**ITCS 6112**

**SOFTWARE SYSTEM DESIGN AND IMPLEMENTATION**

Report on

# CREDIT CARD FRAUD DETECTION

Using Python

**Team Members**

ABHAY MARATHE

AKASH RAGHUVANSHI

ANKIT KELKAR

KEYUR HANSOTI

KIRAN KOREY

PRASHANT RAVINDRA DESAI

Under the guidance of

**Dr. Ali Sever**

UNC CHARLOTTE

**DEPARTMENT OF COMPUTER SCIENCE**

**College of Computing and Informatics**

**UNC Charlotte, North Carolina – 28262**

**Spring 2018**

## TABLE OF CONTENT

## Abstract:

With the recent advancements in technology, humans have achieved undistinguished feats in 21st century which were thought to be beyond the reach previously. Cashless transactions have become an inseparable part of each financial sector. Healthcare, Education, E-Commerce and etc. profusely make use of digital transactions. But, as these technologies has alleviated human efforts, it has also generated severe threats to security. Credit cards for online transactions are used at its' peak and it has given rise to credit card fraud too. In actual scenario, fraudulent transactions are not that much easy to detect with conventional methods as these transactions look very similar to genuine ones. In this Project, we have modeled credit card fraud detection system through applying software system development concepts at each phase.

## Introduction:

Credit card fraud is a prevalent term for theft and fraud made using or involving a payment card, like a credit card or debit card, as a fraudulent mean of funds in a transaction. The intention can be to obtain goods without paying, or to obtain unauthorized funds from any strange account. According to the United States Federal Trade Commission, the rate of identity theft had increased by 21 percent in 2008, which used to be holding steady during the mid-2000s. Around 46% of Americans have been victim to credit card fraud in the past 5 years. The Nilson Report estimates that in 2016, losses topped $24.71 billion. That represents a 12% increase over the previous year.

In this project, we have used Kaggle dataset that had transactions of two days in September by European cardholders. We have used Python due to its' efficient and professional structure for machine learning projects. We have used Python libraries such as pandas, numpy, matplotlib for plotting the graphs, pickle etc. This project is divided into five modules such as Input module, Train module, Test module, Performance metrics module and Display module. We have implemented QDA (Quadratic Discriminant Analysis) and LDA (Linear Discriminant Analysis) and added some utility methods to measure the performance of the algorithms. The final output in the form of *Trained Model* can be hosted as a web service to further detect fraudulent transactions. Along with that xml diagrams such as Architecture diagram, Class diagram, Sequence diagram and Activity diagrams are also drawn in accordance to project requirements. While developing the project, we have strictly followed the software development methodology approach.

## Data Description:

The dataset which we used from Kaggle contains transaction made in two days in September 2013 by European cardholders. Out of 284,807 records 492 were frauds. This is very unbalanced dataset. The dataset contains only numerical input variables which are result of a PCA transformation. There are total 31 columns in the dataset out of first 28 columns entitled as V1 to V28 are called Features except first column. The first column Time describes the seconds elapsed between each transaction and first transaction in dataset. The second last column Amount is the transacted amount and the last column which is Class has value 1 in case of fraud and 0 otherwise.

The dataset was originally collected and analyzed during a research collaboration of Worldline and the Machine Learning Group (http://mlg.ulb.ac.be) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection.

## Why Python:

We have analyzed the fraud detection using Python programming language. Python is very popular and professional to work on machine learning projects. Python has particular tools like which are very useful in working with machine learning and we have used those tools in our analysis.

Python has huge set of libraries which can be easily used such as pandas, numpy, matplotlib for plotting the graphs, pickle etc.

## Software and Hardware requirements

### Software requirements:

- Windows 7/8/10.
- Python version 3.6.
- Anaconda version 3.
- Jupyter notebook/ipython.
- Python libraries(tenserflow, numpy, pandas, sklearn).

### Hardware requirements:

- RAM: 8 GB.
- Processor i5 and above.
- Minimum 10 GB Hard disk space.

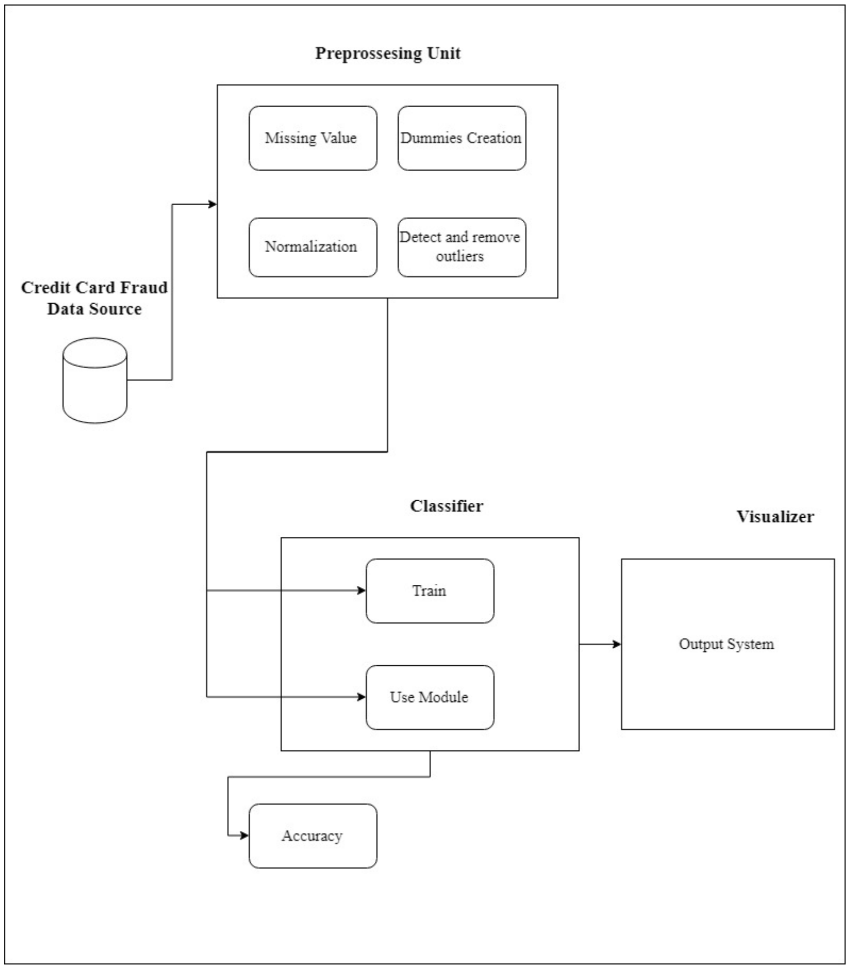
## Implementation description:

- The process will start with getting the essential data sets for credit card fraud detection from authentic sources. The data is taken from this link**.** For the analysis, 2 days of data from September 2013 of European cardholders is considered.
- Total 284,807 rows of data are considered.
- This data is fed into the preprocessing module, where missing values, junk values etc rows are removed. The data is also normalized (brought in the range of 0 to 1) to make the computations faster, this is an optional step.
- The preprocessed data is then split into train and test data and the output class is separated from them as well.
- The trained data is then fed into the classifier and the model is trained using this data.
- Once the training is completed, the test data is used to validate the accuracy of the model. The model is retrained if the required accuracy is not obtained.
- Graphical representations of these accuracies are plotted to get a better visualization of the trained model.
- This trained model can be hosted as a web service to further detect fraudulent transactions.

## Modules:

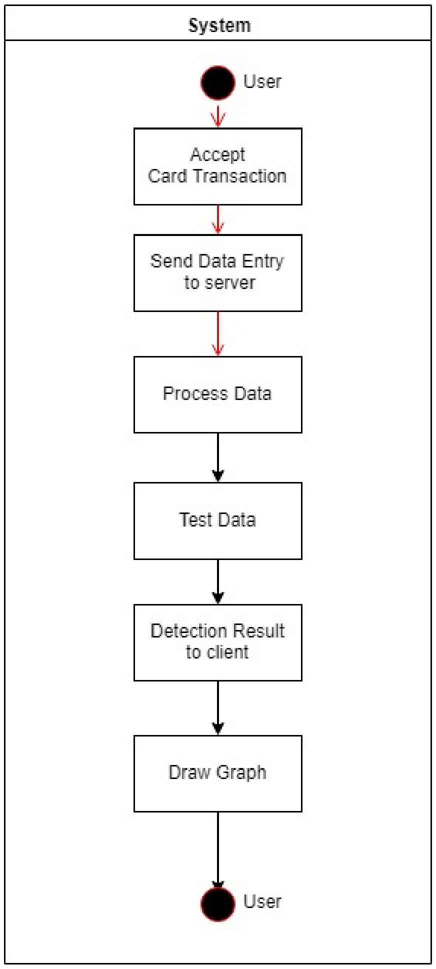
- Input module
- Train module
- Test module
- Performance metrics module
- Display module

## UML Diagrams:

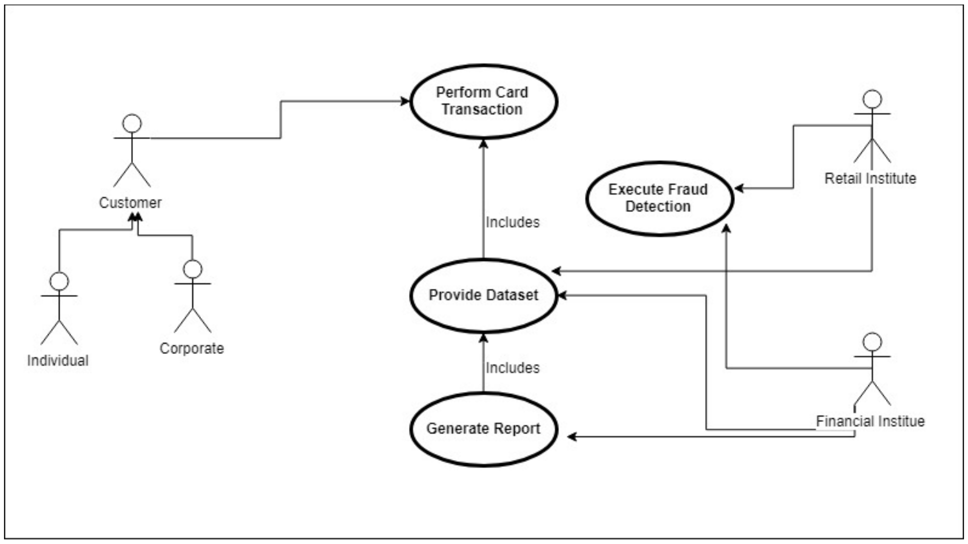


Software architecture diagram

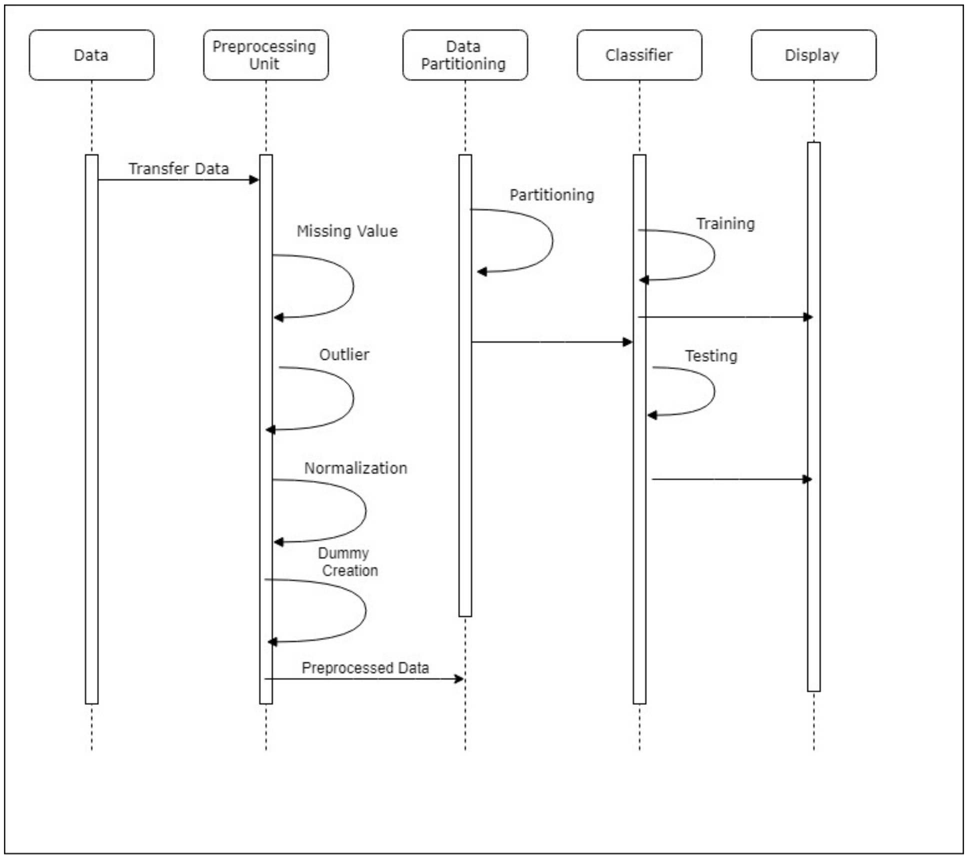
Credit Card Fraud Detection



Activity diagram

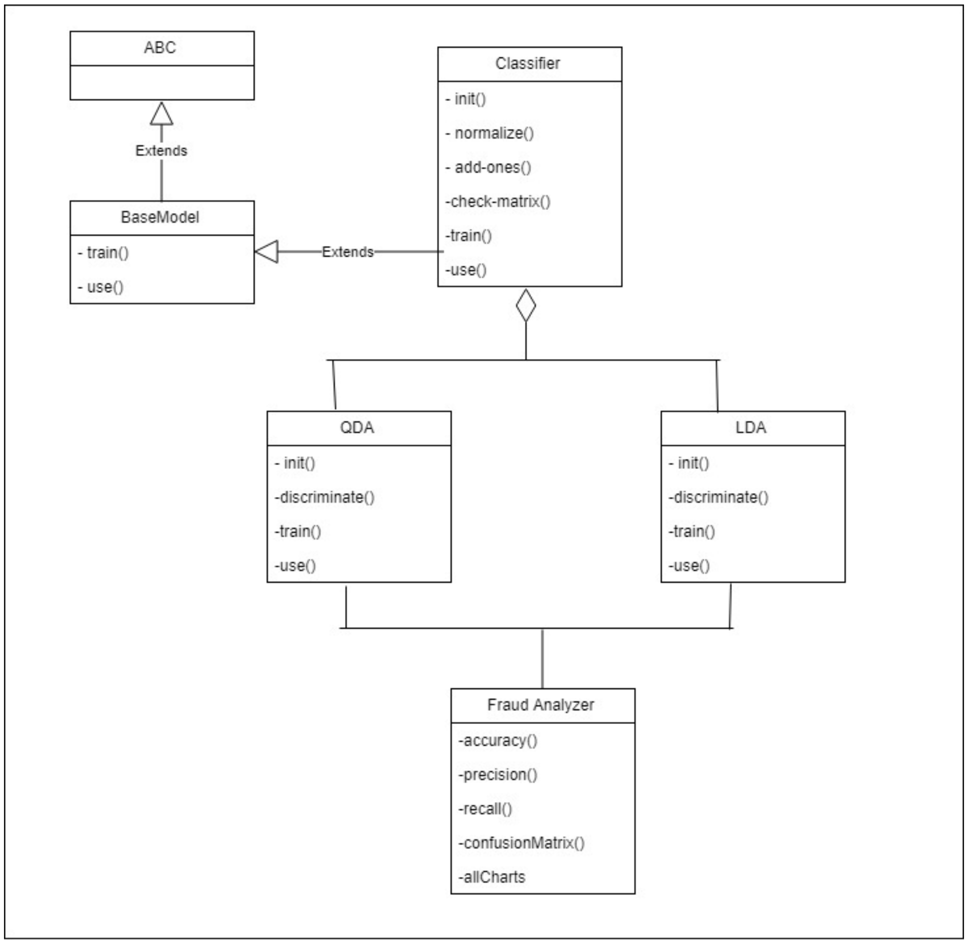


Use case diagram

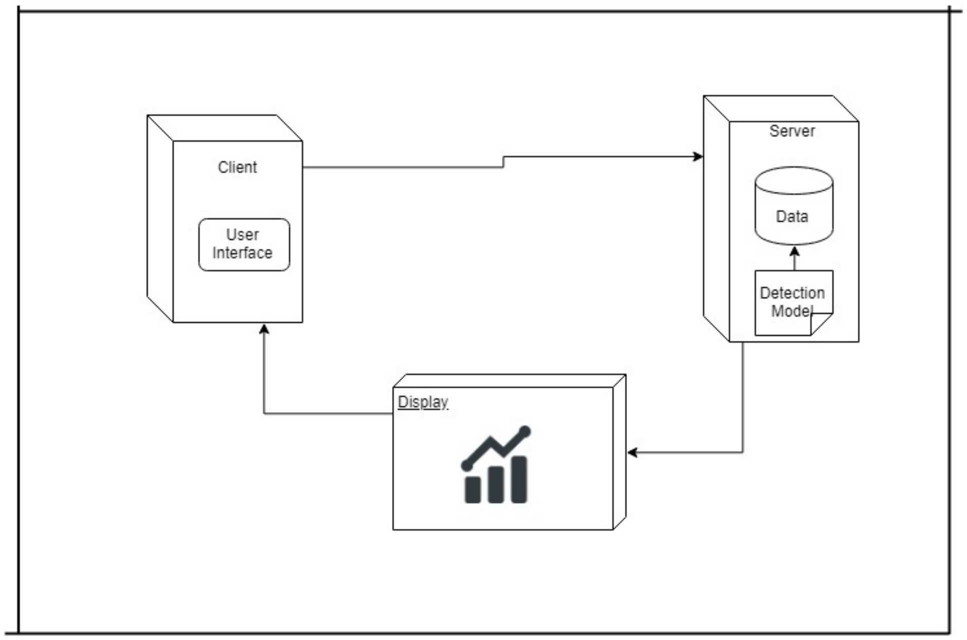


Sequence diagram

Credit Card Fraud Detection



Class diagram



Deployment diagram

## Results:

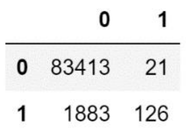
The QDA and LDA algorithms are implemented by following an object-oriented pattern and the correctness of the algorithms is checked by comparing the various evaluation metrics with the same metrics obtained by library implementation of these algorithms.

The result of applying QDA and LDA algorithms on this dataset is very interesting. Both the models give a high accuracy but to be not blindfolded by the accuracy alone as this dataset is highly unbalanced we have to also look at the false positive count.

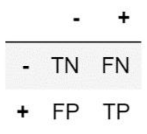
## QDA on complete dataset:

The QDA classifier was applied to the complete dataset and a accuracy of 97% was obtained, but the instances that actually matter are the instances which are actually fraudulent transaction.

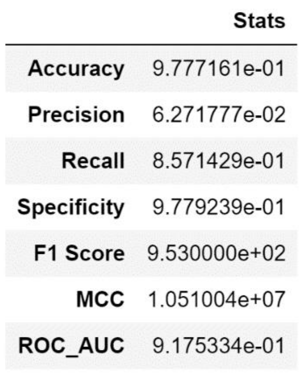
As we can see from the False Positive count in the below confusion matrix 1883 instances where classified as normal transactions where as in reality they were fraudulent transactions. Hence, we cannot depend on this model.



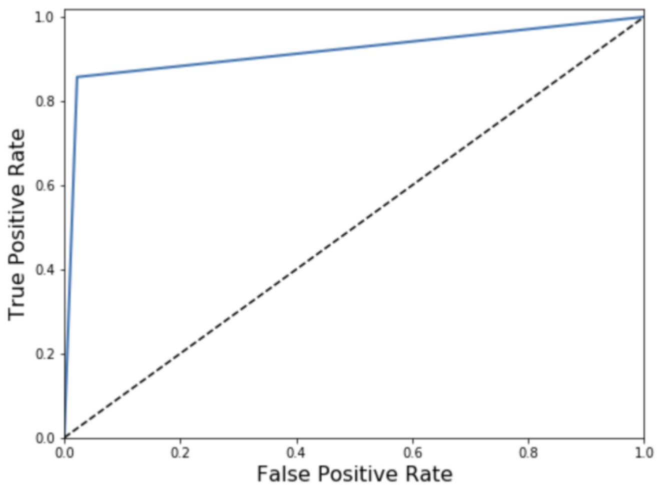
| | 0 | 1 |
|---|---|---|
| 0 | 83413 | 21 |
| 1 | 1883 | 126 |



| | - | + |
|---|---|---|
| - | TN | FN |
| + | FP | TP |



| | Stats |
|---|---|
| Accuracy | 9.777161e-01 |
| Precision | 6.271777e-02 |
| Recall | 8.571429e-01 |
| Specificity | 9.779239e-01 |
| F1 Score | 9.530000e+02 |
| MCC | 1.051004e+07 |
| ROC_AUC | 9.175334e-01 |



## QDA on Under-Sampled dataset:

Since the QDA did not work on the complete dataset because of the imbalance nature of the dataset, let's apply the same on the Under-Sampled dataset.

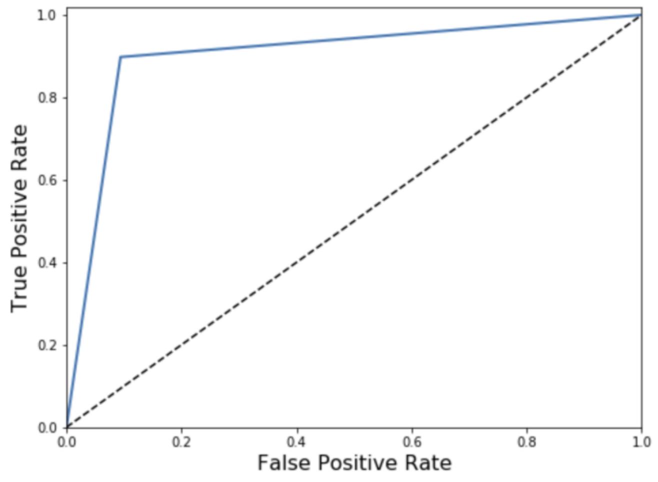
As we can see we obtained a low False positive count and a good True Positive dataset and accuracy of 90%. This model looks good so far.

Let's compare this model with another Classifier i.e. LDA and see how it performs.

|  | 0 | 1 |
|---|---|---|
| **0** | 135 | 15 |
| **1** | 14 | 132 |

|  | - | + |
|---|---|---|
| **-** | TN | FN |
| **+** | FP | TP |

|  | Stats |
|---|---|
| **Accuracy** | 0.902027 |
| **Precision** | 0.904110 |
| **Recall** | 0.897959 |
| **Specificity** | 0.906040 |
| **F1 Score** | 15.500000 |
| **MCC** | 17819.990412 |
| **ROC_AUC** | 0.902000 |



## QDA using Sklearn Library:

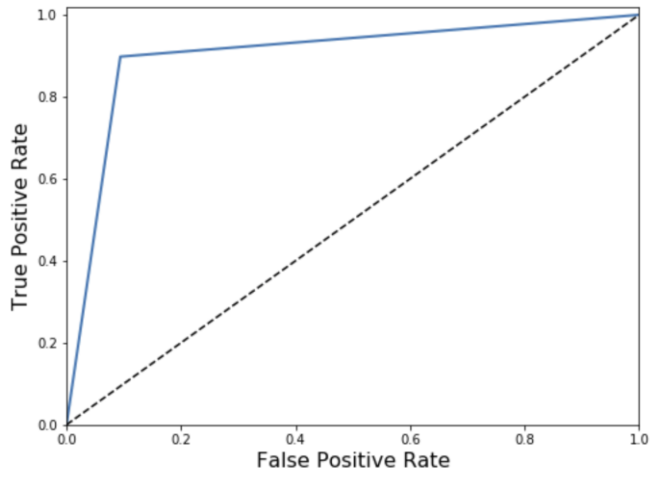
Before moving on to a different classifier we applied QDA from a standard sklearn library to check the correctness of the implemented algorithm.

As we can see from the confusion matrix and the stats table below, they are inline with those of the implemented version of it.

| | 0 | 1 |
|---|---|---|
| **0** | 135 | 15 |
| **1** | 14 | 132 |

| | - | + |
|---|---|---|
| **-** | TN | FN |
| **+** | FP | TP |

| | Stats |
|---|---|
| **Accuracy** | 0.902027 |
| **Precision** | 0.904110 |
| **Recall** | 0.897959 |
| **Specificity** | 0.906040 |
| **F1 Score** | 15.500000 |
| **MCC** | 17819.990412 |
| **ROC_AUC** | 0.902000 |



## Apply LDA on Original Data

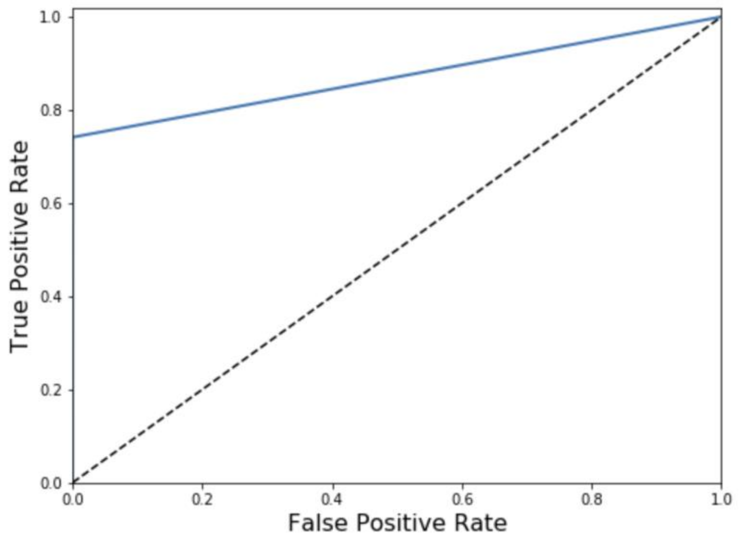
The LDA classifier was applied to the complete dataset and a accuracy of 99% was obtained, as mentioned before the instances that actually matter are the instances which are actually fraudulent transaction.

As we can see from the False Positive count in the below confusion matrix only15 instances where classified as normal transactions where as in reality they were fraudulent transactions. So, this model performs pretty good on the unbalanced dataset as well.

| | 0 | 1 |
|---|---|---|
| **0** | 85281 | 38 |
| **1** | 15 | 109 |

| | - | + |
|---|---|---|
| **-** | TN | FN |
| **+** | FP | TP |

| | Stats |
|---|---|
| **Accuracy** | 9.993797e-01 |
| **Precision** | 8.790323e-01 |
| **Recall** | 7.414966e-01 |
| **Specificity** | 9.998241e-01 |
| **F1 Score** | 2.750000e+01 |
| **MCC** | 9.295629e+06 |
| **ROC_AUC** | 8.706604e-01 |



## Apply LDA on Under-Sampled

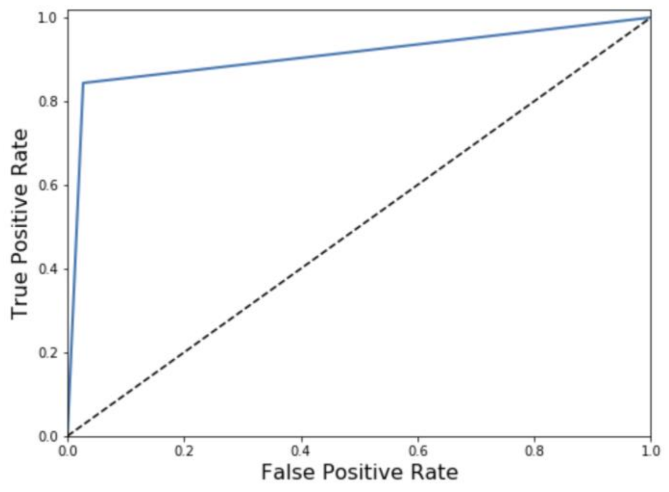
Let’s apply the same on the Under-Sampled dataset just to be double sure.

As we can see we obtained a very low False positive count, a good True Positive count and accuracy of 90%. This model performs the same as QDA for equally proportionate dataset.

| | 0 | 1 |
|---|---|---|
| **0** | 145 | 23 |
| **1** | 4 | 124 |

| | - | + |
|---|---|---|
| **-** | TN | FN |
| **+** | FP | TP |

| | Stats |
|---|---|
| **Accuracy** | 0.908784 |
| **Precision** | 0.968750 |
| **Recall** | 0.843537 |
| **Specificity** | 0.973154 |
| **F1 Score** | 14.500000 |
| **MCC** | 17979.995761 |
| **ROC_AUC** | 0.908346 |



## LDA using Sklearn Library

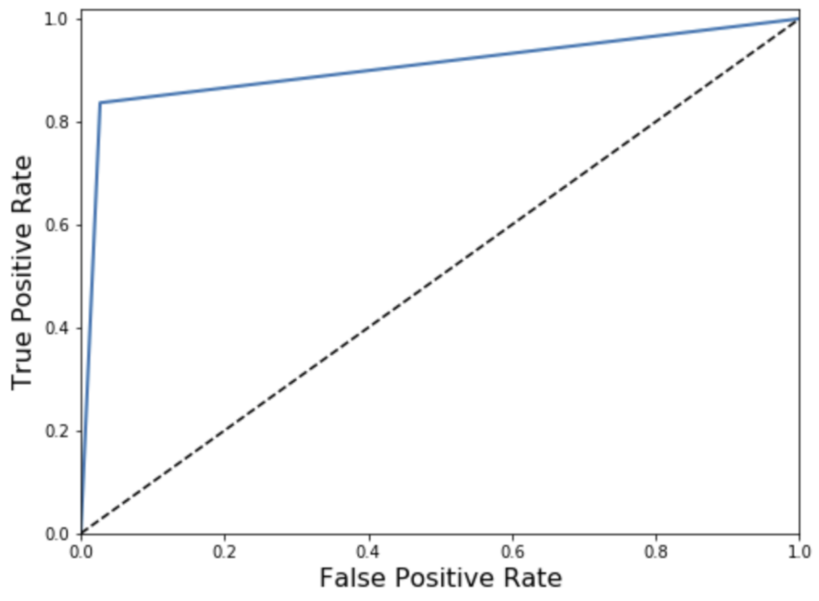
Before concluding we applied LDA from a standard sklearn library to check the correctness of the implemented algorithm.

As we can see from the confusion matrix and the stats table below, they are in line with those of the implemented version of it.

| | 0 | 1 |
|---|---|---|
| **0** | 145 | 24 |
| **1** | 4 | 123 |

| | - | + |
|---|---|---|
| **-** | TN | FN |
| **+** | FP | TP |

| | Stats |
|---|---|
| **Accuracy** | 0.905405 |
| **Precision** | 0.968504 |
| **Recall** | 0.836735 |
| **Specificity** | 0.973154 |
| **F1 Score** | 15.000000 |
| **MCC** | 17834.995572 |
| **ROC_AUC** | 0.904945 |



## Conclusion:

The experiment demonstrates the execution and viability of our framework and exhibit the usefulness of learning the transactions of the cardholders. Comparative investigations reveal that the Accuracy of the framework is near **90%** percent over a wide variety of the input information. The framework is additionally adaptable for taking care of extensive volumes of transactions. Qualitative data analysis (QDA) and Linear Distribution Analysis (LDA) over complete data set and under sampled data shows the model trained on under sampled data give high accuracy compared to the counterpart. The results are also close to accuracy calculated using Python's Sklearn library which verifies the performance of our python script.

## Future Enhancements:

- A web-interface can be created to provide a front end to load data files and display results.
- The model accuracy should be near to 100% or 1 resulting in detection of all the fraud transaction.
- The model can be improvised using reinforced learning which enables the model to detect fraud and train itself at the same time.

## References:

- Dataset - https://www.kaggle.com
- https://medium.com/@curiousily/credit-card-fraud-detection-using-autoencoders-in-keras-tensorflow-for-hackers-part-vii-20e0c85301bd
- Draw.io to draw diagrams- www.draw.io