CIS 5603

HW #7

Zhengkun Ye

03/25/2022

Comparison between Credit Card Fraud Detection via Supervised Method and Unsupervised Method

Introduction: Credit card companies must be able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase. The goal of this analysis is to predict credit card fraud in transactional data. As far as I am concerned, it will have great value in preventing financial crimes.

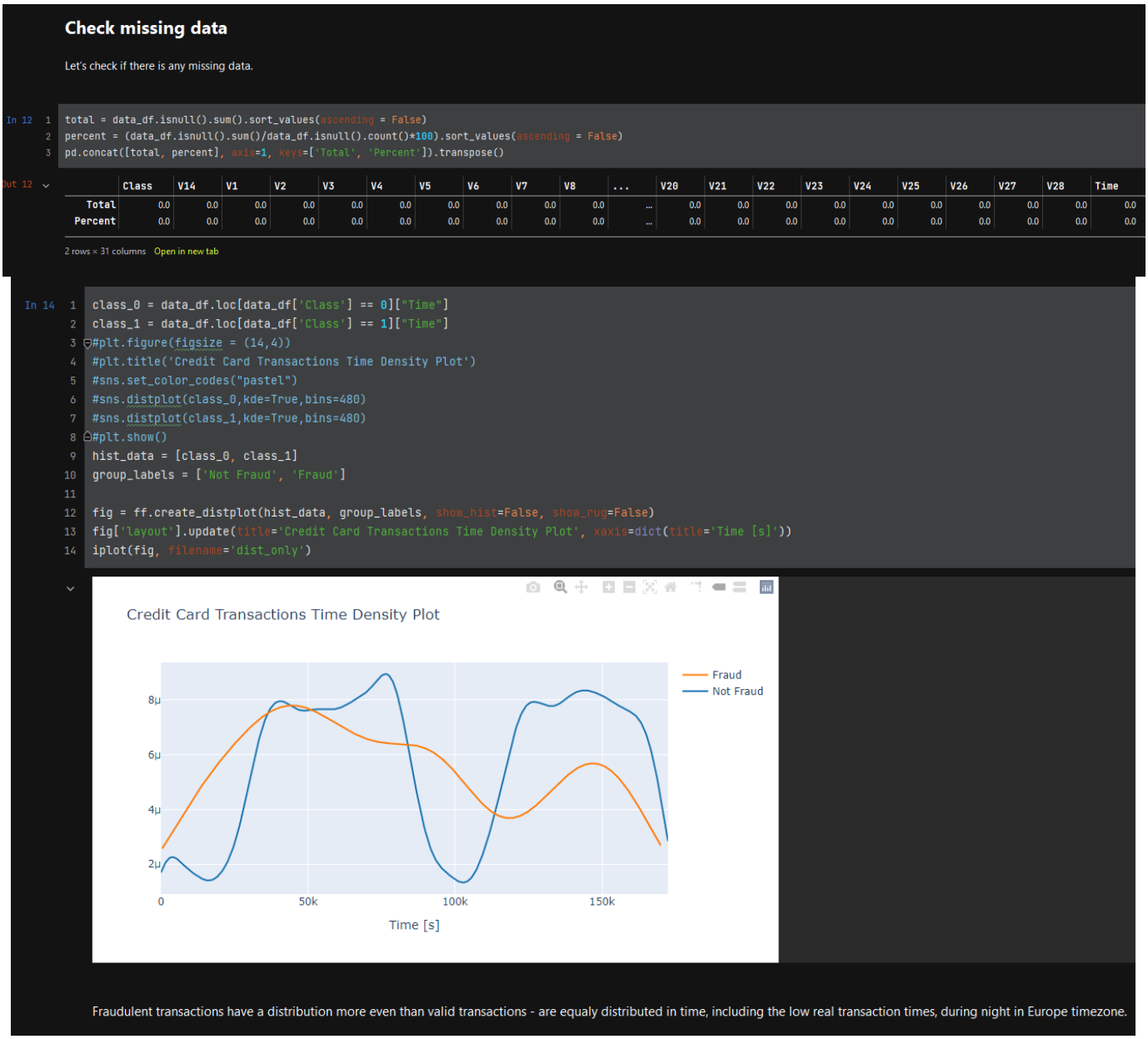
Data: The whole dataset is 31 Columns x 284808 Rows. The datasets contain transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. Looking at the Time feature, we can confirm that the data contains 284,807 transactions, during 2 consecutive days (or 172792 seconds). There is no missing data in the entire dataset. The dataset is highly unbalanced, the positive class (frauds) account for **0.172%** of all transactions. The training data set is the former 198,365 (70%) instances, 383 represents fraud transactions, 197,982 transactions are genuine. The test data is the later 86,442 (30%) transactions.

Methods: Use and compare some different supervised algorithms V.S. unsupervised learning methods such as *Random Forest, AdaBoost, Catboost, LightGBM, XGBoost, etc.* (XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework) with *Multivariate Gaussian probability*, *Auto Encoders*, *Local Outlier Factor LOF*, *Robust Covariance (Elliptic Envelope)*, *Isolation Forest* and *One Class SVM*. Our goal was to determine the classifiers we are going to use and decide which one has higher accuracy. PCA (principal component analysis) was implemented at the very beginning: It contains only numerical input variables which are the result of a PCA transformation. Due to confidentiality issues, there are not provided the original features and more background information about the data. Features V1, V2, … V28 are the principal components obtained with PCA. 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 made use of for example-dependent cost-sensitive learning. Feature Class is the response variable, and it takes value 1 in case of fraud and zero otherwise. AUC is an estimate of the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. For this reason, the AUC is widely considered to be a better measure than a classification error rate based upon a single prior probability or KS statistic threshold.

Approach:

- Explore the data (EDA process). Check missing data.



Check missing data

Let's check if there is any missing data.

```
In 12  1  total = data_df.isnull().sum().sort_values(ascending = False)
       2  percent = (data_df.isnull().sum()/data_df.isnull().count()*100).sort_values(ascending = False)
       3  pd.concat([total, percent], axis=1, keys=['Total', 'Percent']).transpose()
```

Out 12

	Class	V14	V1	V2	V3	V4	V5	V6	V7	V8	...	V20	V21	V22	V23	V24	V25	V26	V27	V28	Time
Total	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Percent	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

2 rows × 31 columns Open in new tab

```
In 14  1  class_0 = data_df.loc[data_df['Class'] == 0]["Time"]
       2  class_1 = data_df.loc[data_df['Class'] == 1]["Time"]
       3 #plt.figure(figsize = (14,4))
       4  #plt.title('Credit Card Transactions Time Density Plot')
       5  #sns.set_color_codes("pastel")
       6  #sns.distplot(class_0,kde=True,bins=480)
       7  #sns.distplot(class_1,kde=True,bins=480)
       8 #plt.show()
       9  hist_data = [class_0, class_1]
      10  group_labels = ['Not Fraud', 'Fraud']
      11
      12  fig = ff.create_distplot(hist_data, group_labels, show_hist=False, show_rug=False)
      13  fig['layout'].update(title='Credit Card Transactions Time Density Plot', xaxis=dict(title='Time [s]'))
      14  iplot(fig, filename='dist_only')
```

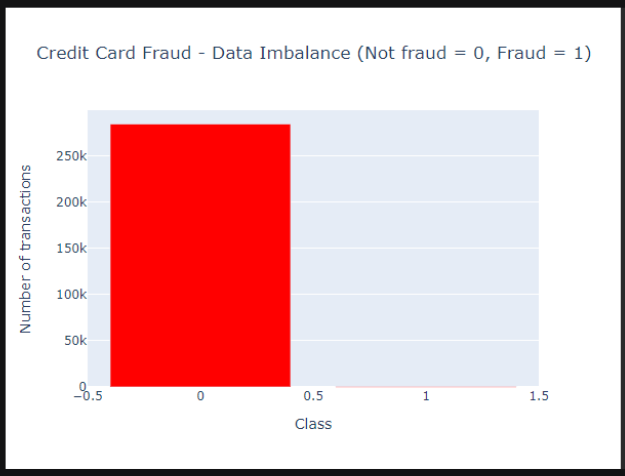

Fraudulent transactions have a distribution more even than valid transactions - are equaly distributed in time, including the low real transaction times, during night in Europe timezone.

Imbalance Dataset

Let's check data imbalance with respect with *target* value, i.e. **Class**.

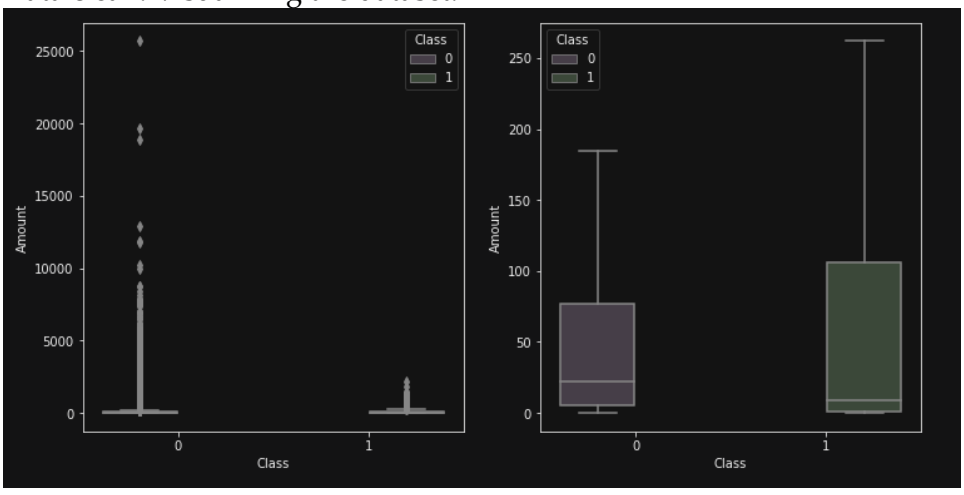
In 13
```
temp = data_df["Class"].value_counts()
df = pd.DataFrame({'Class': temp.index,'values': temp.values})

trace = go.Bar(
    x = df['Class'],y = df['values'],
    name="Credit Card Fraud - Data Imbalance(Not fraud = 0, Fraud = 1)",
    marker=dict(color="Red"),
    text=df['values']
)
data = [trace]
layout = dict(title = 'Credit Card Fraud - Data Imbalance (Not fraud = 0, Fraud = 1)',
          xaxis = dict(title = 'Class', showticklabels=True),
          yaxis = dict(title = 'Number of transactions'),
          hovermode = 'closest',width=600
         )
fig = dict(data=data, layout=layout)
iplot(fig, filename='class')
```



Only **492** (or **0.172%**) of transaction are fraudulent. That means the data is highly imbalanced with respect with target variable **Class**.

- Data clean. Visualizing the dataset.

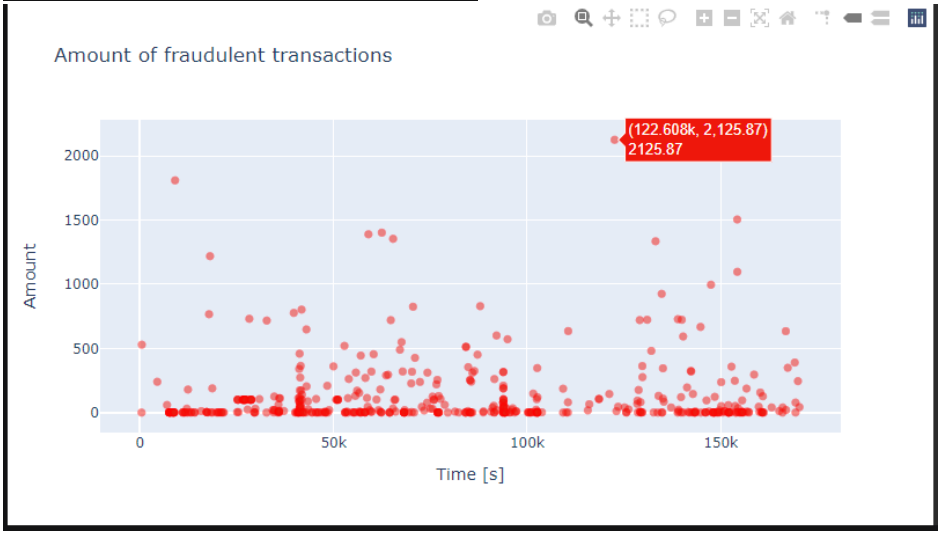


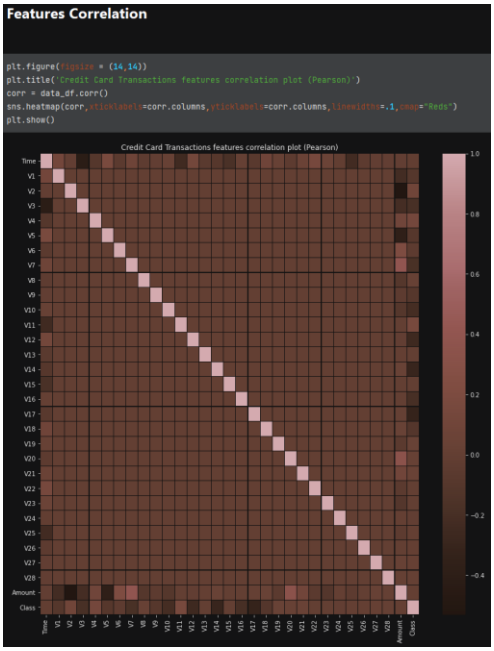
```
class_0.describe()

count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

```
class_1.describe()

count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

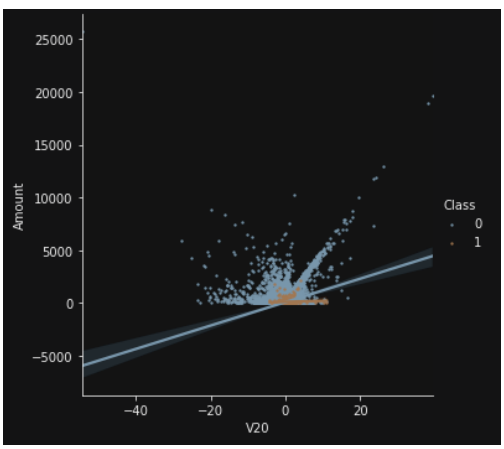


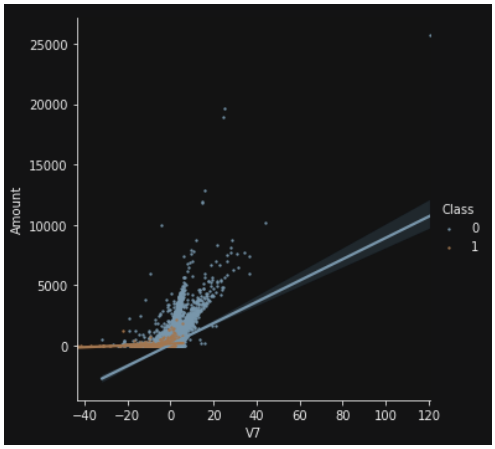


As expected, there is no notable correlation between features V1-V28. There are certain correlations between some of these features and Time (inverse correlation with V3) and Amount (direct correlation with V7 and V20, inverse correlation with V1 and V5).

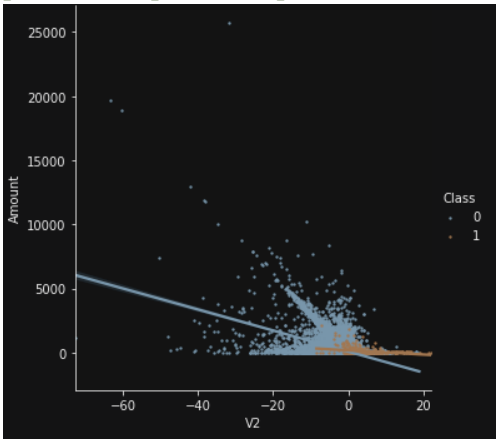
Let's plot the correlated and inverse correlated values on the same graph.

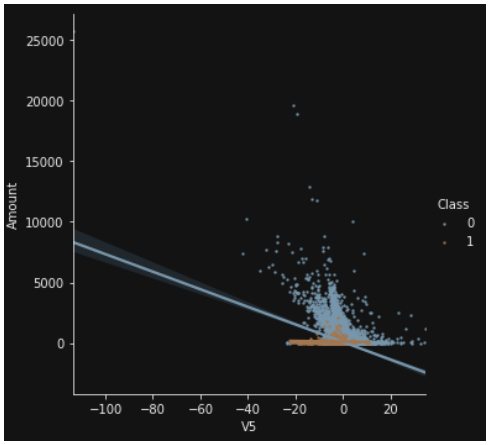
Let's start with the direct correlated values: {V20; Amount} and {V7; Amount}.



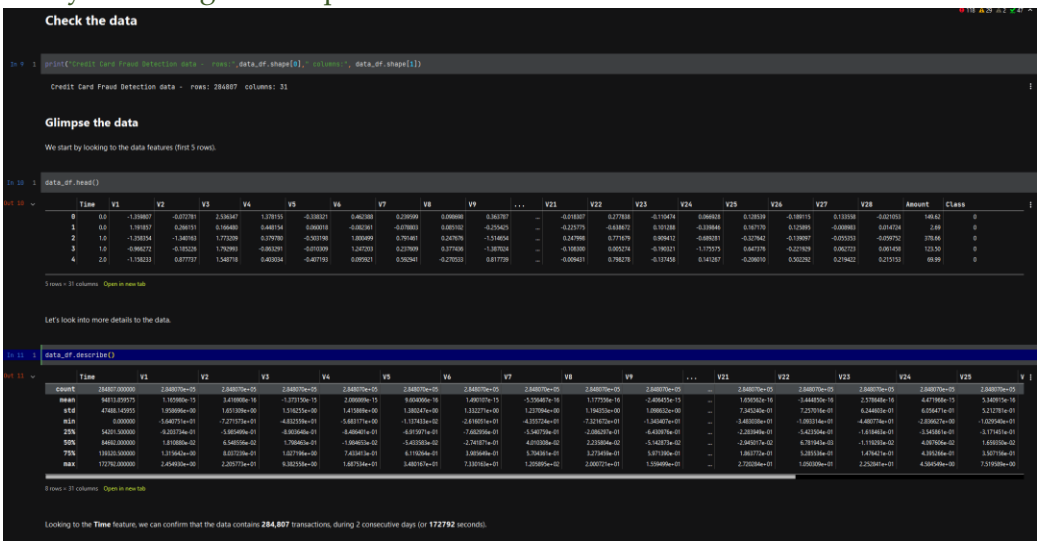


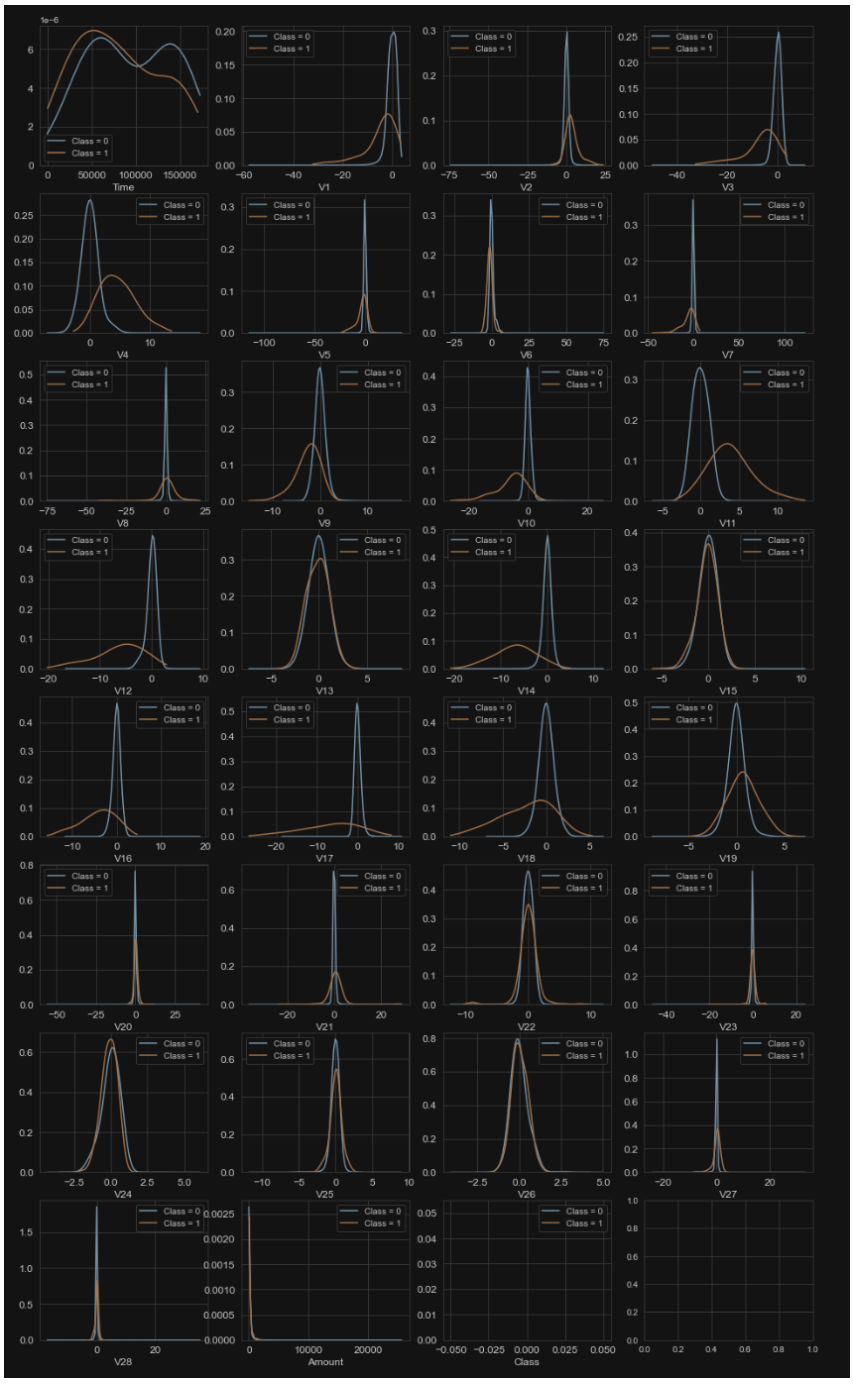
We can confirm that the two couples of features are correlated (the regression lines for Class = 0 have a positive slope, whilst the regression line for Class = 1 have a smaller positive slope). Let's plot now the inverse correlated values:





We can confirm that the two couples of features are inverse correlated (the regression lines for "Class = 0" have a negative slope while the regression lines for "Class = 1" have a very small negative slope.

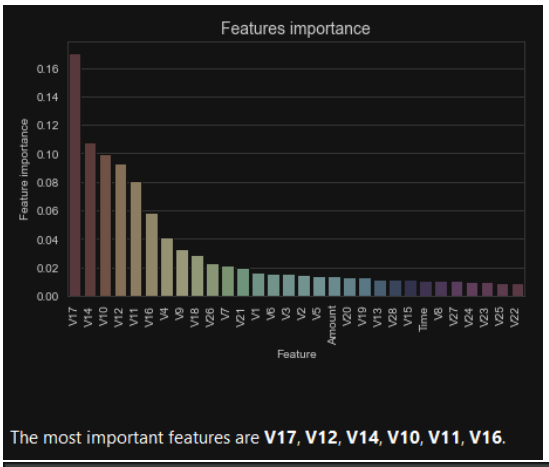




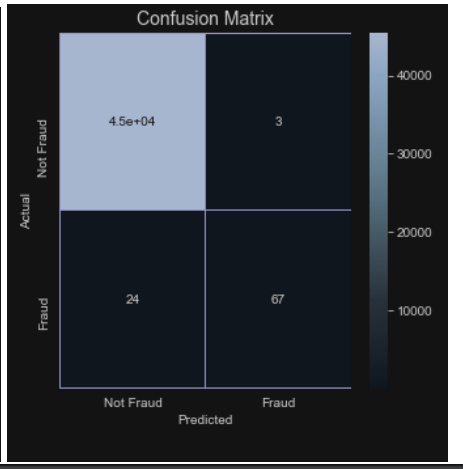
For some of the features we can observe a good selectivity in terms of distribution for the two values of Class: V4, V11 have separated distributions for Class values 0 and 1, V12, V14, V18 are partially separated, V1, V2, V3, V10 have a quite distinct profile, whilst V25, V26, V28 have similar profiles for the two values of Class. In general, with just a few exceptions (Time and Amount), the distribution of the feature for legitimate transactions (values of Class = 0) is centered around 0, sometimes with a long queue at one of the extremities. At the same time, the fraudulent transactions (values of Class = 1) have a skewed (asymmetric) distribution.

Supervised Method_Predictive Models Implementation:

1. Random Forest Classifier



The most important features are **V17**, **V12**, **V14**, **V10**, **V11**, **V16**.

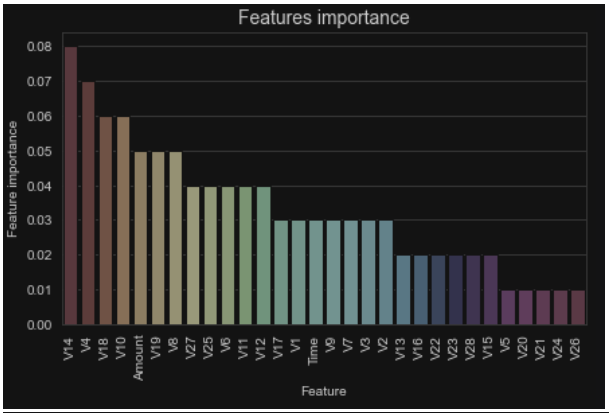


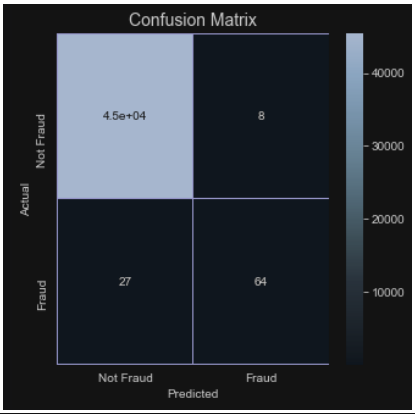

```
roc_auc_score(valid_df[target].values, preds)
```

```
0.8680988851510862
```

The **ROC-AUC** score obtained with **Random Forest Classifier** is **0.868**.

2. AdaBoost Classifier





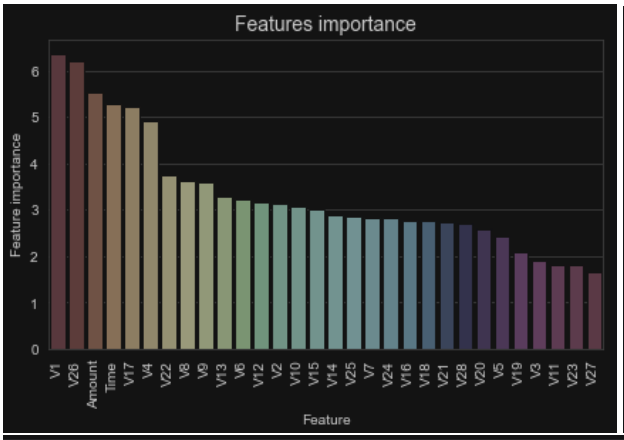

```
roc_auc_score(valid_df[target].values, preds)
```

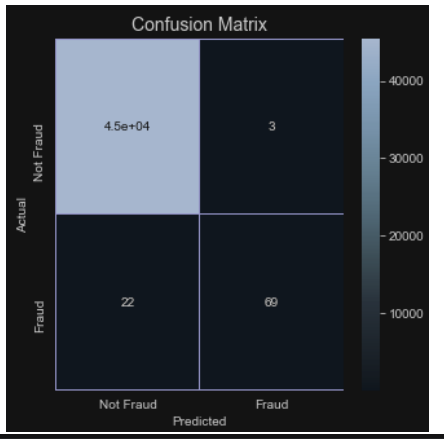
```
0.8515603970329333
```

Add Code Cell | Add Markdown Cell

The ROC-AUC score obtained with AdaBoost Classifier is **0.852**.

3. CatBoost Classifier






```
roc_auc_score(valid_df[target].values, preds)
```

```
0.8790878961400972
```

The ROC-AUC score obtained with CatBoostClassifier is **0.879**.

4. XGboost

Train the model

Let's train the model.

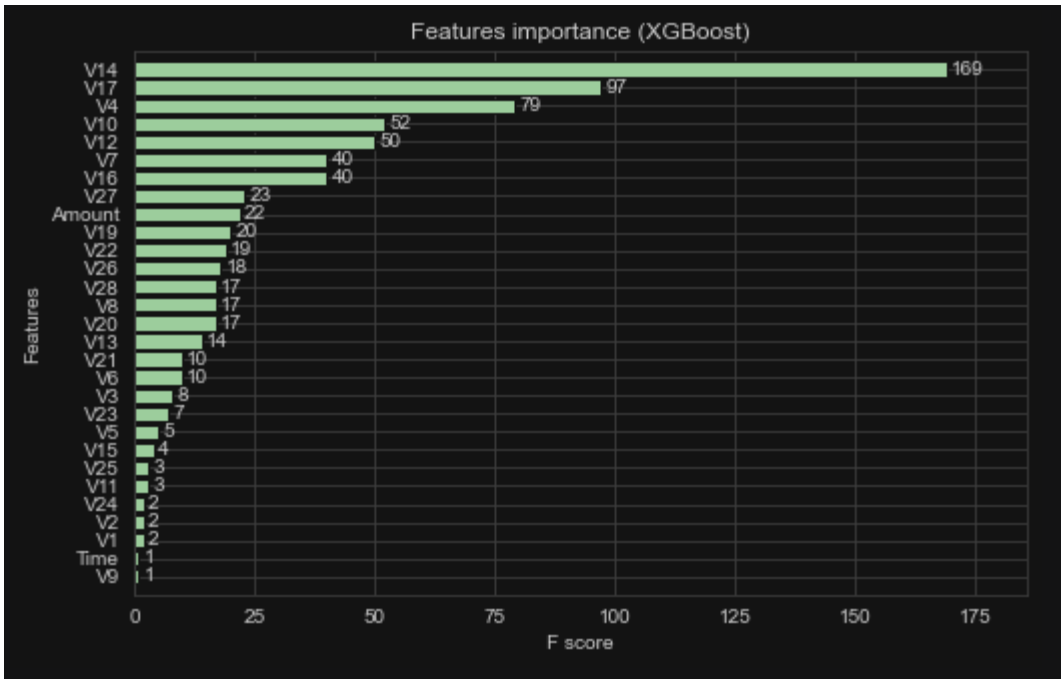
```
MAX_ROUNDS = 1000 #iterations
EARLY_STOP = 50 #early stop

model = xgb.train(params,
                  dtrain,
                  MAX_ROUNDS,
                  watchlist,
                  early_stopping_rounds=EARLY_STOP,
                  maximize=True,
                  verbose_eval=VERBOSE_EVAL)
```

```
[0] train-auc:0.86186   valid-auc:0.83501
Multiple eval metrics have been passed: 'valid-auc' will be used for
early stopping.

Will train until valid-auc hasn't improved in 50 rounds.
[50]    train-auc:0.91859   valid-auc:0.92285
[100]   train-auc:0.95066   valid-auc:0.94726
[150]   train-auc:0.97502   valid-auc:0.97256
[200]   train-auc:0.98911   valid-auc:0.98063
[250]   train-auc:0.99247   valid-auc:0.97975
Stopping. Best iteration:
[200]   train-auc:0.98911   valid-auc:0.98063
```

The best validation score (ROC-AUC) was **0.981**, for round **200**.




```
roc_auc_score(test_df[target].values, preds)
```

```
0.9874261106028059
```

The AUC score for the prediction of fresh data (test set) is **0.987**.

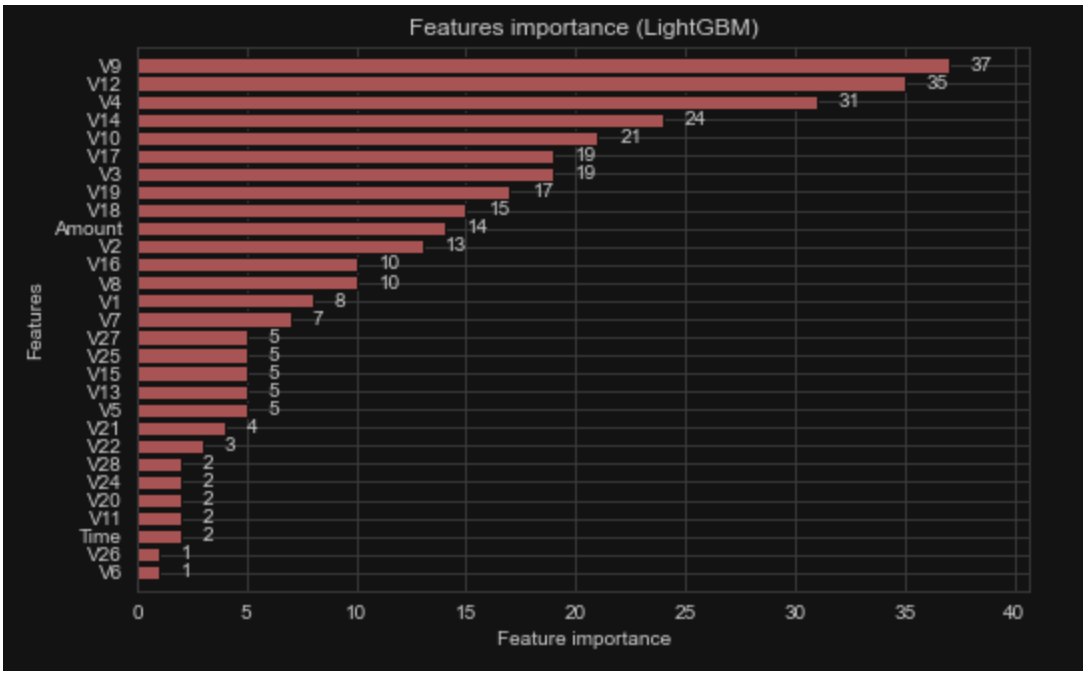
5. LightGBM

```
Training until validation scores don't improve for 100 rounds
[50]    train's auc: 0.98424    valid's auc: 0.919256
[100]   train's auc: 0.993413   valid's auc: 0.895608
[150]   train's auc: 0.989089   valid's auc: 0.89223
Early stopping, best iteration is:
[54]    train's auc: 0.985393   valid's auc: 0.92095
```

Add Code Cell | Add Markdown Cell

Best validation score was obtained for round **54**, for which **AUC** ~= **0.985**.

Let's plot variable importance.




```
roc_auc_score(test_df[target].values, preds)
```

```
0.9478398477391069
```

The ROC-AUC score obtained for the test set is 0.948.

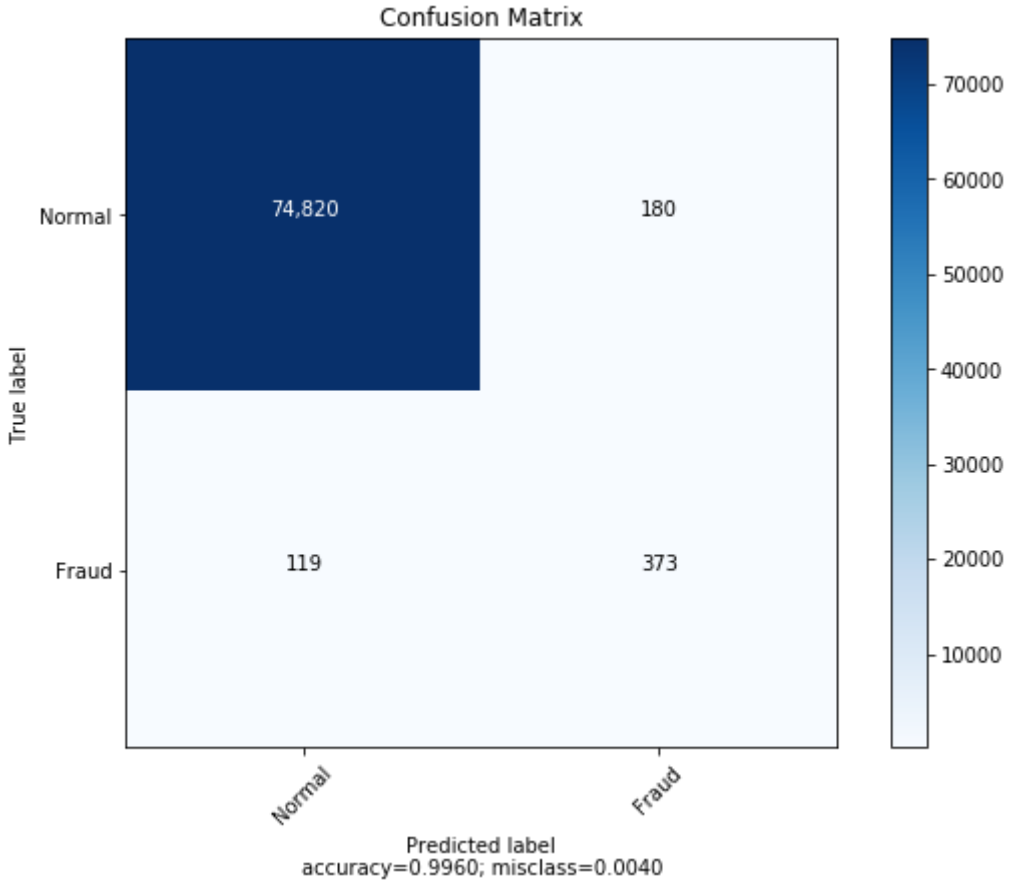
A little thought:

GBMs are more sensitive to overfitting if the data is noisy. Training takes longer because trees are built sequentially. GBMs are harder to tune than RF (Boosting is based on weak learners (high bias, low variance)/ Random Forest (low bias, high variance)). There are typically three parameters: the number of trees, the depth of trees, and the learning rate, and each tree built is shallow. The most prominent application of random forest is multi-class object detection in large-scale real-world computer vision problems. RF methods can manage a large amount of training data efficiently and are inherently suited for multi-class problems. In a real-world scenario, an unsupervised model is used primarily as a seed to create labeled data unless the risk rules based on domain knowledge can be tailored for the problem. For example, if the problem is to identify anomalies in network traffic metrics, such as time between logins and distance between origins can be used to formulate a risk rule. However, abnormal logins obtained by applying this rule must be audited to determine their accurate labels. Take dentifying the probability of an employee committing securities fraud as an example. The behavioral data that the organization

captures is very high-dimensional, and the relationship between the data points is complex. Hence without in-depth domain knowledge, formulating risk rules is difficult. Furthermore, it is tough to validate these risk rules with confidentiality issues.

Unsupervised Method_(Unsupervised Anomaly Detection Models)

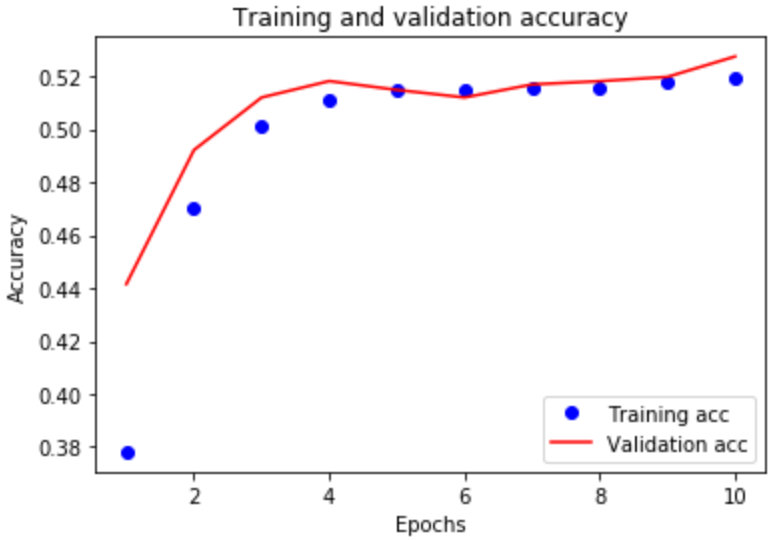
1. Multi-variate Gaussian prob distribution

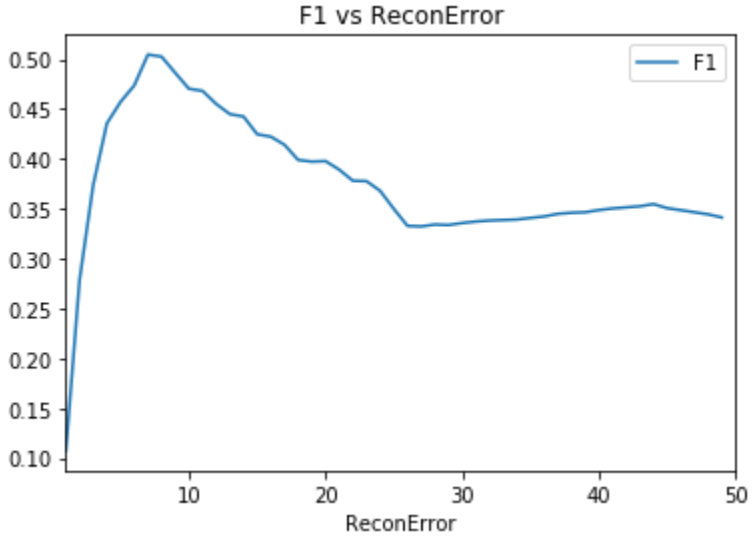



```
# F1 Score
#print("F1 score", round(f1_score(y_valid,pred, average='binary'), 4))
precision,recall,fbeta_score, support  = precision_recall_fscore_support(y_test,pred, average='b
inary')
print("precision ", round((precision), 3))
print("recall ", round((recall), 3))
print("F1 score on Test", round((fbeta_score), 3))
```

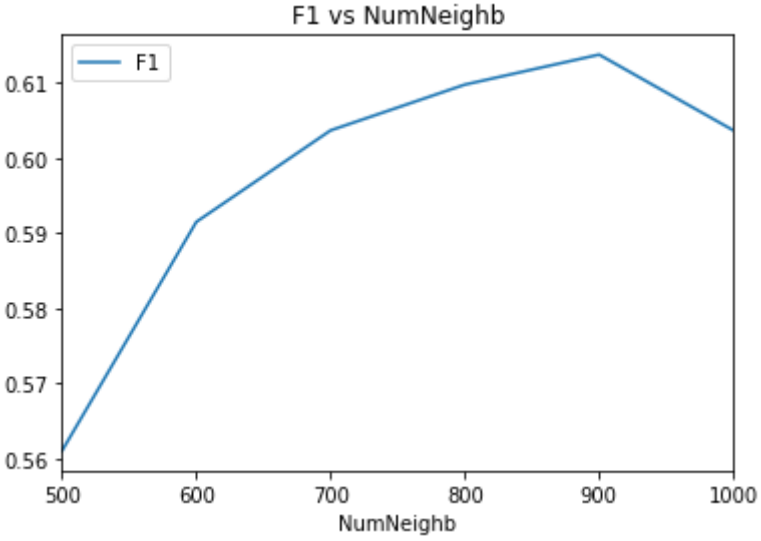
```
precision  0.675
recall  0.758
F1 score on Test 0.714
```

2. Auto Encoders

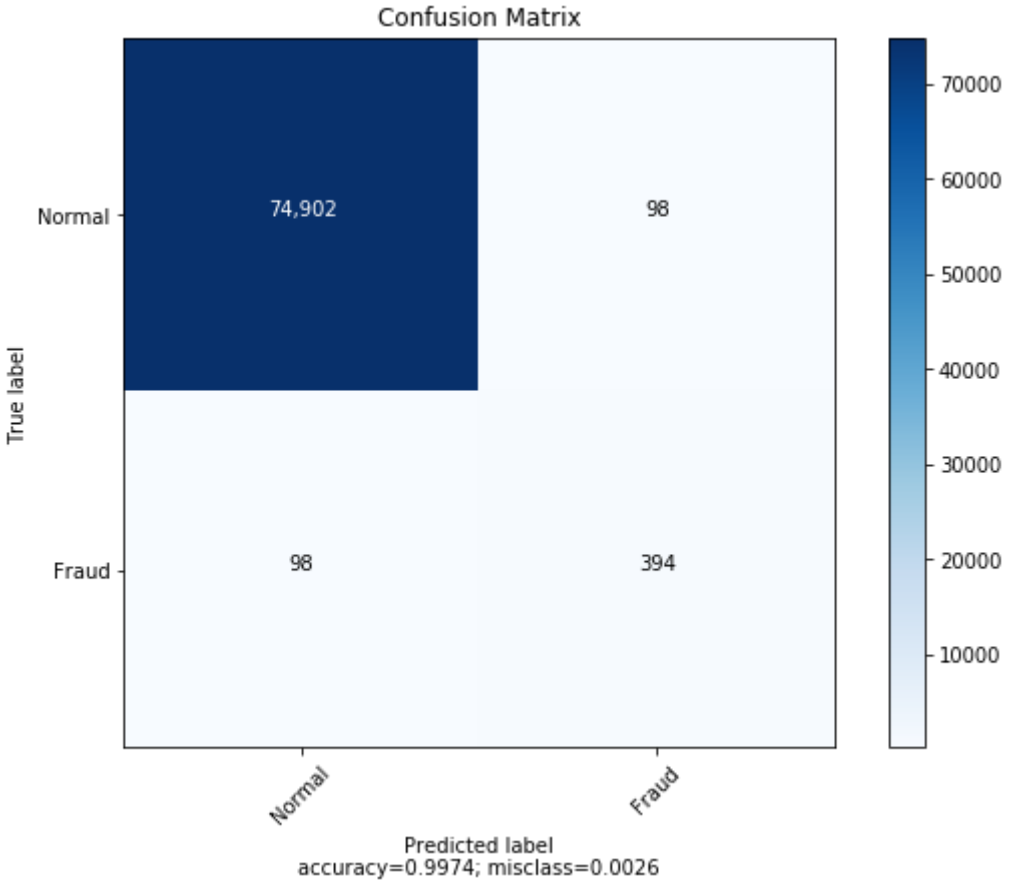


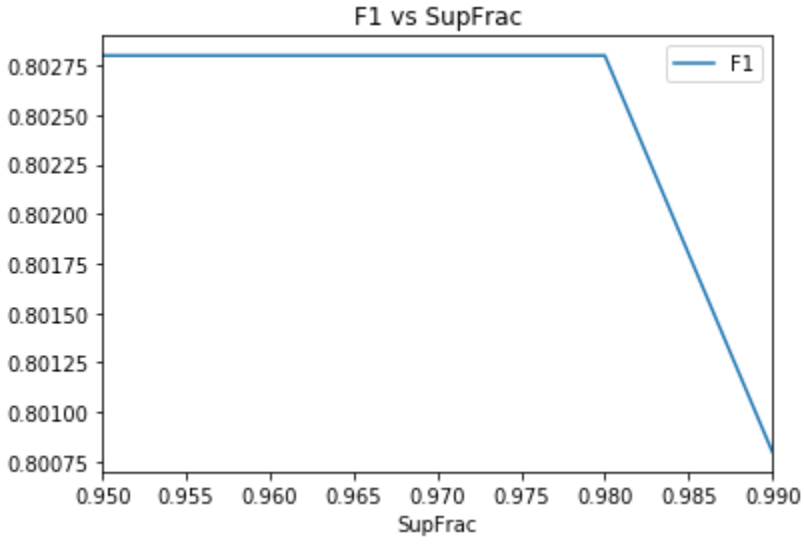


3. Local Outlier Factor (LOF)

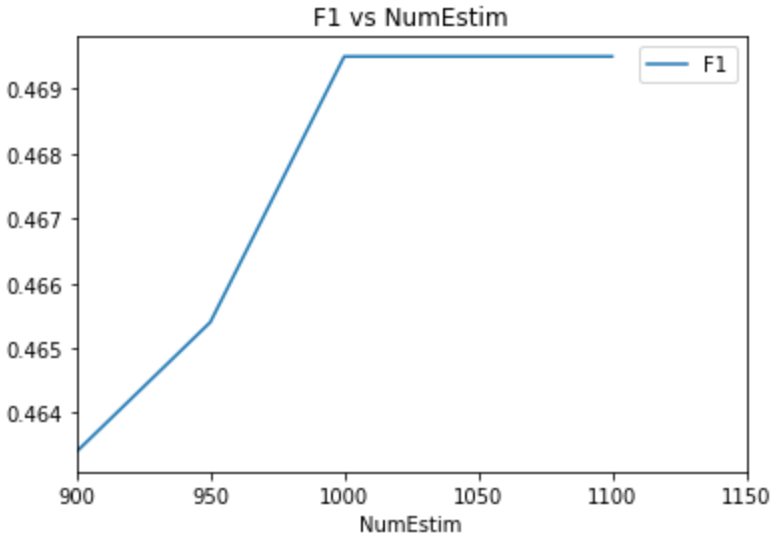


4. Robust Covariance (Elliptic Envelope)

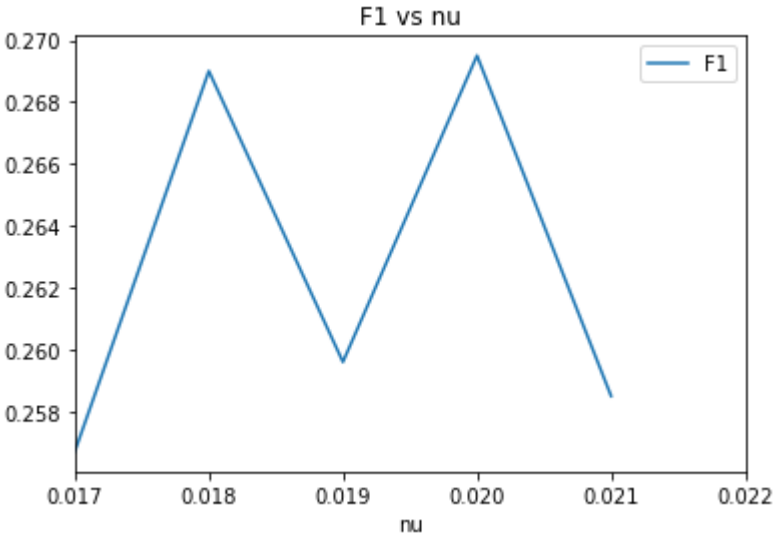




5. Isolation Forest



6. One Class SVM



Conclusion:

The data was split into 3 parts, a train set, a validation set, and a test set. For the first three models, we only used the training dataset and test set. **1.** We started with Random Forest Classifier, for which we obtained an AUC score of 0.868 when predicting the target for the test set. **2.** We followed with an AdaBoost Classifier model, with a lower AUC score (0.852) for the prediction of the test set target values. **3.** We then followed with a CatBoost Classifier, with the AUC score after training five hundred iterations 0.879. **4.** We then experimented with an XGBoost model. In this case, we used the validation set for validation of the training model. The best validation score obtained was 0.987. Then we used the model with the best training step, to predict the target value from the test data; the AUC score obtained was 0.981. **5.** We then presented the data to a

LightGBM model. We utilized both train-validation split and cross-validation to evaluate the model effectiveness to predict 'Class' value, i.e., detecting if a transaction was fraudulent. With the first method, we obtained values of AUC for the validation set around 0.985. For the test set, the score obtained was 0.948. With the cross-validation, we obtained an AUC score for the test prediction of 0.969. For unsupervised methods, we got the F-1 score results as: Multivariate Gaussian prob distribution F1= 0.71, Auto-encoders F1= 0.53 (autoencoders are a type of neural network that is used for unsupervised learning), LOF Local Outlier Factor F1= 0.57 , Robust Covariance (Elliptic Envelope) F1= 0.80, Isolation Forest F1= 0.47, One Class SVM F1= 0.27.

Discussion:

For this substantial imbalanced dataset, the supervised method we utilized outperformed the unsupervised method with way better accuracy. Supervised learning model produces an accurate result. Unsupervised learning model may give less accurate result as compared to supervised learning. supervised learning uses labeled input and output data, while an unsupervised learning algorithm does not. In supervised learning, the algorithm "learns" from the training dataset by iteratively making predictions on the data and adjusting for the correct answer. Unsupervised Anomaly Models can serve as newly approach to the problem is called outlier detection. Also, most of the real-world datasets come without classified labels, and unsupervised methods play a significant role in detecting anomaliesAssume the fraud detection data are just points distributed in n-dimensional space and can be transformed to a Gaussian. Suppose there is a delta between the behavior of fraudsters and regular customers, and this delta is statistically significant. In that case, you should be able to detect it with the right features and even get probabilities with the Null hypothesis test. The unconventional nature of the fraud transaction will result in a data point far away from the centroid of everyday transactions.

Our unsupervised anomaly models will take all the points in the original feature space and use algorithms to return a surjective mapping of all points onto a one-dimensional line called anomaly score. Then, in this lower subspace, it will decide whether or not the attribute is anonymous and produce a label.

- For the Multivariate Normal Distribution, a multivariate distribution describes the probabilities for a group of continuous random variables, particularly if the individual variables follow a normal distribution. The Gaussian distribution arises in many different contexts and can be motivatedfrom a variety of different perspectives. It is applicative analyzing the relationship between multiple normally distributed variables. However, if the random variable components in the vector are not normally distributed themselves, the result is definitely not multivariate normally distributed.

- The Autoencoder method has a particular advantage over traditional ML techniques such as principal component analysis for dimensionality reduction. They can represent data as

nonlinear representations and work particularly well in feature extraction. But autoencoders rely on a considerable amount of "clean data," once there is insufficient training data or training the wrong use case, the performance of this unsupervised learning method will be significantly discounted.

- Local outlier factor (LOF) values identify an outlier based on the local neighborhood. It gives better results than the global approach to find outliers. In other words, the Local Outlier Factor (LOF) is a score that tells how likely a specific data point is an outlier/anomaly. However, due to the LOF is a ratio, it is tough to interpret ,and there is no specific threshold value above which a point is defined as an outlier.

- Robust Covariance for Anomaly Detection is to detect anomalies and outliers by means of the Mahalanobis distance. Robust covariance methods are based on the fact that outliers lead to an increase of the values (entries) in Σ, making the spread of the data apparently larger. Consequently, $|\Sigma|$ (the determinant) will also be larger, which would theoretically decrease by removing extreme events. Things will be different when low data quality and the presence of noise bring a colossal challenge to outlier detection. The "noise" may distort the data, blurring the distinction between ordinary objects and outliers.

- A technique used to detect anomalies is named as Isolation Forests. The algorithm works because anomalies are data points that are far from regular ones. As a result, anomalies are susceptible to a mechanism called isolation. The algorithm isolates points by randomly selecting a feature and then splitting on values between the maximum and minimum values of the attribute until the points are separated from each other. However, the Isolation Forest algorithm has disadvantage in detecting local anomaly point, which affects the accuracy of algorithm.

- One-class support vector machine (OC-SVM) can serve as an effective and outstanding unsupervised method to find data samples that do not follow certain patterns of the majority of data points. Nevertheless, extreme sensitivity to the presence of outliers and noises in the training set is considered an important drawback.

Reference:

1. https://www.kaggle.com/mlg-ulb/creditcardfraud
2. https://www.kaggle.com/code/drscarlat/compare-6-unsupervised-anomaly-detection-models
3. Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson and Gianluca Bontempi. Calibrating

Probability with Undersampling for Unbalanced Classification. In Symposium on Computational Intelligence and Data Mining (CIDM), IEEE, 2015

4. Dal Pozzolo, Andrea; Caelen, Olivier; Le Borgne, Yann-Ael; Waterschoot, Serge; Bontempi, Gianluca. Learned lessons in credit card fraud detection from a practitioner perspective, Expert systems with applications,41,10,4915-4928,2014, Pergamon
5. Dal Pozzolo, Andrea; Boracchi, Giacomo; Caelen, Olivier; Alippi, Cesare; Bontempi, Gianluca. Credit card fraud detection: realistic modeling and a novel learning strategy, IEEE transactions on neural networks and learning systems,29,8,3784-3797,2018, IEEE
6. Principal Component Analysis, Wikipedia Page, https://en.wikipedia.org/wiki/Principal_component_analysis
7. ROC-AUC characteristic, https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve

```
Training until validation scores don't improve for 50 rounds
[50]    training's auc: 0.969446    valid_1's auc: 0.959516
[100]   training's auc: 0.975391    valid_1's auc: 0.962895
Early stopping, best iteration is:
[72]    training's auc: 0.976315    valid_1's auc: 0.967322
Fold  1 AUC : 0.967322
Training until validation scores don't improve for 50 rounds
[50]    training's auc: 0.976681    valid_1's auc: 0.952957
Early stopping, best iteration is:
[49]    training's auc: 0.976834    valid_1's auc: 0.953152
Fold  2 AUC : 0.953152
Training until validation scores don't improve for 50 rounds
[50]    training's auc: 0.974153    valid_1's auc: 0.963634
[100]   training's auc: 0.977403    valid_1's auc: 0.97551
Early stopping, best iteration is:
[90]    training's auc: 0.976759    valid_1's auc: 0.976591
Fold  3 AUC : 0.976591
Training until validation scores don't improve for 50 rounds
[50]    training's auc: 0.970799    valid_1's auc: 0.975831
Early stopping, best iteration is:
[46]    training's auc: 0.97193 valid_1's auc: 0.97691
Fold  4 AUC : 0.976910
Training until validation scores don't improve for 50 rounds
[50]    training's auc: 0.972326    valid_1's auc: 0.987801
[100]   training's auc: 0.969539    valid_1's auc: 0.9881
Early stopping, best iteration is:
[81]    training's auc: 0.970589    valid_1's auc: 0.989584
Fold  5 AUC : 0.989584
Full AUC score 0.969356
```