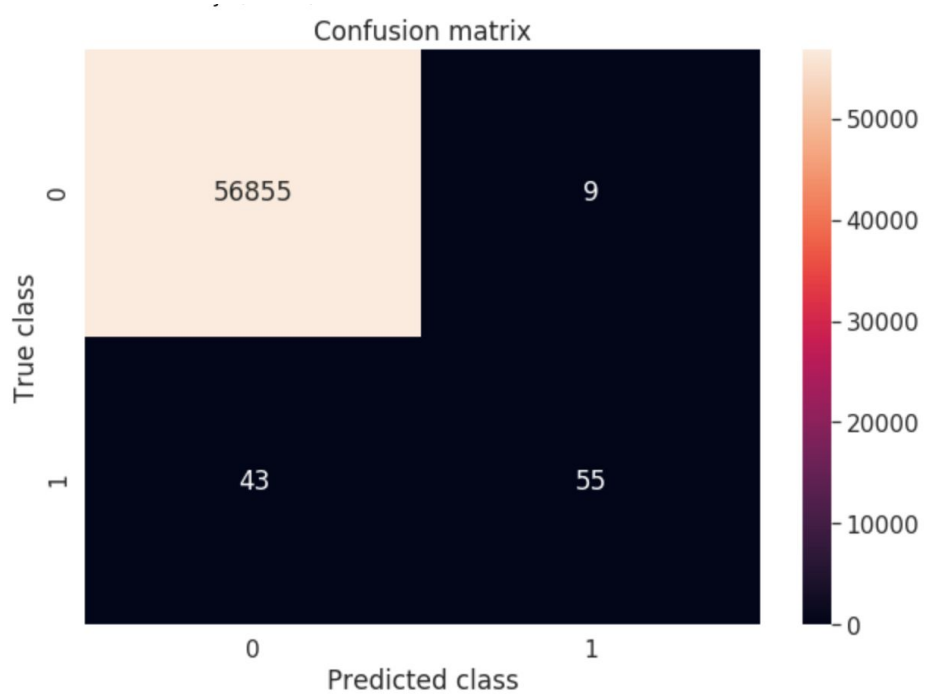
We evaluate the model using  the test set by computing the score, F1, recall and precision to see the trade-off between them.

```
#Prediction
y_pred_log = LR.predict(X_test)

# Accuracy
print("Test Data Accuracy (Score): %0.5f" % accuracy_score(y_test, y_pred_log))

#Let's see how our model performed
print(classification_report(y_test, y_pred_log))
```

We obtain:

```
Test Data Accuracy (Score): 0.99909
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.86      0.56      0.68        98

   micro avg       1.00      1.00      1.00     56962
   macro avg       0.93      0.78      0.84     56962
weighted avg       1.00      1.00      1.00     56962
```

We can see that the model is doing well regarding these values. The final evaluation will be the confusion matrix:




Our Logistic Regression model seems to catch an average lot of fraudulent cases. With this out-of-sample test set, we capture 55 of the 98 fraudulent transactions, and mistakenly mark only 9 transactions as fraudulent.

## K-Means

### Building the model

Our final model will be the K-Means algorithm. The K-means is vastly used for clustering in many data science applications, especially useful if you need to quickly discover insights from unlabeled data. We will train a K-Means model with Scikit-learn.

```
# M-Means algorithm
clusterNum = 2
k_means = KMeans(init = "k-means++", n_clusters = clusterNum, n_init = 50)
k_means.fit(X_train)
```

### Evaluation

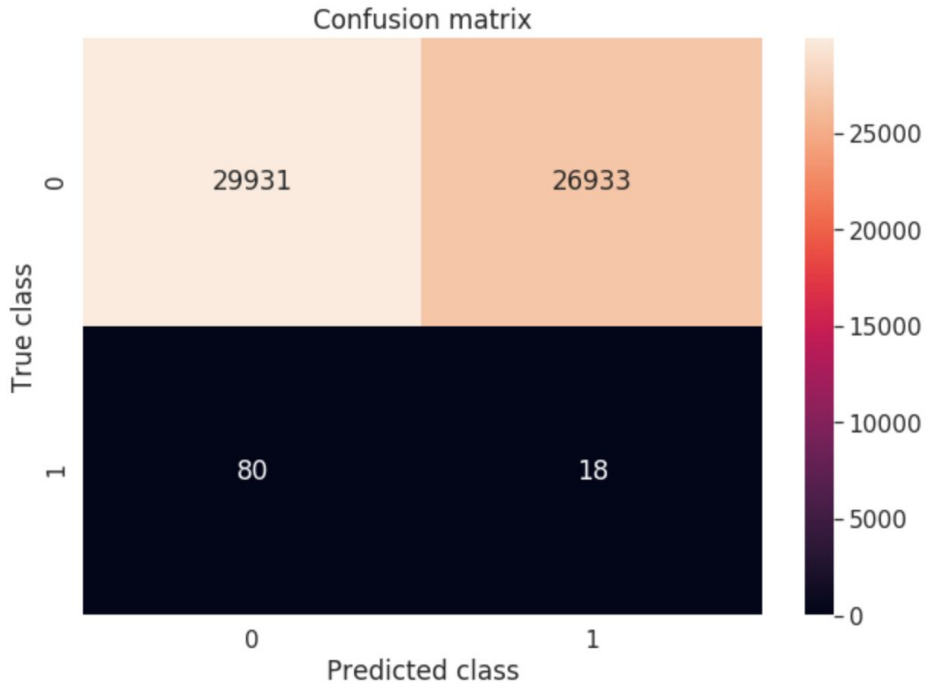
Similarly, we evaluate the model using  the test set by computing the score, F1, recall and precision.

```
#Prediction
y_pred_kmeans = k_means.predict(X_test)

# Accuracy
print("Test Data Accuracy (Score): %0.5f" % accuracy_score(y_test, y_pred_kmeans))

#Let's see how our model performed
print(classification_report(y_test, y_pred_kmeans))
```

We obtain:

```
Test Data Accuracy (Score): 0.52577
              precision    recall  f1-score   support

           0       1.00      0.53      0.69     56864
           1       0.00      0.18      0.00        98

   micro avg       0.53      0.53      0.53     56962
   macro avg       0.50      0.36      0.35     56962
weighted avg       1.00      0.53      0.69     56962
```

We can see clearly that the model is not doing well. The final evaluation will be the confusion matrix:




The K-Means algorithm **seems not** to catch a good lot of fraudulent cases. With this out-of-sample test set, we capture 18 of the 98 fraudulent transactions, and mistakenly mark 26933 transactions as fraudulent.

## Results and Discussion

I've created a Deep Learning algorithm in Keras, a Clustering and a Classification Machine Learning algorithm that can reconstruct what non fraudulent transactions look like on the same data set to detect the credit card frauds. All the algorithms have been analyzed and compared on basis of accuracy for predicting normal cases and frauds.The following table represents accuracy indices for the three algorithms :

| Algorithm | Score | f1-score | Precision | Recall |
|---|---|---|---|---|
| Neural Network | 0.9994 | 0.91 | 0.92 | 0.90 |
| Logistic Regression | 0.9991 | 0.84 | 0.93 | 0.78 |
| K-Means | 0.5257 | 0.35 | 0.36 | 0.50 |

It's quite clear that Artificial Neural Networks are more performant than other Machine Learning algorithms. Indeed, The ANN algorithm is well converged and exceeds the rest of algorithms on 4 accuracy indicators. The Logistic Regression is doing very well on Score but

not too good on the other indicators. The K-Means algorithm does not seem to converge at all and all indicators are giving bad values.

## Conclusion

Credit card fraud has become more and more prevalent in recent years. To improve the level of merchant risk management automatically and effectively, setting up a credit card risk monitoring model that is precise and easy to handle is one of the key tasks of merchant banks.

In this project, I presented the credit card fraud detection problem and provided a brief explanation on the free accessed data used to construct this project.There are many ways of detection of credit card fraud. I used some Deep Learning and Machine Learning algorithms to show that simple Machine Learning methods, like Logistic Regression and K-Means,  would not be a good choice as a fraud detection algorithm and so chose to explore Artificial Neural Networks using Keras platform. I analyzed the utility of these models using several evaluation indicators. By employing neural networks, effectively, banks can detect fraudulent use of a card, faster and more efficiently.