Imbalance Dataset Approaches and Supervised Learning Comparative Analysis: Credit Card Fraud Detection

Antonio Mpembe Franco
Faculty of Engineering, Environment & Computing
Coventry University
Coventry, England
mpembefraa@uni.coventry.ac.uk

***Abstract*—This paper focuses on approaches to imbalance dataset such as outlier detection, novelty detection, weighted classification and hybrid sampled data classification. Exploring these approaches on the credit card fraud detection problem. With many techniques used from exploratory data analysis such as data cleansing and data analysis to machine learning methods supervised and unsupervised. Exploring explicit generalization models and Instance-based algorithm and optimizing their hyperparameters.**

Keywords—Class imbalance, classification, hyperparameter optimization, outlier detection, novelty detection, supervised, unsupervised, ensemble.

I. INTRODUCTION

As a society, there is a desire for people to gain fairly. And fraud is still unfair even done legally. This intentional deception in monetary gain has a financial and reputational impact on many businesses. Organizations typically lose 5% of their revenue, and the fraud by owners or executive cost is 9 times more than employee fraud. The industries most affected are banking, manufacturing, and government (Fraud, 2020). With the convenience of the credit card to pay merchants for good and services, this card that differs from a charge card, requiring the debt to be repaid with interest, has gained popularity. There were 1.122 billion credit cards in circulation in 2018 in the U.S (Credit card, 2020). Credit card fraud has a big prevalence, mainly for the payments via the internet with 73% of all card fraud. In 2016, fraud losses estimated to be 1.32 billion in the Single Euro Payments Area (SEPA) (Executive summary, 2018). Card fraud is sometimes prevented or discovered due to fraud prevention, which prevents the occurrence of fraudulent transactions, and fraud detection, where fraudulent transactions are identified after the event. This layers to block and identify card fraud applies machine learning models and other measures.

The paper focuses on fraud detection, credit card fraud detection. This problem is related to anomaly detection, which is also referred to as outliers or noise. The anomaly detection is a difficult problem to solve, meanly because the purpose is to identifier items or events that differ from the rest of the data. Normally anomaly detection problems have an imbalance dataset because the purpose is normally to identify rare items (Anomaly detection, 2020).

II. APPROACHES

There are many ways to approach credit card fraud detection, some of which are beyond the scope of this paper, like neural network and others. The paper will explorer approaches to an imbalanced dataset, such as outlier detection, novelty detection, class sensitive and hybrid sampling. The paper will start with a statistical approach for outlier detection, followed by an unsupervised approach for novelty detection, finally with supervised approaches from case-sensitive support vector machine and hybrid sampling. On the hybrid sampling, the paper will explorer and compare many types of algorithms and select one of each type. And last applying a boosting algorithm. Exploring as well as different approaches for hyperparameters optimization such as grid search, Bayesian optimization and gradient-based optimization.

At the same time, the parametric and nonparametric algorithm will be explored. Parametric algorithms are ones that independently on the amount of the training data it will summarize the data with a fixed size set of parameters. And nonparametric are ones that focus more on patterns, for this, there is no need to worry too much on selecting the right features (Brownlee, 2020).

Table 1
Explicit generalization models

Algorithm	Type	Approach
One-class support vector machine	Linear model, Nonparametric	Unsupervised
Support vector machine	Linear model, Nonparametric	Supervised
Naïve Bayes	Probabilistic, Parametric	Supervised
Decision Tree	Tree-based, Nonparametric	Supervised

Note. These are models with the ability to adapt properly to unseen data (Machine learning crash course, 2020).

Table 2
Instance-based learning algorithms

Algorithm	Type	Approach
k-nearest neighbours	Proximity-based, Nonparametric	Supervised

Note. Algorithms that compare new data with data seen in training, which are stored in memory (Instance-based learning, 2020)

III. Related Works

The paper compares credit card fraud detection from many points of view. Giving examples for each type. The focus is on a supervised scenario and trying to compare the types as fair as possible.

There are many related works this problem with many approaches, such as outlier detection. From papers to libraries uncovering ways to deal with this problem. The related works are a starting point, to take good practices and apply them.

Starting with libraries in python for machine learning problems. There is one library that comes to mind, a very popular library used in this project as well, Scikit-learn. Scikit-learn has a section dedicated to novelty and outlier detection. This section is mainly about the unsupervised approach for detecting outliers and novelties (Novelty and Outlier Detection, 2020). The novelty is an unsupervised learning way to solve this problem. This paper will demonstrate the novelty detection approach as well. As well as supervised approach work for this problem, real-life situations are very difficult to obtain labels for the datasets especially when this task is performed by humans. But to find more about a comparison between unsupervised learning methods, reading the paper, *Fraud Detection in Credit Card Data using Unsupervised Machine Learning Based Scheme* (Dwivedi, 2020) would be a nice starting point.

A statistical approach to measure distance is very important, not only important to find outliers, but also an important tool for proximity-based algorithms, sampling data and other areas of machine learning. So, for the statistical point of view to find multivariate outliers, one of the algorithms used for this purpose is Mahalanobis distance which the paper, *"Locally centred Mahalanobis distance: A new distance measure with salient features towards outlier detection"*, demonstrates that the algorithm and other techniques add towards better detection and understanding of outliers (Roberto Todeschini, 2013).

Because this dataset is imbalanced it is important to look to the ways it can balance the data for the models. Sampling data is intuitively a very good way to deal with imbalanced data. The paper, *"Performance Evaluation of Class Balancing Techniques for Credit Card Fraud Detection"* (D. S. Sisodia, 2017), Compares 9 sample techniques applied in the same dataset of this paper. With the result obtained in the experiments, they concluded that SMOTE ENN had the best performance as the oversampling technique and TL for undersampling technique.

IV. Evaluation

The algorithms are evaluated on a laptop Xiaomi Mi Notebook Pro i5 2017.

Table 3
The laptop specification

Processor	Storage	Memory
Intel Core i5-8250U	Samsung PM961 MZVLW256HEHP, 256 GB	7.89 GB, DDR4 2400MHz

Note. The computer contains other components these are the main components for the experiments.

- Confusion matrix

A confusion matrix will be presented, this is a cross-tabulation table with 2 dimensions presenting the actual class and the predicted class (Confusion matrix, 2020).

Table 4
Confusion matrix description

		Actual class	
		Positive(P)	Negative(N)
Predicted class	Positive (P)	true positive (TP)	false positive (FP)
	Negative (N)	false negative (FN)	true negative (TN)

Note. TP eqv. with a hit, TN eqv. with correct rejection, FP eqv. with a false alarm, FN eqv. with a miss. All of this depends on which class is the focus

- Derivations from a confusion matrix

The classification report will be presented, this focuses on the precision "$\frac{TP}{(TP+FP)}$", this shows the per cent of the predictions are correct and recall "$\frac{TP}{(TP+FN)}$", this shows the actual class are correct. And the f1-score "$2 * \frac{precision * recall}{precision + recall}$" which is the harmonic mean of recall and precision. Finally, the macro average which is the average of each field and weighted average which is the weighted average of the fields.

The area under the ROC curve "$\int_{x=0}^{1} TPR(FPR^{-1}(x))$" will be presented, true positive rate (TPR) and false negative rate (FPR), this will show how much the classifier is capable to differentiate between the 2 classes.

- Log Loss (Binary cross-entropy)

This is a negative log average of the sum of the difference between the predicted probability to actual probability "$H_q(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i * log(p(y_i)) + (1 - y_i) * log(1 - p(y_i))$" (Godoy, 2018).

V. Exploratory Data Analysis

A. Data acquisition

Dataset obtained from Kaggle uploaded by "Machine Learning Group – ULB" (ULB, 2017). This dataset is made off transactions by European credit cardholders from September 2013. The dataset has 284807 examples with 31 features. On this dataset, dimensionality reduction was applied, to protect user identities and sensitive features. Features from V1 to V28 are a result of the principal component analysis (PCA). Only 3 features were not transformed. The feature 'Amount' is the amount of money of the transaction and the feature 'Time' is the seconds between each transaction and the first transaction in the dataset. The dataset is imbalanced on the feature "Class", group in two categories fraud taking the value "1" and normal transaction taking the value "0". The dataset has a distribution of 492 examples of frauds out of 284,807 total transactions.

Table 5
Dataset Features

Feature	Description	Data type

Time	Seconds between each transaction and the first transaction	Numeric
V1		Numeric
….	….	….
V28		Numeric
Amount	Amount of money of the transaction	Numeric
Class	Categories fraud transaction with the value "1" and normal transaction with the value "0"	Categorical

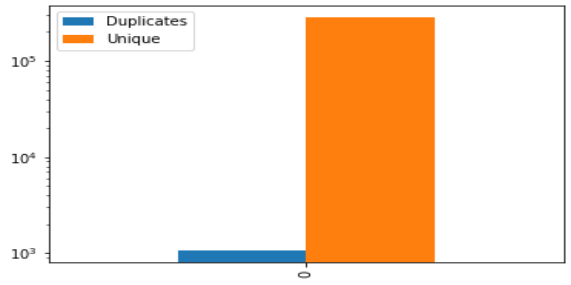
Note. The 3 points describe features "V" and incremental number from 2 to 27 with a numeric data type.

B. *Data cleansing*

It is important to get an overview of the data. Finding missing data, wrong data type value, duplicates examples and constant features. The dataset does not contain any missing value nor does it contain any wrong data type value. No constant feature was found in the dataset. But the dataset does contain duplicated examples. For the duplicated instances, there may be a need to remove them because they can cause overfitting.

Figure 1

Distribution of duplicates and unique examples



Note. Duplicates: 1081, Unique: 283726 instances

C. *Data Distribution*

This is a general behaviour of each feature how there are spread from instance to instance. They are many tools that can be used to measure the variability of a feature, such as range, interquartile range, variance, standard deviation, and others (Frost, 2020). For example, the variance is the average of the difference squared between the values and the mean, which is the average of the total sum of all values.

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}$$

It is also necessary to look at the distribution of the two classes. Because the variance depends on the amount of data, the variance of a certain feature may mislead people. The variance may lead people to suppose something that is not true.

Table 6

The variance of 2 main features relative they classes

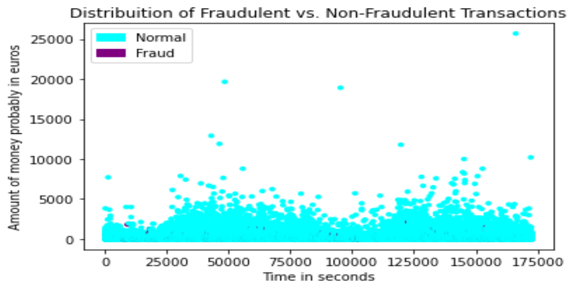
Feature	Fraud transaction	Normal transaction
Time	2.253928e+09	2.365478e+09
Amount	6.266252e+04	6.770979e+04

Note. This variance is only for unique instances

And from the "Table 2", the variance of the features, "Time" and "Amount" shows that there is a difference between the 2 classes. The variance of all the data would favour the majority class. The variance of all the data can be misleading when the dataset is imbalance because it does not reflect each class of the dataset. The variance of the whole dataset can be seen on the distribution of all the dataset on the "Figure 2".

Figure 2

Distribution of data between "Class" "0" and "1"



Note. Normal: 283253, Fraud: 473 instances. Variance for "Time": 2.254450e+09 and for "Amount": 6.267726e+04

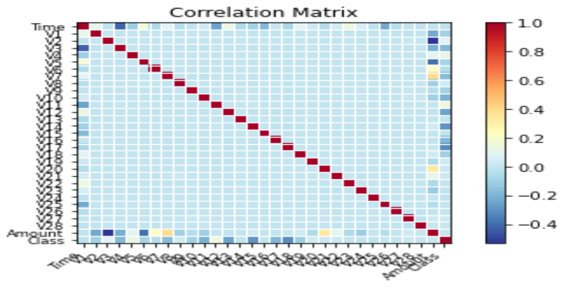
D. *Relation between features*

It is important to understand the relation between features. This knowledge helps to understand better the dataset, how each feature relates to the other. This is also helpful for recognizing the most important features of the dataset. The use of covariance or correlation helps to observer the relation between the features. Conceptually the 2 are almost identical because correlation is simply a normalized form of covariance (Kumar, 2018).

$$\text{Corr}\,(X, Y) = \frac{\text{Cov}(X, Y)}{\sigma_y * \sigma_x}$$

Figure 3

The Correlation Matrix between all the features



Note. Feature "V17" has the highest correlation with "Class"

VI. EXPERIMENTS

There are 2 types of outliers, the univariate and multivariate. Univariate outliers can be found when analysing the distribution of one feature. And multivariate outliers can be found when looking at 2 or more features together (Univariate and Multivariate Outliers, 2020). There is also different type of outliers in the way they differ from the rest of the data, where there is a strong and weak outlier. Strong outliers are the ones that differ them most from the rest of the data, and weak outliers are normally difficult to find. The outliers in this problem are weak multivariate outliers (Statistical Outliers: Detection and Treatment, 2017).

A. *Statistical approach*

1) *Outlier detection*

The statistical approach to find outliers are normally for the strong outlier in a single feature where the use of z-score, interquartile ranges and others method may prevail. For multivariate outliers, Euclidean distance can be used. But the problem with this approach is when features are correlated because it does not consider this.

a) Mahalanobis distance

The solution is to use the Mahalanobis distance instead of Euclidean distance. This transform features into uncorrelated variables then calculate the Euclidian distance (Prabhakaran, n.d.).

$$D_m(\vec{x}) = \sqrt{(\vec{x} - \vec{\mu})^T S^{-1}(\vec{x} - \vec{\mu})}$$

The "$\vec{x}$" is the vector of data, the "$\vec{\mu}$" is the mean of each feature of data and "S" is the covariance matrix. And the diagonal values of the matrix are the measure of the Euclidian distance of each example. Now, z-score can be applied.

Z-score also is known as the standard score is the difference between a value and the population mean divided by the population standard score (Standard score, 2020).

$$z = \frac{x - \mu}{\sigma}$$

Because matrix operations are heavy on memory, only 0.12 per cent of all the data was selected for the Mahalanobis distance.

Table 7

Z-score and Mahalanobis error matrix

Normal	3356	30
Fraud	45	16

Note. Error matrix is just another name for the confusion matrix

Table 8

Z-score and Mahalanobis classification report

	precision	recall	f1-score	support
Normal "0"	0.99	0.99	0.99	3386
Fraud "1"	0.35	0.26	0.30	61
Accuracy			0.99	3447
Macro avg	0.67	0.63	0.80	3447
Weighted avg	0.98	0.98	0.98	3447

Note. This report is hand calculated from the result of the error matrix

B. Unsupervised

1) Novelty detection

There are many sources for outliers, and generally, outliers are associated with errors. Sometimes there are not from errors, such as for the credit card fraud detection dataset. And these outliers are called novelties (Santoyo, 2017). In the detection of novelties there an interest in detecting outliers in a new sample of data by comparing it to the training sample with no outliers.

a) One-class support vector machine

One class SVM is an unsupervised support vector machine for novelty detection. Hyperparameter tuning was not used.

And support vector machine (SVM) is a linear model for classification and regression problems. For this problem it will be used as a linear classifier, meaning that the classification is based on the value of linear combination characteristics and the classes are separated by a line or plane or a hyperplane (Linear classifier, 2020).

SVM map data to a higher-dimensional space by a defined kernel function. The training points for SVM are called support vectors because hyperplanes are defined as the set of points whose dot product with an orthogonal set of vectors is constant (Support vector machine, 2020).

$$\sum_i \alpha_i \mathrm{K}(x_i, x) = constant$$

The "α_i" are parameters of the feature vectors "x_i" and "$\mathrm{K}(x_i, x)$" represent the kernel. The selected kernel for this problem is a radial basis function kernel (Radial basis function kernel, 2020).

$$exp\left(-\frac{\| x - x' \|^2}{2\sigma^2}\right)$$

Table 9

One-class SVM matching matrix

Normal	28751	28763
Fraud	199	274

Note. matching matrix is a confusion matrix for unsupervised algorithms

Table 10

One-class SVM classification report

	precision	recall	f1-score	Support
Normal "0"	0.99	0.50	0.67	57514
Fraud "1"	0.01	0.58	0.02	473
Accuracy			0.50	58460
Macro avg	0.50	0.50	0.50	58460
Weighted avg	0.97	0.50	0.66	58460

Note. This report is hand calculated from the result of the matching matrix

C. Supervised

In supervised learning, the learning methods should be given sufficient knowledge so that it can get an unbiased prediction. And because in fraud detection, there is less fraud compare to the normal transaction, the dataset will be an imbalance.

And to get a good fit for some learning algorithms, they have values that control the learning process. Knowing which value gives the best fit is a tuning problem known as hyperparameter optimization. There are many ways to approach this problem (Hyperparameter optimization, 2020).

1) Class weighted (cost-sensitive algorithm)

There are many ways to solve imbalanced data. One way is to attribute class weights. This helps the current training algorithm to consider the distribution of the classes.

Some algorithms are a cost-sensitive algorithm by using the class weight, this penalizes misclassification. In this case, the balanced class weight was used, this adjusts the weights to the inverse proportion of the class frequency (sklearn.utils.class_weight.compute_sample_weight, 2020).

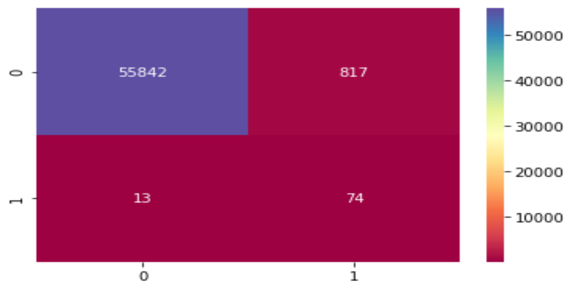
$$\text{weight} = \frac{\text{number of samples}}{(\text{number of classes} * \text{occurrences of the class})}$$

a) Grid search weighted support vector machine

Grid search was used on the support vector machine to tune hyperparameters. And this consists of an exhaustive search through a manually specified set of hyperparameter values. (Hyperparameter optimization, 2020).

Figure 4

Grid search weighted SVM confusion matrix



Note. 0 is for the normal transaction and 1 is for the fraudulent transaction

Table 11

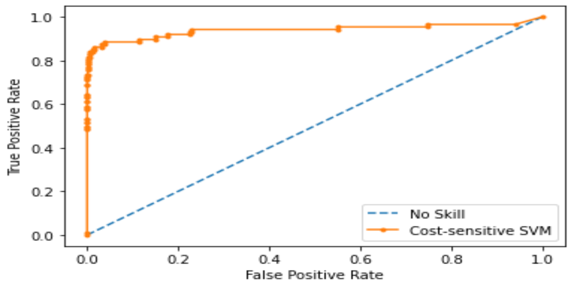
Grid search weighted SVM classification report and log loss

	precision	recall	f1-score	Support
Normal "0"	1.00	0.99	0.99	56659
Fraud "1"	0.08	0.85	0.15	87
Accuracy			0.99	56746
Log loss			0.004	56746
Macro avg	0.54	0.92	0.57	56746
Weighted avg	1.00	0.99	0.99	56746

Note. K-fold Negative log loss: -0.005 with a σ of 0.001 and Accuracy: 0.997 with a σ of 0.001

Figure 5

Grid search weighted SVM ROC curve graphical plot



Note. Graph AUC: 0.940; k-fold AUC: 0.940 with a σ of 0.012

2) *Hybrid Sampling*

One of the ways to deal with imbalance data is with sampling data, undersampling and oversampling data. These are techniques that adjust the class distribution, where undersampling removes samples from the majority class and oversampling add samples to the minority class. Hybrid sampling is applying or combining the 2 techniques.

This problem a combine oversampling technique was used, this is mainly oversampling technique, but it is combined with undersampling meaning that it adds samples to minority class and when necessary removes majority class samples. In this case, a combine over- and under-sampling were used, this uses SMOTE and edited nearest neighbours. The training data was sampled to keep a fair comparison to the weighted class.

SMOTE (synthetic minority oversampling technique) samples new synthetic samples for the minority class by setting neighbourhoods for the examples of this class then generate sample in those neighbourhoods (tyagikartik4282, 2019).

$$x' = x + rand(0,1) * |x - x_{\mathcal{K}}|$$

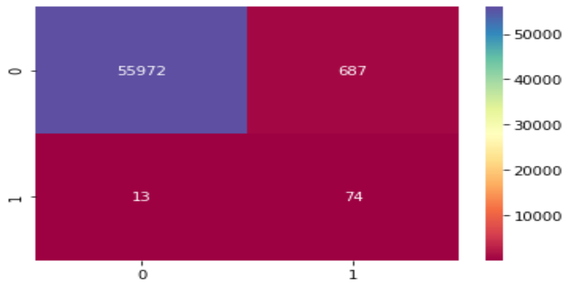
And for ENN (edited nearest neighbours) this technique is based on removing majority class samples closer to the decision boundary, by using the same logic as smote, using the k-nearest neighbour, and comparing if the majority class are closer or in the minority neighbourhood and then removing them (G. Lemaitre, imblearn.under_sampling.EditedNearestNeighbours, 2017).

And in the experiments, the Tomek links technique for undersampling was not removing data. For this reason, random undersampling was used, this removes randomly majority class samples (G. Lemaitre, imblearn.under_sampling.RandomUnderSampler, 2017).

a) Grid search support vector machine

Figure 6

Grid search SVM confusion matrix with sampled data



Note. 0 is for the normal transaction and 1 is for the fraudulent transaction

Table 12

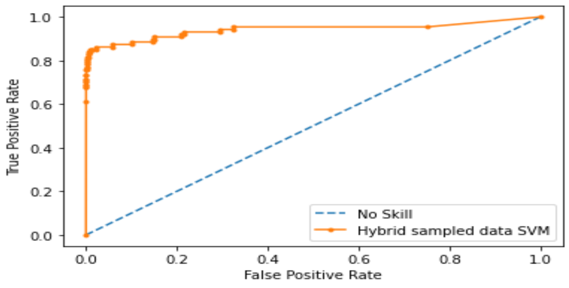
Grid search weighted SVM classification report and log loss

	precision	recall	f1-score	Support
Normal "0"	1.00	0.99	0.99	56659
Fraud "1"	0.10	0.85	0.17	87
Accuracy			0.99	56746
Log-loss			0.039	56746
Macro avg	0.55	0.92	0.58	56746
Weighted avg	1.00	0.99	0.99	56746

Note. K-fold Negative log loss: -0.005 with a σ of 0.003 and Accuracy: 0.999 with a σ of 0.000; runtime 89.5 seconds

Figure 7

Grid search SVM with sampled data ROC curve graphical plot



Note. Graph AUC: 0.942; k-fold AUC: 0.940 with a σ of 0.012

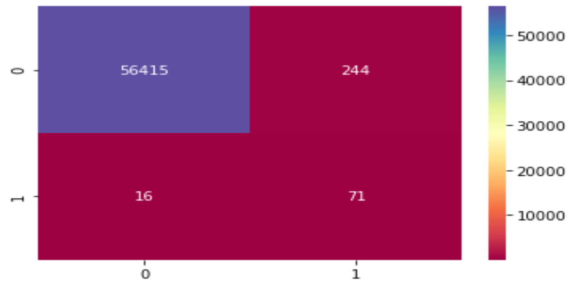
b) Grid search decision tree classifier

The decision tree is the foundation of all tree-based models. Tree-based models use a series of if-then rules to generate predictions from one or more trees (Gross, 2020). In classification trees are built by splitting the source set, this action of splitting is based on a set of rules (Decision tree learning, 2020). The splitting criteria selected by the grid search was Gini impurity, this calculates the sum of probabilities of a randomly selected feature to be incorrectly classified (Tyagi, 2020).

$$Gini\ Index = 1 - \sum_{i}^{n} (P_i)^2$$

Figure 8

Grid search decision tree confusion matrix



Note. 0 is for the normal transaction and 1 is for the fraudulent transaction

Table 13

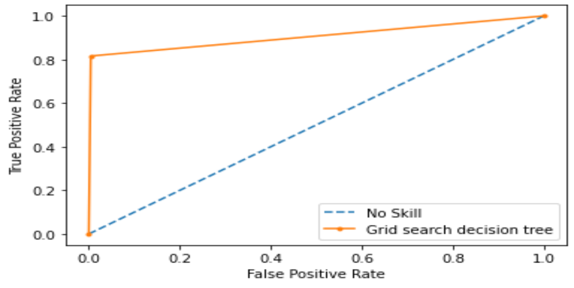
Grid search decision tree classification report and log loss

	precision	recall	f1-score	Support
Normal "0"	1.00	1.00	1.00	56659
Fraud "1"	0.23	0.82	0.35	87
Accuracy			1.00	56746
Log loss			0.158	56746
Macro avg	0.61	0.91	0.86	56746
Weighted avg	1.00	1.00	1.00	56746

Note. K-fold Log loss: -0.041 with a σ of 0.011 and Accuracy: 0.999 with a σ of 0.000; runtime 186.2 seconds

Figure 9

Grid search decision tree with sampled data ROC curve graphical plot



Note. Graph AUC: 0.906; k-fold AUC: 0.831 with a σ of 0.042

c) Bayesian optimized Naïve Bayes

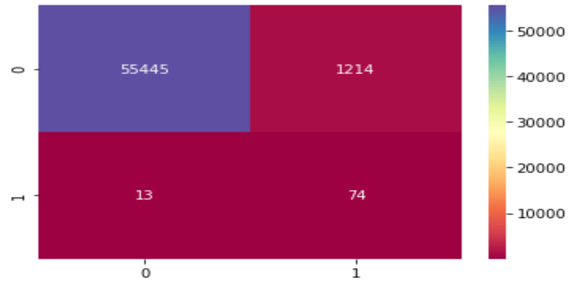
Bayesian optimization is another approach for hyperparameter optimization just like grid search. So, in contrast with grid search and random search that do not take any information of previous training result to select new parameters, Bayesian optimization takes in consideration all previous result to select the next hyperparameter by using the of concept exploration and exploitation. Avoids a brute force approach thus saving time (Hyperparameter optimization, 2020). So, this optimization was applied to a gaussian naïve Bayes algorithm.

Naïve Bayes is a probabilistic classifier, meaning that it predicts the probability of a given test sample belonging to a certain class from the knowledge of a given training sample's probability distribution over the classes (Probabilistic classification, 2019). To be more precise naïve Bayes is a conditional probability model, that applies the Bayes' theorem (Naive Bayes classifier, 2020). For this case, Gaussian Naïve Bayes was used.

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Figure 10

Bayesian optimized Naïve Bayes confusion matrix



Note. 0 is for the normal transaction and 1 is for the fraudulent transaction

Table 14

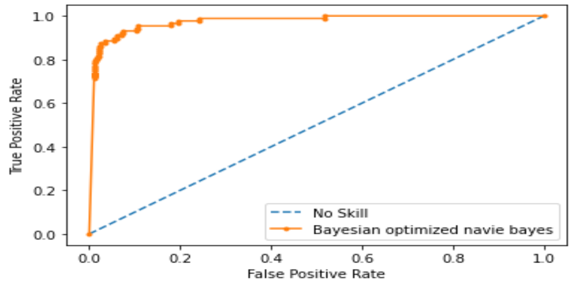
Bayesian optimized Naïve Bayes classification report and log loss

	precision	recall	f1-score	Support
Normal "0"	1.00	0.98	0.99	56659
Fraud "1"	0.06	0.85	0.11	87
Accuracy			0.98	56746
Log loss			0.508	56746
Macro avg	0.53	0.91	0.55	56746
Weighted avg	1.00	0.98	0.99	56746

Note. K-fold Log loss: -0.518 with a σ of 0.031 and Accuracy: 0.978 with a σ of 0.001; runtime 2 seconds

Figure 11

Bayesian optimized Naïve Bayes with sampled data ROC curve graphical plot



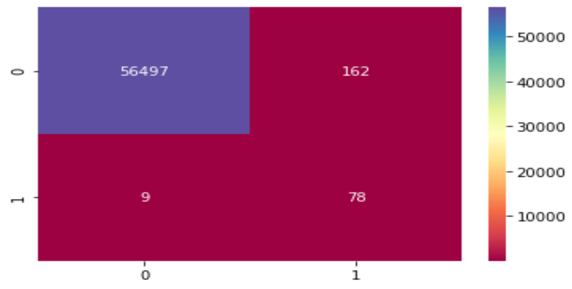
Note. Graph AUC: 0.975; k-fold AUC: 0.974 with a σ of 0.012

d) Bayesian optimized k-nearest neighbours classifier

K-nearest neighbours (KNN) is a proximity-based classifier, meaning that data is classified in the same class based on the idea of proximity. Proximity-based methods can be applied in mainly one of 3 ways. Clustering methods, cluster-based methods, density-based methods, and distance-based methods. K-nearest neighbours are a distance-based method that calculate the distance between point with a statistical approach such as Euclidian or Mahalanobis or others. And new testing samples are compared to samples in memory weighted $^1/_{\mathcal{K}}$ and from their distance, a class is predicted (k-nearest neighbors algorithm, 2020).

Figure 12

Bayesian optimized KNN confusion matrix



Note. 0 is for the normal transaction and 1 is for the fraudulent transaction

Table 15

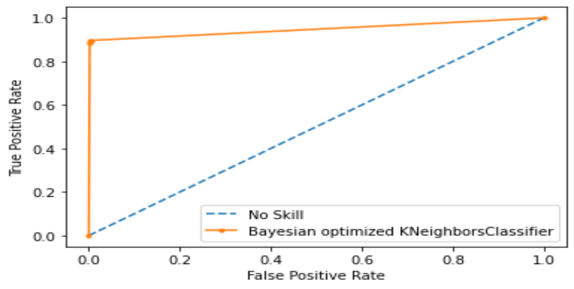
Bayesian optimized KNN classification report and log loss

	precision	recall	f1-score	Support
Normal "0"	1.00	1.00	1.00	56659
Fraud "1"	0.33	0.90	0.48	87
Accuracy			1.00	56746
Log loss			0.069	56746
Macro avg	0.66	0.95	0.74	56746
Weighted avg	1.00	1.00	1.00	56746

Note. K-fold Log loss: -0.014 with a σ of 0.007 and Accuracy: 0.999 with a σ of 0.000; runtime 144.2 seconds

Figure 13

Bayesian optimized KNN with sampled data ROC curve graphical plot



Note. Graph AUC: 0.947; k-fold AUC: 0.912 with a σ of 0.044

D. *Ensemble*

1) Boosting

Boosting works similarly to the bagging methods, by building a family of the same models that are aggregated to obtain a strong learner that performs better. Unlike bagging that aims to reduce variance, boosting aims to reduce bias by giving more weight to failed predicted train samples that were chosen for testing for the subsequent training model (Joseph Rocca, 2019).

a) Extreme gradient boosting

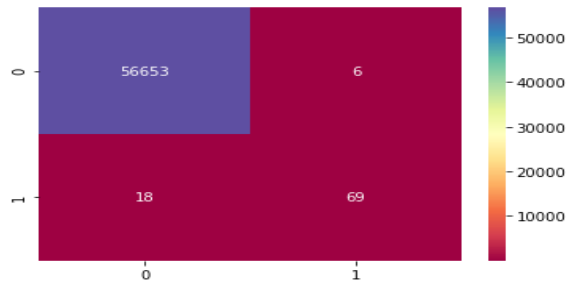
A decision tree model was chosen to be boosted by a gradient boosting, meaning that the problem is cast into a gradient descent problem where the next iteration is fitted with the current error called pseudo-residuals. "First, theoretical gradient descent process over the ensemble model can be written" (Joseph Rocca, 2019).

$$\text{“}\mathcal{S}_{\mathcal{L}}(.) = \mathcal{S}_{\mathcal{L}-1}(.) - \mathcal{C}_{\mathcal{L}} \times \nabla_{\mathcal{S}_{\mathcal{L}-1}} E(\mathcal{S}_{\mathcal{L}-1})(.)\text{”}$$

This will be done with the extreme gradient boost (XGBoost) a boosting technique based on a hyperparameter optimization called gradient-based optimization.

Figure 14

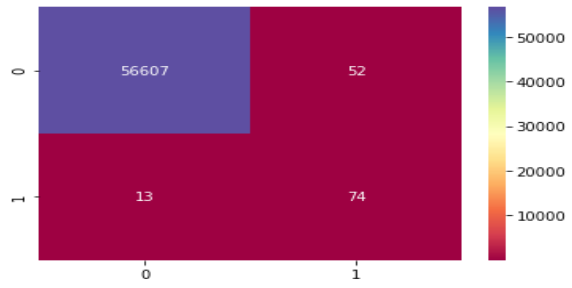
XGBoost confusion matrix



Note. 0 is for the normal transaction and 1 is for the fraudulent transaction

Figure 15

XGBoost with sampled data confusion matrix



Note. 0 is for the normal transaction and 1 is for the fraudulent transaction

Table 16

A lo loss and AUC XGBoost comparison to XGBoost with sampled data

	XGBoost k-fold	XGBoost	XGBoost sampled k-fold	XGBoost sampled
Log-loss	0.003 (0.001)	0.003	0.003 (0.001)	0.005
AUC	0.981 (0.017)	0.985	0.981 (0.017)	0.986

Note. Values inside the parentheses are σ of the scoring metric with k-fold

VII. Conclusion

Different approaches are stated in this paper, so what is the best approach? Well that it is difficult, it depends on the situation, as can be seen from the experiments parametric approach are a faster compare to the nonparametric approach. One of the disadvantages of the parametric approach they require a good selection of feature, this was not a problem for the dataset in this paper because dimensionality reduction was already applied to the dataset. Apart from weighted SVM other nonparametric method did suffer from the reduction of train data as can be seen on the AUC value, which went down significantly. But all nonparametric had better result compare to the parametric method, Naïve Bayes.

Looking on the type of algorithm, the proximity-based approach KNN had less misclassified samples but this is not the whole story. Looking at the probability through the log loss it is clear to see that the linear classifier SVM had the smallest difference between the predicted probability and the ideal probability on average.

Overall, the method with less misclassification is XGBoost, the ensemble one. More often ensemble method does improve the classification. Taking a closer look at the XGBoost comparison between the method with sampled data and without sampled data, it is possible to see that the method with sampled data method has better recall but has the worst precision. This means that it is doing a better a job to recognise fraud but the worst job for the normal transaction. Meaning that oversampling data had a positive impact but undersampling had a negative impact. A problem with sampling data it may lead the algorithm to a bias classification, for example, with oversampling or it may increase false negative by excluding data with undersampling. In this case, undersampling lead to an increase in false negative fraud.

VIII. References

Anomaly detection. (2020, October 26). (Wikipedia) Retrieved November 16, 2020, from Wikipedia: https://en.wikipedia.org/wiki/Anomaly_detection

Brownlee, J. (2020, August 15). *Parametric and Nonparametric Machine Learning Algorithms*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/parametric-and-nonparametric-machine-learning-algorithms/

Confusion matrix. (2020, September 27). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Confusion_matrix

Credit card. (2020, November 14). (Wikipedia) Retrieved November 15, 2020, from Wikipedia: https://en.wikipedia.org/wiki/Credit_card

D. S. Sisodia, N. K. (2017). *Performance evaluation of class balancing techniques for credit card fraud detection*. Chennai: IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI).

Decision tree learning. (2020, November 18). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Decision_tree_learning

Dwivedi, A. K. (2020). *Fraud Detection in Credit Card Data using Unsupervised Machine Learning Based Scheme*. Coimbatore, India: IEEE International Conference on Electronics and Sustainable Communication Systems (ICESC).

Executive summary. (2018). (European Central Bank) Retrieved November 15, 2020, from European Central Bank: https://www.ecb.europa.eu/pub/cardfraud/html/ecb.cardfraudreport201809.en.html#toc1

Fraud. (2020, November 8). (Wikipedia) Retrieved November 15, 2020, from Wikipedia: https://en.wikipedia.org/wiki/Fraud

Frost, J. (2020, August 13). *Statistics By Jim*. Retrieved from StatisticsByJim: https://statisticsbyjim.com/basics/variability-range-interquartile-variance-standard-deviation/

G. Lemaitre, F. N. (2017). *imblearn.under_sampling.EditedNearestNeighbours*. Retrieved from imbalanced-learn: https://imbalanced-learn.org/stable/generated/imblearn.under_sampling.EditedNearestNeighbours.html

G. Lemaitre, F. N. (2017). *imblearn.under_sampling.RandomUnderSampler*. Retrieved from imbalanced-learn: https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.under_sampling.RandomUnderSampler.html

Godoy, D. (2018, November 21). *Understanding binary cross-entropy / log loss: a visual explanation*. Retrieved from towards data science: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

Gross, K. (2020, April 6). *Tree-Based Models: How They Work (In Plain English!)*. Retrieved from data iku: https://blog.dataiku.com/tree-based-models-how-they-work-in-plain-english

Hyperparameter optimization. (2020, November 17). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Hyperparameter_optimization

Instance-based learning. (2020, June 15). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Instance-based_learning

Joseph Rocca, B. R. (2019, April 23). *towards data science*. Retrieved from Ensemble methods: bagging, boosting and stacking: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205

k-nearest neighbors algorithm. (2020, November 29). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Kumar, A. (2018, September 10). *Covariance and Correlation*. Retrieved from Acadgild: https://acadgild.com/blog/covariance-and-correlation

Linear classifier. (2020, September 22). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Linear_classifier

Machine learning crash course. (2020, February 10). Retrieved from Developers Google: https://developers.google.com/machine-learning/crash-course/generalization/video-lecture

Naive Bayes classifier. (2020, November 11). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Naive_Bayes_classifier

Novelty and Outlier Detection. (2020). Retrieved from Scikit-learn: https://scikit-learn.org/stable/modules/outlier_detection.html

Prabhakaran, S. (n.d.). *Mahalanobis Distance – Understanding the math with examples (python)*. Retrieved from Machine learning plus: https://www.machinelearningplus.com/statistics/mahalanobis-distance/

Probabilistic classification. (2019, September 25). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Probabilistic_classification

Radial basis function kernel. (2020, November 23). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Radial_basis_function_kernel

Roberto Todeschini, D. B. (2013). *Locally centred Mahalanobis distance: A new distance measure with salient features towards outlier detection*. Vienna: ELSEVIER.

Santoyo, S. (2017, September 12). *A Brief Overview of Outlier Detection Techniques*. Retrieved from towards data science: https://towardsdatascience.com/a-brief-overview-of-outlier-detection-techniques-1e0b2c19e561

sklearn.utils.class_weight.compute_sample_weight. (2020). Retrieved from Scikit-learn: https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_sample_weight.html#sklearn.utils.class_weight.compute_sample_weight

Standard score. (2020, October 14). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Standard_score

Statistical Outliers: Detection and Treatment. (2017, January 17). Retrieved from Bista Solutions: https://www.bistasolutions.com/resources/blogs/statistical-outliers-detection-treatment/#:~:text=If%20a%20single,a%20weak%20outlier.

Support vector machine. (2020, November 27). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Support_vector_machine

Tyagi, N. (2020, March 24). *Understanding the Gini Index and Information Gain in Decision Trees*. Retrieved from Medium: https://medium.com/analytics-steps/understanding-the-gini-index-and-information-gain-in-decision-trees-ab4720518ba8

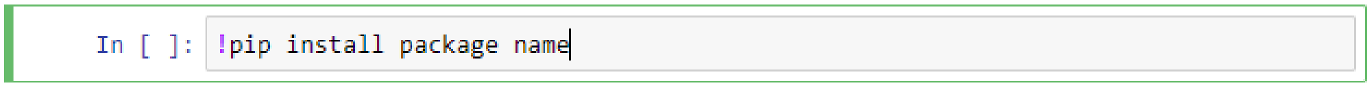
tyagikartik4282. (2019, June 30). *ML | Handling Imbalanced Data with SMOTE and Near Miss Algorithm in Python*. Retrieved from Geeksforgeeks: https://www.geeksforgeeks.org/ml-handling-imbalanced-data-with-smote-and-near-miss-algorithm-in-python/

ULB, M. L. (2017). *Credit Card Fraud Detection*. (Kaggle) Retrieved November 16, 2020, from Kaggle: https://www.kaggle.com/mlg-ulb/creditcardfraud/activity

Univariate and Multivariate Outliers. (2020). Retrieved from StatisticsSolutions: https://www.statisticssolutions.com/univariate-and-multivariate-outliers/

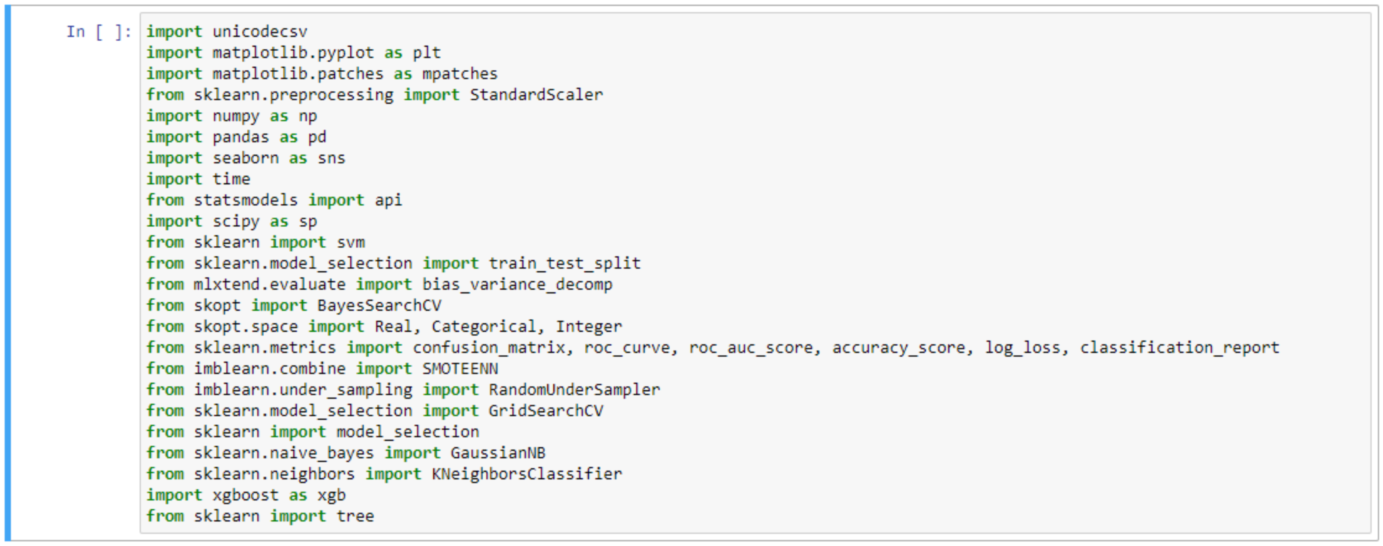
Appendix

1) To install the missing package



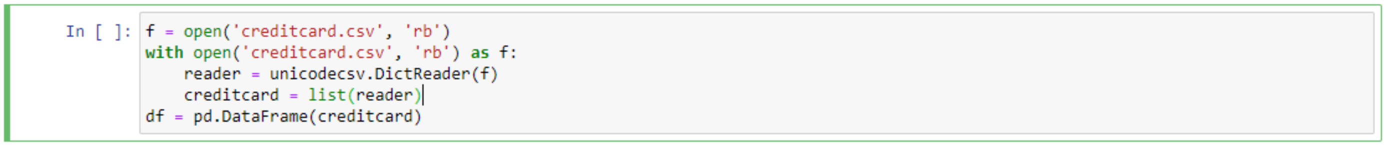
```
In [ ]: !pip install package name
```

2) Import packages



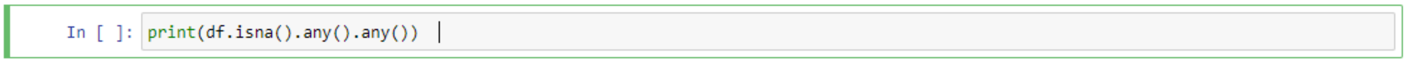
```
In [ ]: import unicodecsv
        import matplotlib.pyplot as plt
        import matplotlib.patches as mpatches
        from sklearn.preprocessing import StandardScaler
        import numpy as np
        import pandas as pd
        import seaborn as sns
        import time
        from statsmodels import api
        import scipy as sp
        from sklearn import svm
        from sklearn.model_selection import train_test_split
        from mlxtend.evaluate import bias_variance_decomp
        from skopt import BayesSearchCV
        from skopt.space import Real, Categorical, Integer
        from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score, accuracy_score, log_loss, classification_report
        from imblearn.combine import SMOTEENN
        from imblearn.under_sampling import RandomUnderSampler
        from sklearn.model_selection import GridSearchCV
        from sklearn import model_selection
        from sklearn.naive_bayes import GaussianNB
        from sklearn.neighbors import KNeighborsClassifier
        import xgboost as xgb
        from sklearn import tree
```

3) Importing data



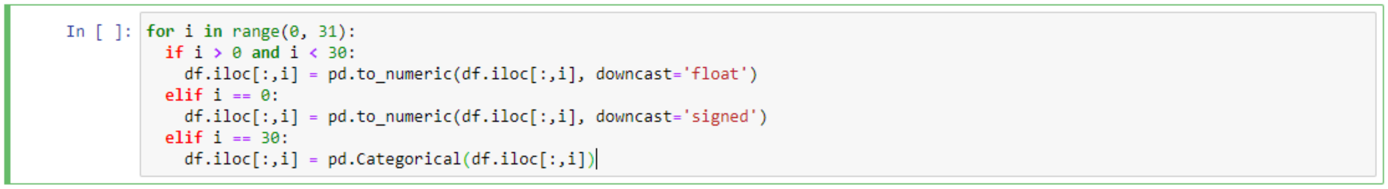
```
In [ ]: f = open('creditcard.csv', 'rb')
        with open('creditcard.csv', 'rb') as f:
            reader = unicodecsv.DictReader(f)
            creditcard = list(reader)
        df = pd.DataFrame(creditcard)
```

4) Checking for null values



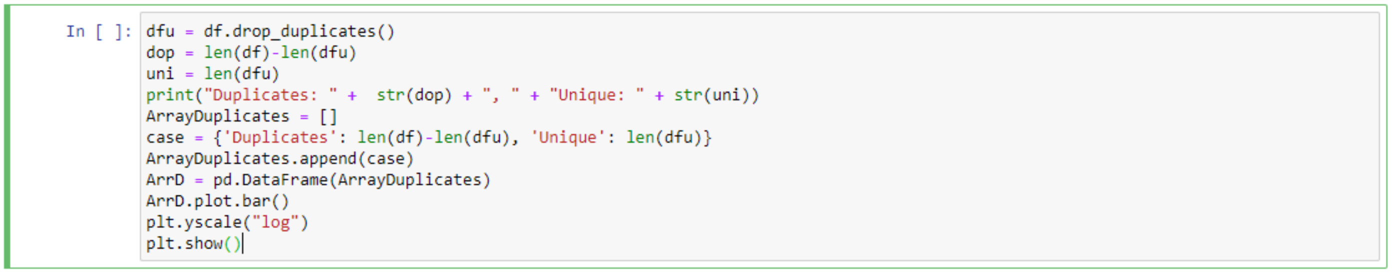
```
In [ ]: print(df.isna().any().any())
```

5) Changing data type and checking for wrong data type



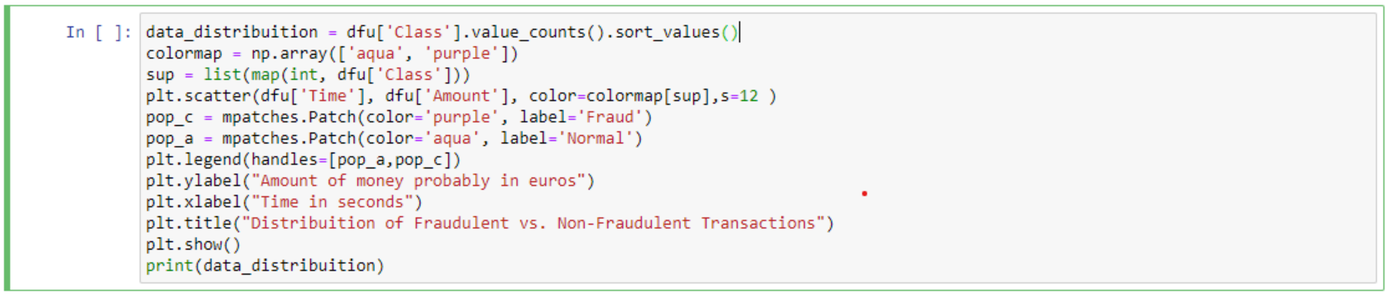
```
In [ ]: for i in range(0, 31):
          if i > 0 and i < 30:
            df.iloc[:,i] = pd.to_numeric(df.iloc[:,i], downcast='float')
          elif i == 0:
            df.iloc[:,i] = pd.to_numeric(df.iloc[:,i], downcast='signed')
          elif i == 30:
            df.iloc[:,i] = pd.Categorical(df.iloc[:,i])
```

6) Duplicated examples



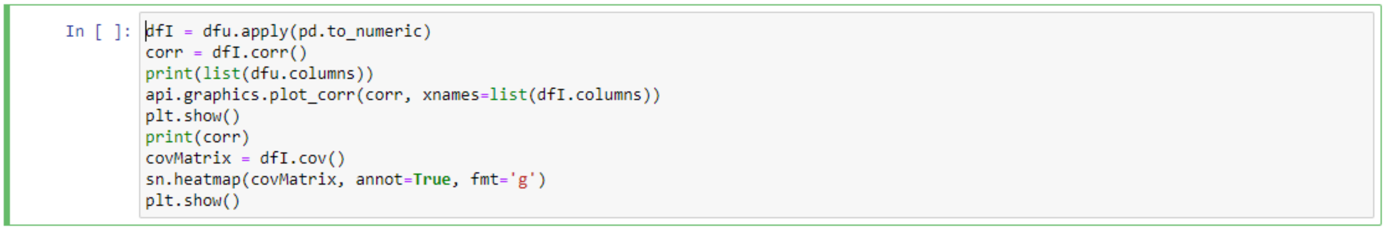
```
In [ ]: dfu = df.drop_duplicates()
        dop = len(df)-len(dfu)
        uni = len(dfu)
        print("Duplicates: " +  str(dop) + ", " + "Unique: " + str(uni))
        ArrayDuplicates = []
        case = {'Duplicates': len(df)-len(dfu), 'Unique': len(dfu)}
        ArrayDuplicates.append(case)
        ArrD = pd.DataFrame(ArrayDuplicates)
        ArrD.plot.bar()
        plt.yscale("log")
        plt.show()
```

7) Data distribution



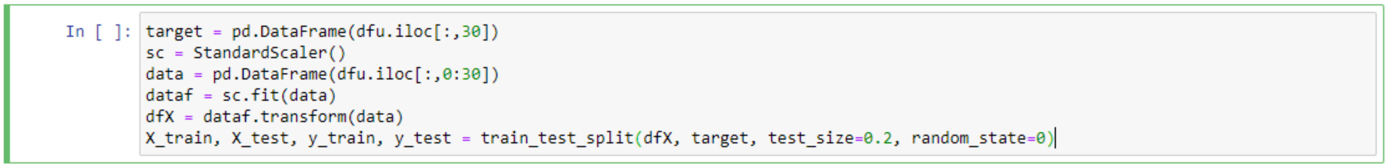
```
In [ ]: data_distribuition = dfu['Class'].value_counts().sort_values()
        colormap = np.array(['aqua', 'purple'])
        sup = list(map(int, dfu['Class']))
        plt.scatter(dfu['Time'], dfu['Amount'], color=colormap[sup],s=12 )
        pop_c = mpatches.Patch(color='purple', label='Fraud')
        pop_a = mpatches.Patch(color='aqua', label='Normal')
        plt.legend(handles=[pop_a,pop_c])
        plt.ylabel("Amount of money probably in euros")
        plt.xlabel("Time in seconds")
        plt.title("Distribuition of Fraudulent vs. Non-Fraudulent Transactions")
        plt.show()
        print(data_distribuition)
```

8) The relation between feature covMatrix



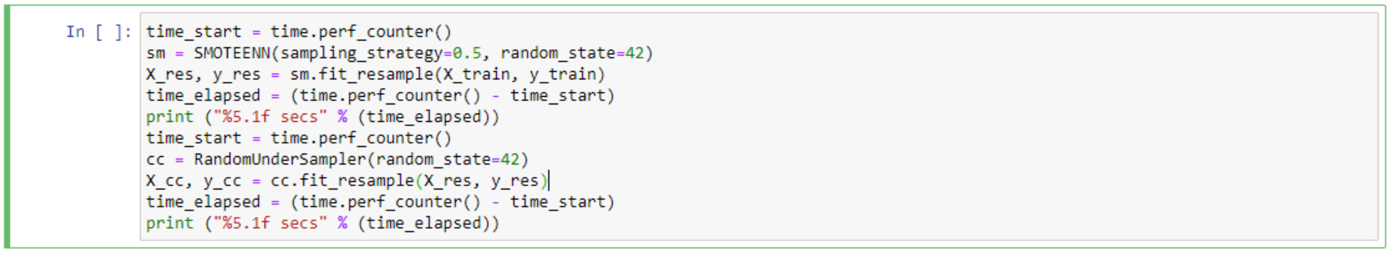
```
In [ ]: dfI = dfu.apply(pd.to_numeric)
        corr = dfI.corr()
        print(list(dfu.columns))
        api.graphics.plot_corr(corr, xnames=list(dfI.columns))
        plt.show()
        print(corr)
        covMatrix = dfI.cov()
        sn.heatmap(covMatrix, annot=True, fmt='g')
        plt.show()
```

9) Normalizing data and splitting data



```
In [ ]: target = pd.DataFrame(dfu.iloc[:,30])
        sc = StandardScaler()
        data = pd.DataFrame(dfu.iloc[:,0:30])
        dataf = sc.fit(data)
        dfX = dataf.transform(data)
        X_train, X_test, y_train, y_test = train_test_split(dfX, target, test_size=0.2, random_state=0)
```

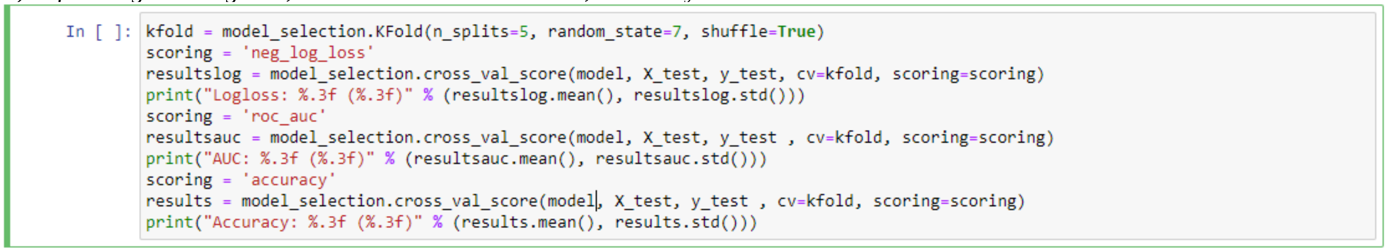
10) Hybrid data sampling SMOTEENN and RandomUnderSampler



```
In [ ]: time_start = time.perf_counter()
        sm = SMOTEENN(sampling_strategy=0.5, random_state=42)
        X_res, y_res = sm.fit_resample(X_train, y_train)
        time_elapsed = (time.perf_counter() - time_start)
        print ("%5.1f secs" % (time_elapsed))
        time_start = time.perf_counter()
        cc = RandomUnderSampler(random_state=42)
        X_cc, y_cc = cc.fit_resample(X_res, y_res)
        time_elapsed = (time.perf_counter() - time_start)
        print ("%5.1f secs" % (time_elapsed))
```

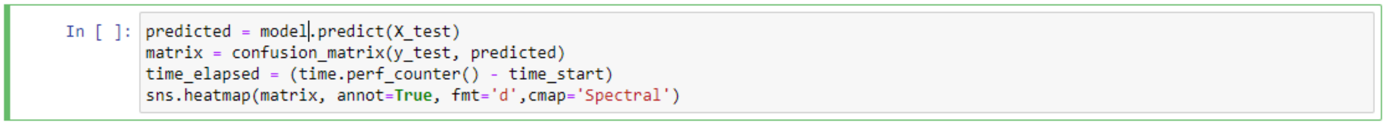
11) Evaluation techniques

a) K-fold negative log loss, area under the ROC curve, Accuracy



```
In [ ]: kfold = model_selection.KFold(n_splits=5, random_state=7, shuffle=True)
        scoring = 'neg_log_loss'
        resultslog = model_selection.cross_val_score(model, X_test, y_test, cv=kfold, scoring=scoring)
        print("Logloss: %.3f (%.3f)" % (resultslog.mean(), resultslog.std()))
        scoring = 'roc_auc'
        resultsauc = model_selection.cross_val_score(model, X_test, y_test , cv=kfold, scoring=scoring)
        print("AUC: %.3f (%.3f)" % (resultsauc.mean(), resultsauc.std()))
        scoring = 'accuracy'
        results = model_selection.cross_val_score(model, X_test, y_test , cv=kfold, scoring=scoring)
        print("Accuracy: %.3f (%.3f)" % (results.mean(), results.std()))
```

b) Confusion matrix



```
In [ ]: predicted = model.predict(X_test)
        matrix = confusion_matrix(y_test, predicted)
        time_elapsed = (time.perf_counter() - time_start)
        sns.heatmap(matrix, annot=True, fmt='d',cmap='Spectral')
```

c) ROC curve graphical plot, plot AUC per cent, Model log loss

```
In [ ]: ns_probs = [0 for _ in range(len(y_test))]
        lr_probs = clf_GS.best_estimator_.predict_proba(X_test)
        lr_probs = lr_probs[:, 1]
        tar = pd.DataFrame(data=y_test, dtype=np.int8)
        loss = log_loss(tar.values.ravel(), lr_probs)
        ns_auc = roc_auc_score(tar.values.ravel(), ns_probs)
        lr_auc = roc_auc_score(tar.values.ravel(), lr_probs)
        print('Log-loss=%.3f' % (loss))
        print('No Skill: ROC AUC=%.3f' % (ns_auc))
        print(': ROC AUC=%.3f' % (lr_auc))
        ns_fpr, ns_tpr, _ = roc_curve(tar.values.ravel(), ns_probs)
        lr_fpr, lr_tpr, _ = roc_curve(tar.values.ravel(), lr_probs)
        pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
        pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Model ROC curve')
        pyplot.xlabel('False Positive Rate')
        pyplot.ylabel('True Positive Rate')
        pyplot.legend()
        pyplot.show()
```

d) Classification report

```
In [ ]: predicted = model.predict(X_test)
        report = classification_report(y_test, predicted)
        print(report)
```

e) Bias-Variance Decomposition

```
In [ ]: tar = pd.DataFrame(data=y_test, dtype=np.int8)
        Ttar = pd.DataFrame(data=y_cc, dtype=np.int8)
        loss, bias, var = bias_variance_decomp(model, X_cc, Ttar.values.ravel(), X_test, tar.values.ravel(),
                                               loss='0-1_loss', num_rounds=50, random_seed=1)
        print('0-1_loss: %.3f' % loss)
        print('Bias: %.3f' % bias)
        print('Variance: %.3f' % var)
```

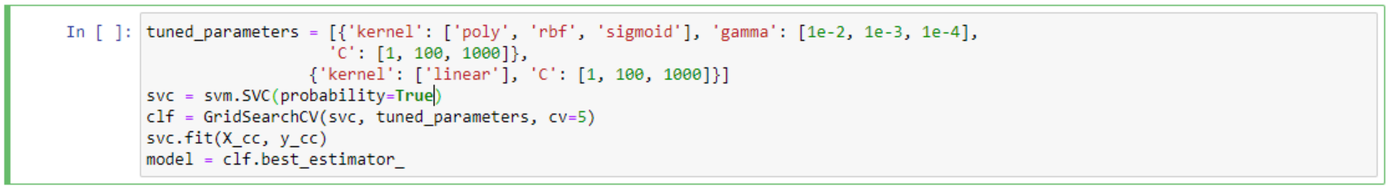
12) *Mahalanobis distance and z-score*

```
In [ ]: df_x = dfu.copy()
        df_x = df_x.sample(frac=0.12, replace=True, random_state=0)
        dt = pd.DataFrame(df_x.iloc[:,0:30])
        dataf = sc.fit(dt)
        df_X = dataf.transform(dt)
        def mahalanobis(x=None, data=None, cov=None):
            x_minus_mu = x - np.mean(dfX)
            if not cov:
                cov = np.cov(np.array(dfX).T)
            inv_covmat = sp.linalg.inv(cov)
            left_term = np.dot(x_minus_mu, inv_covmat)
            mahal = np.dot(left_term, x_minus_mu.T)
            return mahal.diagonal()
        time_start = time.perf_counter()
        df_x['md'] = mahalanobis(x=df_X, data=df_X)
        mean = np.mean(df_x['md'])
        std = np.std(df_x['md'])
        threshold = 3
        for i in df_x['md']:
            z = (i-mean)/std
            if np.abs(z) > threshold:
                if yTarget[j] == "1":
                    w=w+1
                l=l+1
            j=j+1
        print(l)
        print(w)
        print(j)
```

13) *One-class SVM*

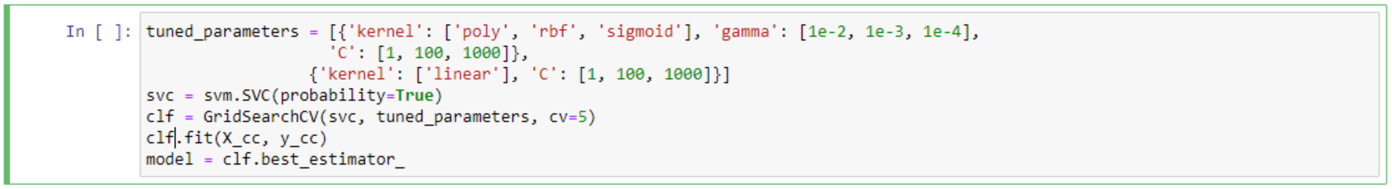
```
In [ ]: from sklearn.svm import OneClassSVM
        X_Class = df_X[dfX['Class'] == '0'].iloc[:,0:30]
        t_y = df_X[dfX['Class'] == '0'].iloc[:,30]
        X_outliers = dfX[df_X['Class'] == '1'].iloc[:,0:30]
        X_train, X_test, y_train, y_test = train_test_split(X_Class, t_y, test_size = 0.2, random_state = 0)
        model = OneClassSVM(gamma=0.001)
        model.fit(X_train)
```

14) *Grid search Class weighted SVM*



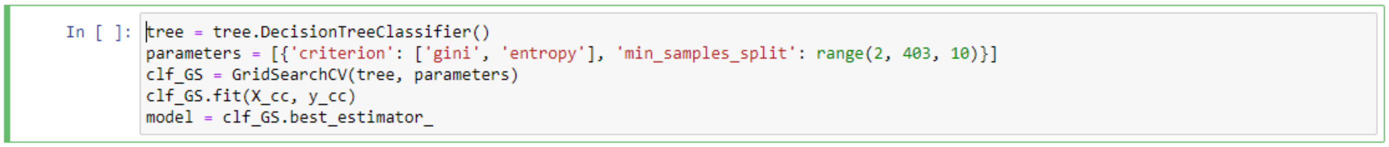
```
In [ ]: tuned_parameters = [{'kernel': ['poly', 'rbf', 'sigmoid'], 'gamma': [1e-2, 1e-3, 1e-4],
                             'C': [1, 100, 1000]},
                            {'kernel': ['linear'], 'C': [1, 100, 1000]}]
        svc = svm.SVC(probability=True)
        clf = GridSearchCV(svc, tuned_parameters, cv=5)
        svc.fit(X_cc, y_cc)
        model = clf.best_estimator_
```

15) *Grid search sampled data SVM*



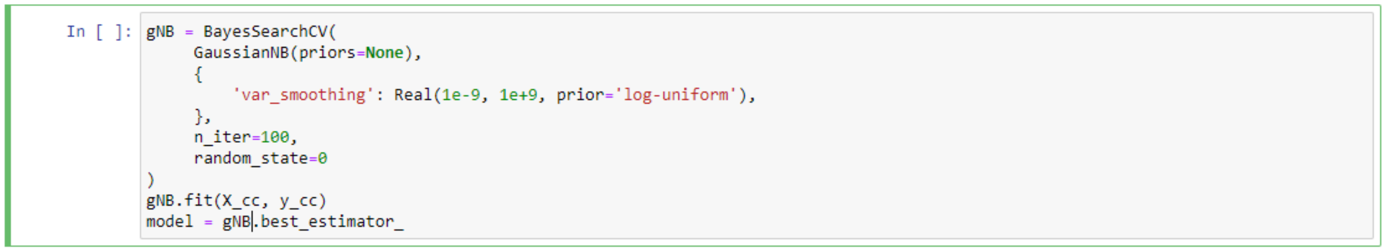
```
In [ ]: tuned_parameters = [{'kernel': ['poly', 'rbf', 'sigmoid'], 'gamma': [1e-2, 1e-3, 1e-4],
                             'C': [1, 100, 1000]},
                            {'kernel': ['linear'], 'C': [1, 100, 1000]}]
        svc = svm.SVC(probability=True)
        clf = GridSearchCV(svc, tuned_parameters, cv=5)
        clf.fit(X_cc, y_cc)
        model = clf.best_estimator_
```

16) *Grid search sampled data decision tree classifier*



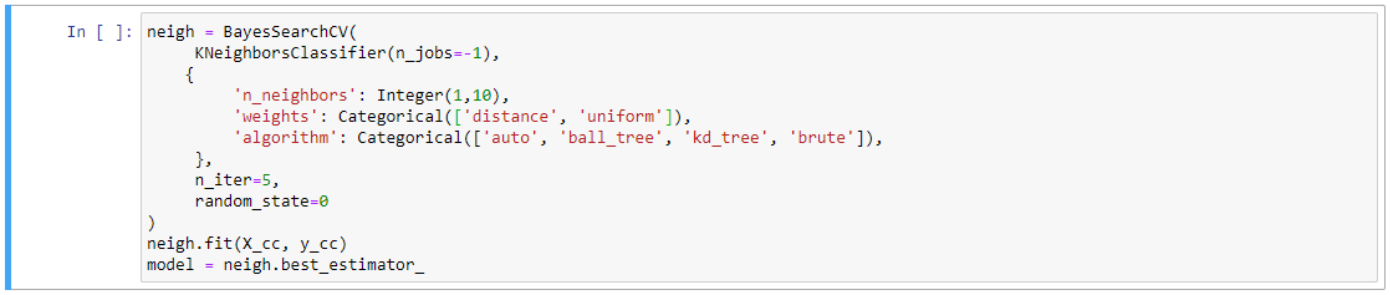
```
In [ ]: tree = tree.DecisionTreeClassifier()
        parameters = [{'criterion': ['gini', 'entropy'], 'min_samples_split': range(2, 403, 10)}]
        clf_GS = GridSearchCV(tree, parameters)
        clf_GS.fit(X_cc, y_cc)
        model = clf_GS.best_estimator_
```

17) *Bayesian search sampled data Gaussian Naïve Bayes*



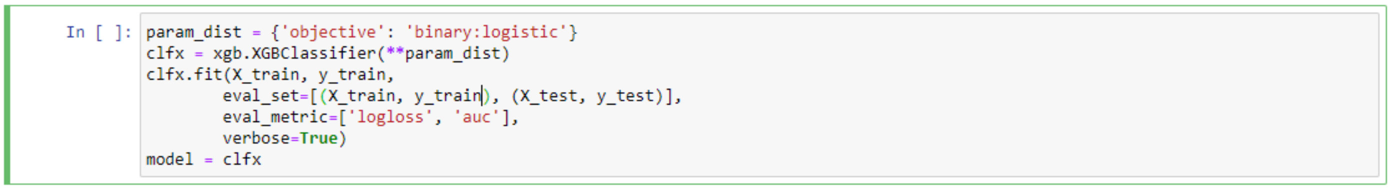
```
In [ ]: gNB = BayesSearchCV(
             GaussianNB(priors=None),
             {
                 'var_smoothing': Real(1e-9, 1e+9, prior='log-uniform'),
             },
             n_iter=100,
             random_state=0
        )
        gNB.fit(X_cc, y_cc)
        model = gNB.best_estimator_
```

18) *Bayesian search sampled data K-nearest neighbour*



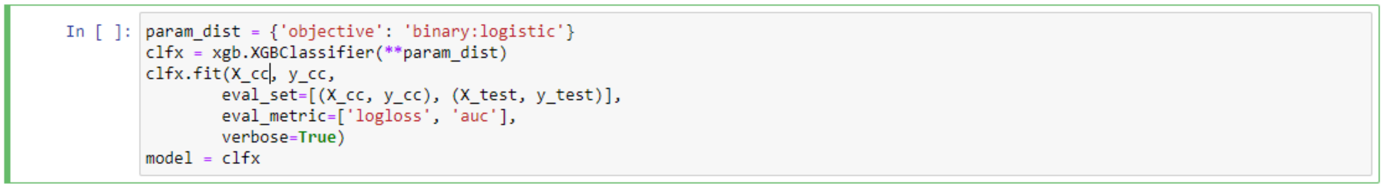
```
In [ ]: neigh = BayesSearchCV(
             KNeighborsClassifier(n_jobs=-1),
            {
                 'n_neighbors': Integer(1,10),
                 'weights': Categorical(['distance', 'uniform']),
                 'algorithm': Categorical(['auto', 'ball_tree', 'kd_tree', 'brute']),
             },
             n_iter=5,
             random_state=0
        )
        neigh.fit(X_cc, y_cc)
        model = neigh.best_estimator_
```

19) *Extreme gradient boost*



```
In [ ]: param_dist = {'objective': 'binary:logistic'}
        clfx = xgb.XGBClassifier(**param_dist)
        clfx.fit(X_train, y_train,
                eval_set=[(X_train, y_train), (X_test, y_test)],
                eval_metric=['logloss', 'auc'],
                verbose=True)
        model = clfx
```

20) *Extreme gradient boost sampled data*



```
In [ ]: param_dist = {'objective': 'binary:logistic'}
        clfx = xgb.XGBClassifier(**param_dist)
        clfx.fit(X_cc, y_cc,
                eval_set=[(X_cc, y_cc), (X_test, y_test)],
                eval_metric=['logloss', 'auc'],
                verbose=True)
        model = clfx
```