Credit Card Fraud Detection

Mário Damhur

Final Project - Advanced Data Science Capstone (IBM)

Part 1

Stakeholder presentation

Use Case

- It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

- Question to answer:
  It is possible to recognized a fraudulant transaction given the informations of the transaction?

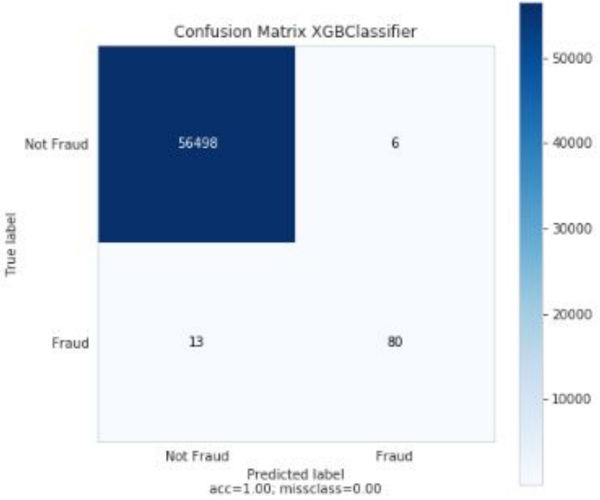
- We want to classifying the transaction on two classes: Fraudulant and Not Fraudulant (Real)

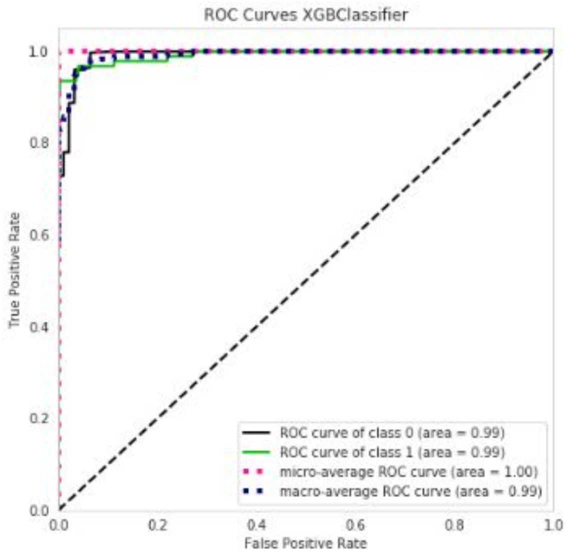
Dataset

- The datasets contains transactions made by credit cards in September 2013 by european cardholders.
- This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions.
- The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.
- There are 31 columns.
- The dataset is public and is available on https://www.kaggle.com/mlg-ulb/creditcardfraud

Solution to Use Case

- Using machine learning to get insights from the data
- Models: Random Forest, XGBoost and Deep Feed Foward Neural Network.
- Best results:
  - XGBoost
    - Acc: 1.0
    - Precision: 0.94
    - Recall: 0.76
    - F1: 0.83
    - AUC: 0.99





Part 2

Peer presentation

Architectural Choices

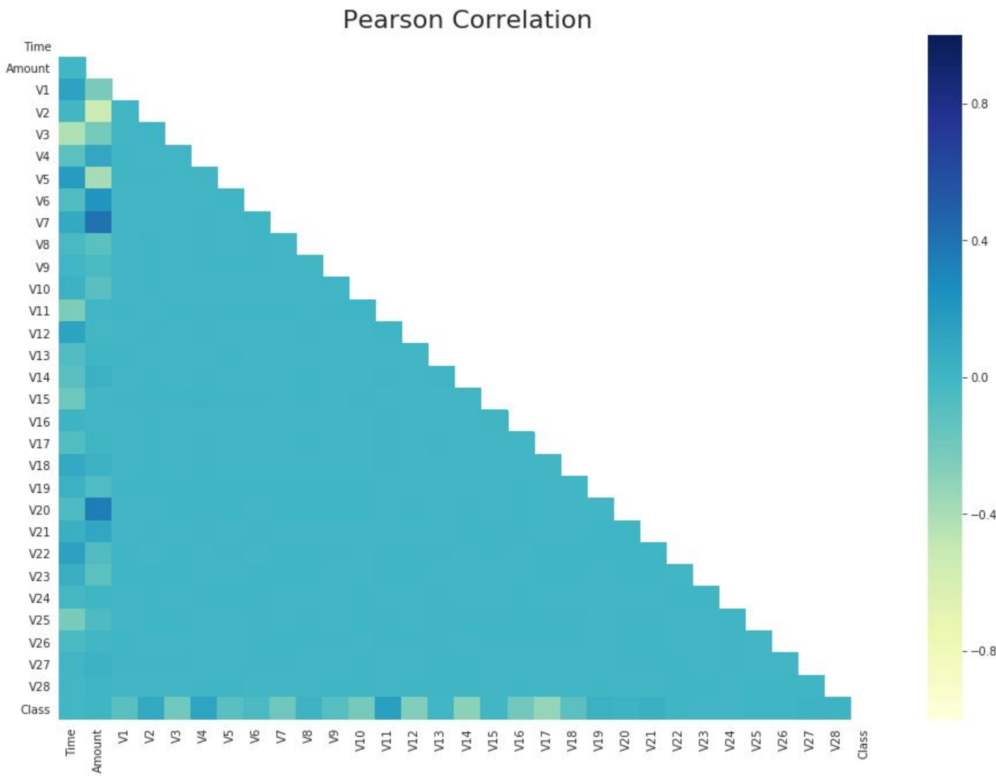
Development Environment

Exploratory Analysis

Model Frameworks

Initial Data Exploration

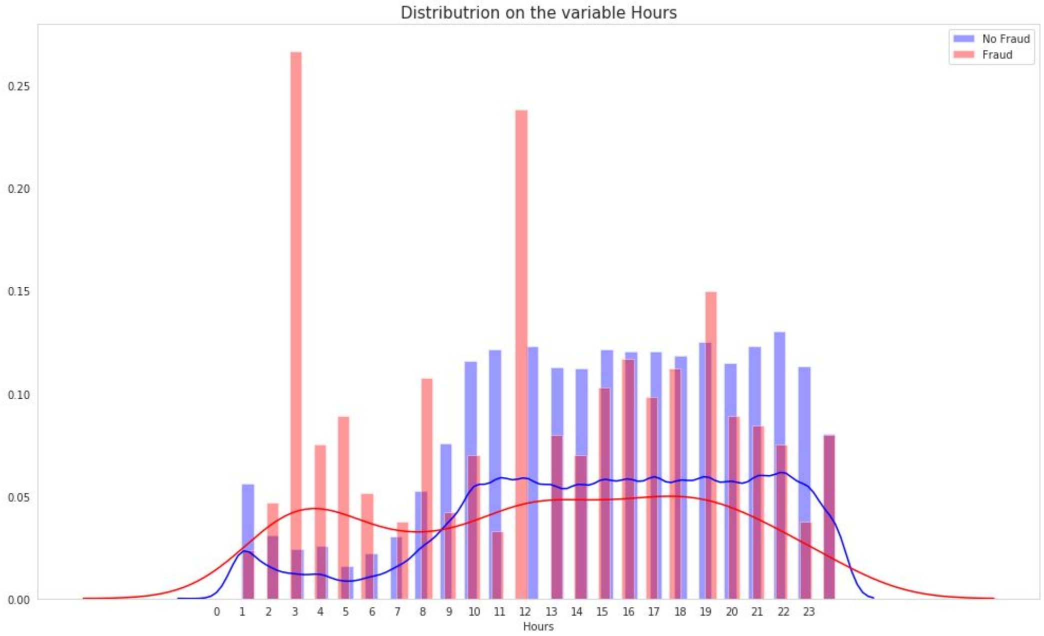
- Dataset has 30 features and 1 target
  - 28 features are confidentials (V1 - V28) generated by PCA
  - The others 2 are Time and Amount
- Extremely unbalanced
- No missing values
- Drop few outliers
  - Amount with value 0
- Data is all clean



	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.000	-1.360	-0.073	2.536	1.378	-0.338	0.462	0.240	0.099	0.364	...	-0.018	0.278	-0.110	0.067	0.129	-0.189	0.134	-0.021	149.620	0
1	0.000	1.192	0.266	0.166	0.448	0.060	-0.082	-0.079	0.085	-0.255	...	-0.226	-0.639	0.101	-0.340	0.167	0.126	-0.009	0.015	2.690	0
2	1.000	-1.358	-1.340	1.773	0.380	-0.503	1.800	0.791	0.248	-1.515	...	0.248	0.772	0.909	-0.689	-0.328	-0.139	-0.055	-0.060	378.660	0
3	1.000	-0.966	-0.185	1.793	-0.863	-0.010	1.247	0.238	0.377	-1.387	...	-0.108	0.005	-0.190	-1.176	0.647	-0.222	0.063	0.061	123.500	0
4	2.000	-1.158	0.878	1.549	0.403	-0.407	0.096	0.593	-0.271	0.818	...	-0.009	0.798	-0.137	0.141	-0.206	0.502	0.219	0.215	69.990	0

Feature Engineer

- The variable Time contains the seconds elapsed between each transaction and the first transaction in the dataset.
- Since the dataset was collected for two days. We can convert the variable Time to Hours between 00h - 23h
- We can get more insights like: What are the hours that fraudulent transactions most occurred

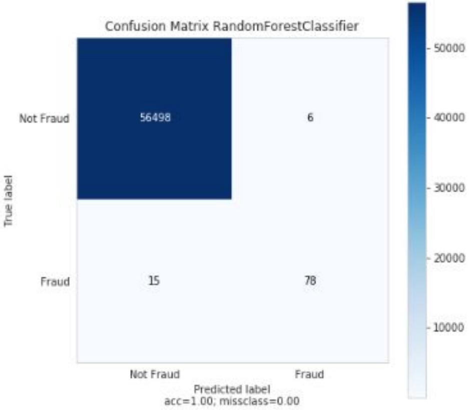


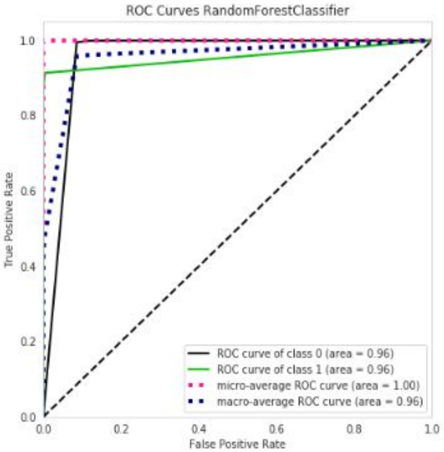
Model Algorithm: Random Forest

- Standard parameters

- Final scores was generated by cross validate with 5 folds

- Results:
  - Acc: 1.0
  - Precision: 0.94
  - Recall: 0.74
  - F1: 0.83
  - AUC: 0.96



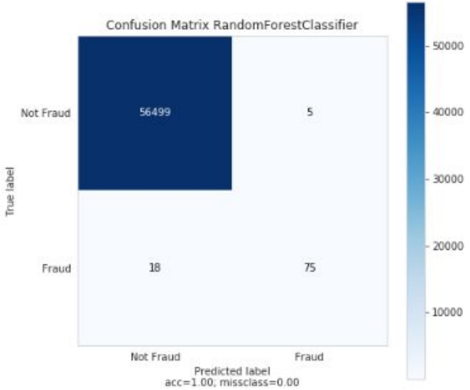


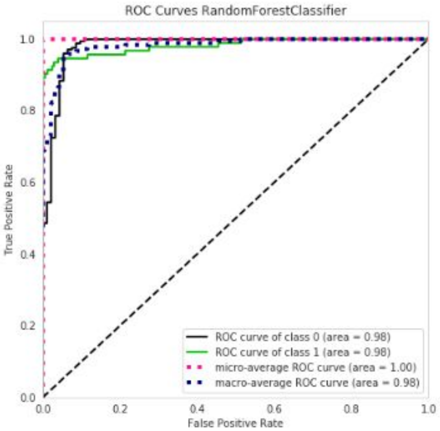
- RandomizedSearchCV

- Final scores was generated by cross validate with 5 folds

```
random_grid = {
    'max_depth': [2, 3, 5, 10, None],
    'n_estimators': [50, 100, 200, 500],
    "criterion": ["entropy", "gini"],
    "class_weight": [None, 'balanced']
}
```

- Results:
  - Acc: 1.0
  - Precision: 0.94
  - Recall: 0.71
  - F1: 0.81
  - AUC: 0.98





Model Algorithm: XGBoost

- Standard parameters

- Final scores was generated by cross validate with 5 folds

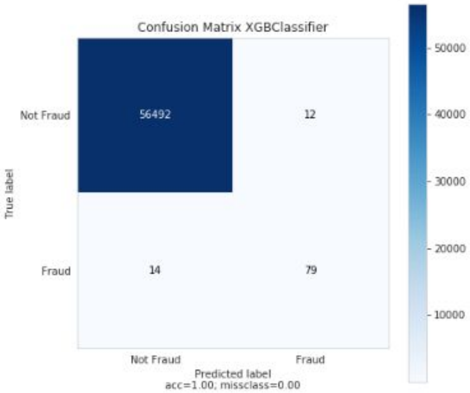
- Results:
  - Acc: 1.0
  - Precision: 0.95
  - Recall: 0.76
  - F1: 0.84
  - AUC: 0.99

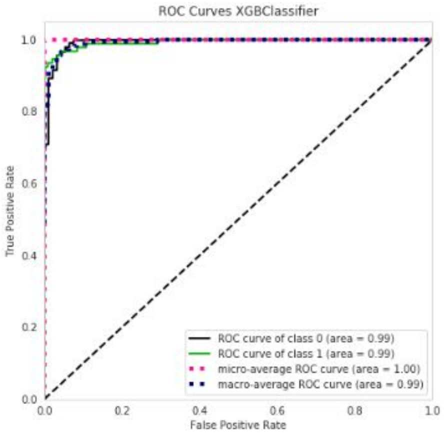
- RandomizedSearchCV

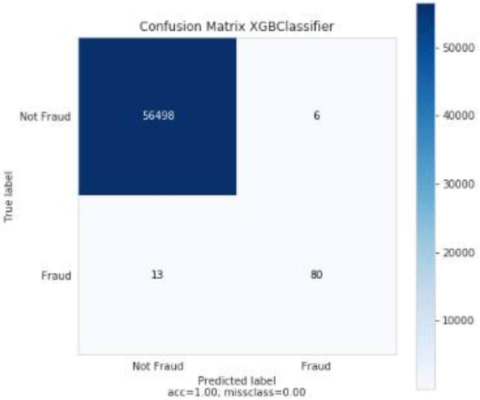
- Final scores was generated by cross validate with 5 folds

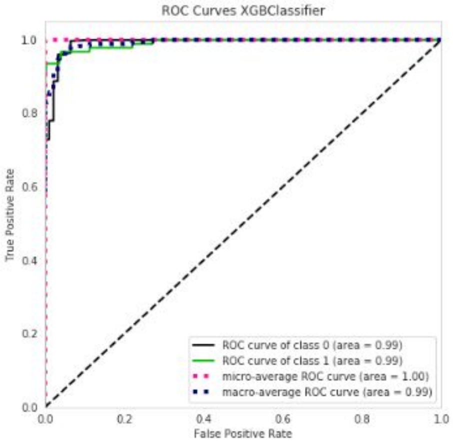
```
random_grid = {
        'learning_rate' : [0.02, 0.1, 0.2, 0.3, 0.4],
        'n_estimators' : [50, 100, 200, 500],
        'gamma': [0.5, 1, 1.5, 2, 5],
        'max_depth': [2, 3, 5, 10]
        }
```

- Results:
  - Acc: 1.0
  - Precision: 0.94
  - Recall: 0.76
  - F1: 0.83
  - AUC: 0.99



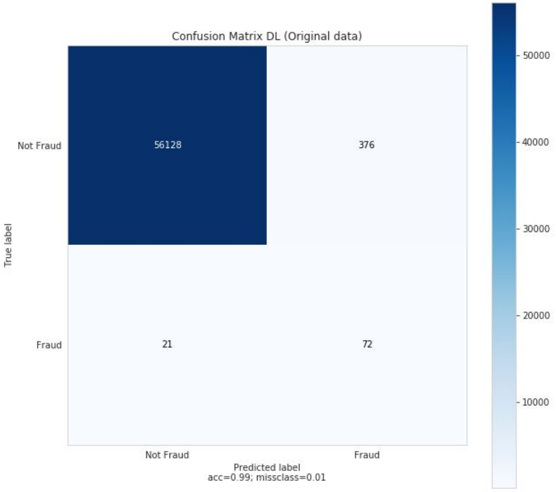


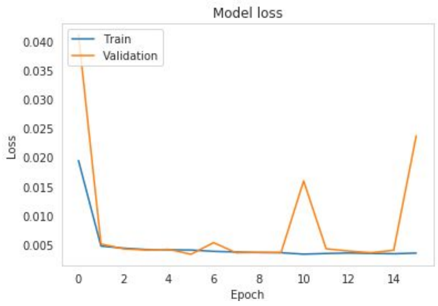




Model Algorithm: Deep Feed Foward Neural Network

- Preprocessing the data
- Early stopping: 10 patience
- Batch size: 64
- Epochs: 100
- Optimizer: Nadam
- Loss: Binary CrossEntropy






```
def model_fnn():
    model = tf.keras.models.Sequential()

    model.add(tf.keras.layers.Dense(512, input_shape=(X_train.shape[1],)))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Activation("relu"))
    #model.add(tf.keras.layers.Dropout(0.2))

    model.add(tf.keras.layers.Dense(256, kernel_initializer="he_normal", use_bias=False))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Activation("relu"))
    #model.add(tf.keras.layers.Dropout(0.2))

    model.add(tf.keras.layers.Dense(128, kernel_initializer="he_normal", use_bias=False))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Activation("relu"))
    #model.add(tf.keras.layers.Dropout(0.2))

    model.add(tf.keras.layers.Dense(64, kernel_initializer="he_normal", use_bias=False))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Activation("relu"))
    #model.add(tf.keras.layers.Dropout(0.2))

    model.add(tf.keras.layers.Dense(32, kernel_initializer="he_normal", use_bias=False))
    model.add(tf.keras.layers.BatchNormalization())
    model.add(tf.keras.layers.Activation("relu"))
    model.add(tf.keras.layers.Dropout(0.2))

    model.add(tf.keras.layers.Dense(1, activation="sigmoid"))

    model.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Nadam())

    return model
```

Summary

- Best model: XGBoost with RandomizeSearchCV

- DL model with just few epochs achieved viable performance, however, the falses positives and negatives are relatively high.

- Future work:
  - Experiment algorithms to deal with unbalanced dataset like:
    - SMOTE, ADASYN
    - Cost sensitive learning