1. Introduction

1.1 The problem of Fraud Detection

Fraud is as old as humanity itself and can take an unlimited variety of different forms. Moreover, the development of new technologies provides additional ways in which criminals may commit fraud, for instance in e-commerce the information about the card is sufficient to perpetrate a fraud. The use of credit cards is prevalent in modern day society and credit card fraud has kept on growing in recent years. Financial losses due to fraud affect not only merchants and banks (e.g. reimbursements), but also individual clients. If the bank loses money, customers eventually pay as well through higher interest rates, higher membership fees, etc. Fraud may also affect the reputation and image of a merchant causing non-financial losses that, though difficult to quantify in the short term, may become visible in the long period. For example, if a cardholder is victim of fraud with a certain company, he may no longer trust their business and choose a competitor. The actions taken against fraud can be divided into fraud prevention, which attempts to block fraudulent transactions at source, and fraud detection, where successful fraud transactions are identified a posteriori. Technologies that have been used in order to prevent fraud are Address Verification Systems (AVS), Card Verification Method (CVM) and Personal Identification Number (PIN). AVS involves verification of the address with zip code of the customer while CVM and PIN involve checking of the numeric code that is keyed in by the customer. For prevention purposes, financial institutions challenge all transactions with rule based filters and data mining methods as neural networks. Fraud detection is, given a set of credit card transactions, the process of identifying if a new authorized transaction belongs to the class of fraudulent or genuine transactions. A Fraud Detection System (FDS) should not only detect fraud cases efficiently, but also be cost-effective in the sense that the cost invested in transaction screening should not be higher than the loss due to frauds. Bhatla shows that screening only 2% of transactions can result in reducing fraud losses accounting for 1% of the total value of transactions. However, a review of 30% of transactions could reduce the fraud losses drastically to 0.06%, but increase the costs exorbitantly. In order to minimize costs of detection it is important to use expert rules and statistical based models (e.g. Machine Learning) to make a first screen between genuine and potential fraud and ask the investigators to review only the cases with high risk. Typically, transactions are first filtered by checking some essential conditions (e.g. sufficient balance) and then scored by a

predictive model. The predictive model scores each transaction with high or low risk of fraud and those with high risk generate alerts. Investigators check these alerts and provide a feedback for each alert, i.e. true positive (fraud) or false positive (genuine). These feedbacks can then be used to improve the model.

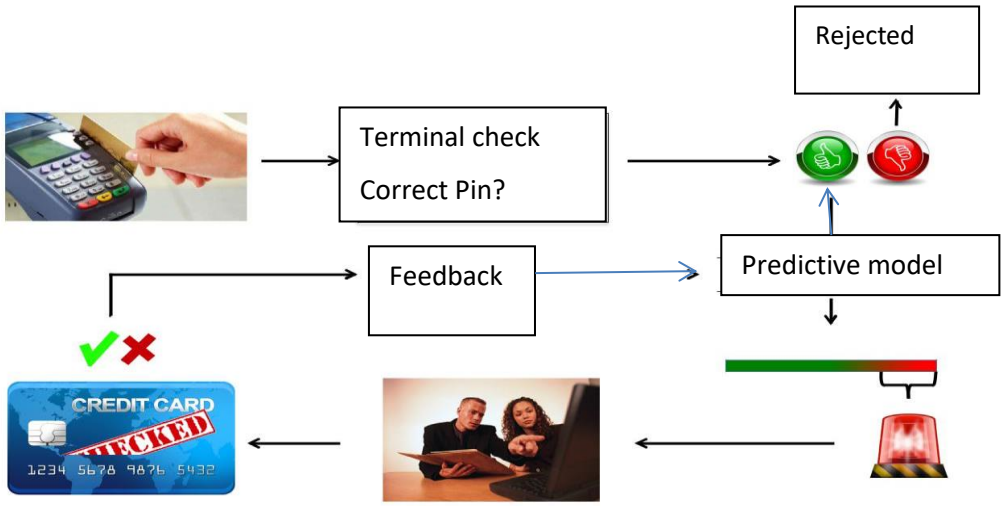


Fig 1. The Credit card Fraud Detection process

A predictive model can be built upon experts' rules, i.e. rules based on knowledge from fraud experts, but these require manual tuning and human supervision. Alternatively, with Machine Learning (ML) techniques we can efficiently discover fraudulent patterns and predict transactions that are most likely to be fraudulent. ML techniques consist in inferring a prediction model on the basis of a set of examples. The model is in most cases a parametric function, which allows predicting the likelihood of a transaction to be fraud, given a set of features describing the transaction. In the domain of fraud detection, the use of learning techniques is attractive for a number of reasons. First, they allow to discovery patterns in high dimensional data streams, i.e. transactions arrive as a continuous stream and each transaction is defined by many variables. Second, fraudulent transactions are often correlated both over time and space. For examples, fraudsters typically try to commit frauds in the same shop with different cards within a short time period. Third, learning techniques can be used to detect and model existing fraudulent strategies as well as identify new strategies associated to unusual behaviour of the cardholders. Predictive models based on ML techniques are also able to automatically integrate investigators' feedbacks to improve the accuracy of the detection, while in the case of expert system, including investigators feedbacks requires rules

revision that can be tedious and time consuming. When a fraud cannot be prevented, it is desirable to detect it as rapidly as possible. In cases, prevention and detection, the problem is magnified by a number of domain constraints and characteristics. First, care must be taken not to prevent too many legitimate transactions or incorrectly block genuine cards. Customer irritation is to be avoided. Second, most banks process vast numbers of transactions, of which only a small fraction is fraudulent, often less than 0.1% .Third, only a limited number of transactions can be checked by fraud investigators, i.e. we cannot ask a human person to check all transactions one by one if it is fraudulent or not. In other words companies and public institutions need automatic systems able to support fraud detection. Credit card frauds may occur in various ways: just to mention some, we can have stolen card fraud, cardholder-not-present fraud and application fraud:

- Stolen card fraud is the most common type of fraud where the fraudster usually tries to spend as much as possible and as quickly as possible. The detection of such a fraud typically relies on the discovery of an unexpected usage pattern of the credit card.
- Stolen card fraud is the most common type of fraud where the fraudster usually tries to spend as much as possible and as quickly as possible. The detection of such a fraud typically relies on the discovery of an unexpected usage pattern of the credit card.
- Application fraud corresponds to the application for a credit card with false personal information. This kind of fraud occurs more rarely since it could be detected during the application by checking the information of the applier, contrary to other frauds that cannot be anticipated.

The previous classification however is not exhaustive, since frauds are continuously evolving whenever a fraud method is detected; criminals adapt their strategies and try others. At the same time, everyday there are new criminals taking part in the game and trying new and old strategies. In this setting it is important to update the detection tools, but keeping old ones as well. Exchange of ideas for detection tools is difficult as fraudsters could benefit from it by testing their strategies. For the same reason datasets are typically not publicly available to the research community.

1.2 Challenges in Data Driven Fraud Detection Systems

The design of FDSs employing DDMs based on Machine Learning algorithms is particularly challenging for the following reasons:

1. Frauds represent a small fraction of all the daily transactions.
2. Frauds distribution evolves over time because of seasonality and new attack strategies.
3. The true nature (class) of the majority of transactions is typically known only several days after the transaction took place, since only few transactions are timely checked by investigators.

The first challenge is also known as the unbalanced problem, since the distribution of the transactions is skewed towards the genuine class. The distributions of genuine and fraud samples are not only unbalanced, but also overlapping. Most Machine Learning algorithms are not designed to cope with both unbalanced and overlapped class distributions. The change in fraudulent activities and costumer behaviour is the main responsible of non-stationary in the stream of transactions. This situation is typically referred to as concept drift and is of extreme relevance for FDSs which have to be constantly updated either by exploiting the most recent supervised samples or by forgetting out-dated information that might be no more useful whereas not misleading. FDS strategies that are not updated or revisited frequently are often losing their predictive accuracy in the long term. The third challenge is related to the fact that, in a real-world setting, it is impossible to check all transactions. The cost of human labour seriously constrains the number of alerts, returned by the FDS, which can be validated by investigators. Investigators check FDS alerts by calling the cardholders, and then provide the FDS with feedbacks indicating whether the alerts were related to fraudulent or genuine transactions. These feedbacks, which refer to a tiny fraction of the daily transactions amount, are the only real-time information that can be provided to train or update classifiers. The class (fraudulent / non-fraudulent) of the rest of transactions is known only several days later. Classes can be automatically assigned when a certain time period has passed, e.g. by assuming a certain reaction time for customers to discover and then report frauds. Standard FDSs ignoring feedbacks from investigators often provide less accurate alerts than FDSs able to use efficiently both feedbacks and the other supervised samples available.

2. Literature Review

[1], describes the “Decision Tree Induction Algorithm” which is used for Credit Card Fraud Detection. In this paper it discusses about the method, decision tree approach is a new cost sensitive technique compared with well-known traditional classification models on a real world credit card fraud data set, which reduces the sum of misclassification cost while selecting the splitting attribute at each non-terminal node is advanced. Credit card fraud detection is to reduce the bank risks, also used to equalize the transaction information with credit card fraud transaction of historical profile pattern to predict the probability of being fraudulent for a new transaction.

Research paper [2], describes the “Credit Card Fraud Detection Using Decision Tree for tracing Email and IP Address”. By using this technique, we can able to find out the fraudulent customer/merchant through tracing the fake mail and IP address. If the mail is fake, the customer/merchant is suspicious and information about the owner/sender is traced through IP address.

Paper [3], describes the “Neural Network” is used for the comparison on one way solution by using neural network, for matching the previous stored patterns and currently used patterns from which we can detect such patterns for Credit Card Fraud Detection.

[4], describes the “Clustering Based Approach” with K-means cluster analysis is a technique for smashing dataset down into narrated constituents in such a way that patterns and order becomes perceivable. Searching outliers is a main duty in k-means clustering for Credit Card Fraud Detection.

Yet another paper [4], describes the “Data Mining Techniques” for Fraud Detection in Credit Card. In this paper, we can detect the hidden information like whether an incoming transaction is fraudulent or not. It also dividing the transaction amount in three categories used on different ranges of transaction amount each group show the aberration symbols. The different steps in credit card transaction processing are represented as the underlying stochastic process of a Hidden Markov Model. HMM is used to model the sequence of operation in credit card transactions with the behaviour of cardholder.

In same sort of paper [5], describes the “Credit Card Fraud Detection System: A Survey”. This survey paper they tell the steps to detect the fraud transaction using Hidden Markov Model.

2.1 Classifier Techniques

In credit card fraud detection there are many methods that have been used so far.

- Logistic Regression
- Decision Tree
- K-Nearest Neighbours
- Random Forest Classifier

2.1.1 Decision Tree:

Decision Tree Algorithm is a Data mining induction technique that recursively shares a set of records. This algorithm used for solving regression and classification problems using tree representation. Decision tree contains one root note, internal nodes and leaf nodes. The nodes are labelled with the use of attribute names, edges are labelled with the values of attributes. For predicting a class label for a record we start from the root of the tree. Compare the values of the root attribute with record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node, comparing our record's attribute values continuously with other internal nodes of the tree until we reach a leaf node with predicted class value. Decision Tree is easy to implement, understand and display comparing to other classification algorithm. Using decision tree we can also tracing the mail and IP address through the credit card fraud detection. This fraud detection depends on the location where the cardholder use the previous credit card transaction compares with the location of current places transaction.

Example of decision tree-

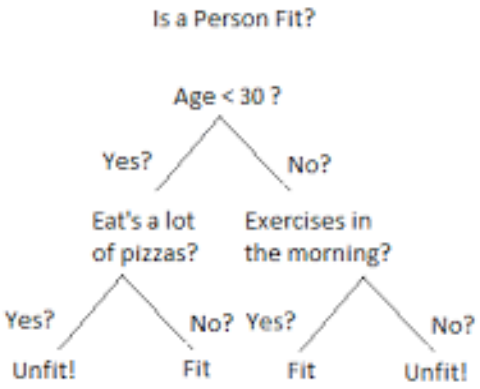


Fig 2. Decision Tree Classifier

Advantages of Decision trees:

- Are simple to understand and interpret. People are able to understand decision tree models after a brief explanation.
- Have value even with little hard data. Important insights can be generated based on experts describing a situation (its alternatives, probabilities, and costs) and their preferences for outcomes.
- Help determine worst, best and expected values for different scenarios.
- Use a white box model. If a given result is provided by a model.
- Can be combined with other decision techniques.

Disadvantages of decision trees:

- They are unstable, meaning that a small change in the data can lead to a large change in the structure of the optimal decision tree.
- They are often relatively inaccurate. Many other predictors perform better with similar data. This can be remedied by replacing a single decision tree with a random forest of decision trees, but a random forest is not as easy to interpret as a single decision tree.
- For data including categorical variables with different number of levels, information gain in decision trees is biased in favour of those attributes with more levels.
- Calculations can get very complex, particularly if many values are uncertain and/or if many outcomes are linked.

2.1.2 K-Nearest Neighbours:

K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection.

It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data)

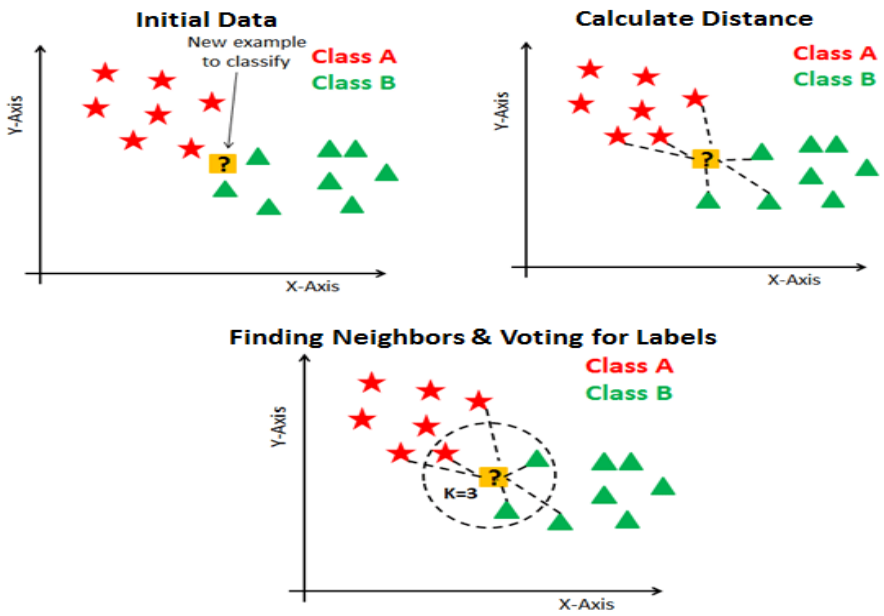


Fig 3. Example of K nearest neighbours

Advantages of KNN-

- Robust to noisy training data (especially if we use inverse square of weighted distance as the "distance")
- Effective if the training data is large

Disadvantages of KNN-

- Need to determine value of parameter K (number of nearest neighbors)
- Distance based learning is not clear which type of distance to use and which attribute to use to produce the best results. Shall we use all attributes or certain attributes only?
- Computation cost is quite high because we need to compute distance of each query instance to all training samples. Some indexing (e.g. K-D tree) may reduce this computational cost.

2.1.3 Logistic Regression:

Logistic Regression measures the relationship between the dependent variable (our label, what we want to predict) and the one or more independent variables (our features), by estimating probabilities using its underlying logistic function.

These probabilities must then be transformed into binary values in order to actually make a prediction. This is the task of the logistic function, also called the sigmoid function. The Sigmoid-Function is an S-shaped curve that can take any real-valued number and map it into a value between the range of 0 and 1, but never exactly at those limits. These values between 0 and 1 will then be transformed into either 0 or 1 using a threshold classifier.

Advantages and Disadvantages of Logistic Regression:

It is a widely used technique because it is very efficient, does not require too many computational resources, it's highly interpretable, it doesn't require input features to be scaled, it doesn't require any tuning, it's easy to regularize, and it outputs well-calibrated predicted probabilities.

Like linear regression, logistic regression does work better when you remove attributes that are unrelated to the output variable as well as attributes that are very similar (correlated) to each other. Therefore Feature Engineering plays an important role in regards to the performance of Logistic and also Linear Regression. Another advantage of Logistic Regression is that it is incredibly easy to implement and very efficient to train. I typically start with a Logistic Regression model as a benchmark and try using more complex algorithms from there on.

Because of its simplicity and the fact that it can be implemented relatively easy and quick, Logistic Regression is also a good baseline that you can use to measure the performance of other more complex Algorithms.

A disadvantage of it is that we can't solve non-linear problems with logistic regression since its decision surface is linear.

2.1.4 Random Forest Classifier:

Random Forest is a supervised learning algorithm. Like you can already see from its name, it creates a forest and makes it somehow random. The forest it builds is an ensemble of Decision Trees, most of the time trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result.

Random Forest has nearly the same hyper parameters as a decision tree or a bagging classifier. Fortunately, you don't have to combine a decision tree with a bagging classifier and can just easily use the classifier-class of Random Forest. Like I already said, with Random Forest, you can also deal with Regression tasks by using the Random Forest regressor.
Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

Therefore, in Random Forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. You can even make trees more random, by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

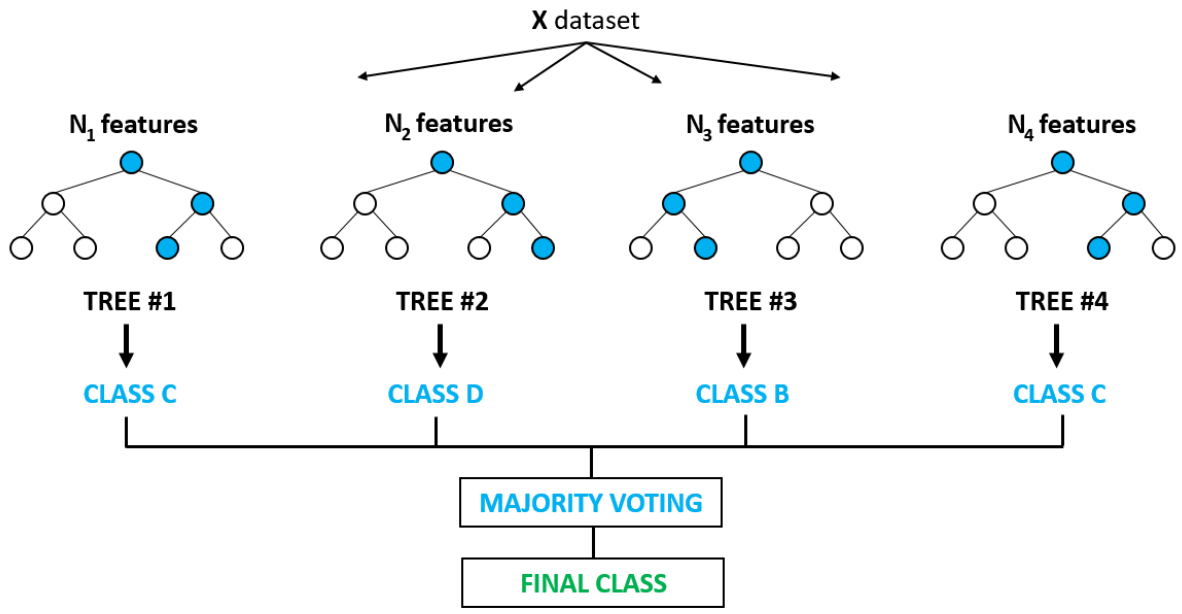


Fig 4. Random Forest Classifier

Advantages of Random Forest:

- As we mentioned earlier a single decision tree tends to overfit the data. The process of averaging or combining the results of different decision trees helps to overcome the problem of overfitting.

- Random forests also have less variance than a single decision tree. It means that it works correctly for a large range of data items than single decision trees.
- Random forests are extremely flexible and have very high accuracy.
- They also do not require preparation of the input data. You do not have to scale the data.
- It also maintains accuracy even when a large proportion of the data are missing.

Disadvantages of Random Forest:

- The main disadvantage of Random forests is their complexity. They are much harder and time-consuming to construct than decision trees.
- They also require more computational resources and are also less intuitive. When you have a large collection of decision trees it is hard to have an intuitive grasp of the relationship existing in the input data.
- In addition, the prediction process using random forests is time-consuming than other algorithms.

2.2 Evaluation of a classification problem

In a classification problem an algorithm is assessed on its overall accuracy to predict the corr`ect classes of new unseen observations and usually error is assessed in terms of Mean Misclassification Error (MME). Let Y1 be the set of positive instances, Y0 the set of negative instances, ~Y1 the set of instances predicted as positive and ~Y0 the ones predicted as negative. For a binary classification problem it is conventional to define a confusion matrix.

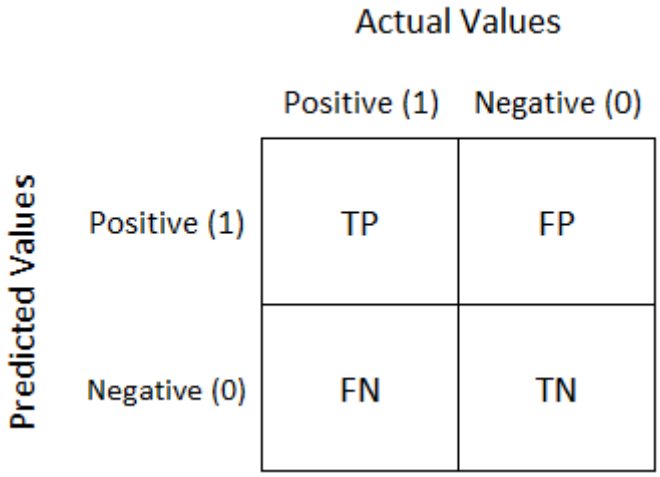


Fig 5. Confusion Matrix

In the case of zero-one loss we can then define MME = |FP∧FN|/ N, where operator |·| defines the cardinality of a set and N the size of the dataset. In the following we will write FP to denote |FP| and similarly for all the other entries of the confusion matrix. From MME we can then define the accuracy of a model as 1 - MME. However, in some situations; the accuracy is not a good measure of performance, especially in unbalanced classification problem where one class is much more frequent than the other. For example consider the case where we have N = 10000 transactions and only 1% are fraudulent (100 positives). Predicting all transactions as genuine (negative) would return to an accuracy of 0.99, but we would miss to detect all fraudulent cases.

- Precision : TP/ (TP +FP)
- Recall: TP / (TP +FN), also called True Positive Rate (TPR), Sensitivity or Hit rate.
- True Negative Rate (TNR): TN / (FP +TN) , also called Specificity
- False Positive Rate (FPR): FP/ (FP +TN)
- False Negative Rate (FNR): FN / (TP +FN)
- Balanced Error Rate (BER): 0.5(FP / (TN+FP) + FN/ (FN+TP))
- G-mean: (Sensitivity × Specificity) ^ 0.5
- F-measure: 2(Precision×Recall / (Precision+Recall)), also called F-score or F1

In an unbalanced classification problem, it is also well known that quantities like TPR and TNR are misleading assessment measures. Imagine a classifier with TPR=99% and TNR=99%, if we have 1% of positive samples, then Precision is only 0.5. Even worse, if we have only 0.1% of positives, then Precision is 0.09.

Balanced Error Rate may be inappropriate too because of different costs of misclassification false negatives and false positives. Precision and Recall have opposite behaviour, having high Precision means bad Recall and vice versa. F-measure gives equal importance to Precision and Recall into a metric ranging between 0 and 1 (the higher the better). F-measure and G-mean are often considered to be relevant measures in unbalanced problem .In general these measures can be computed only once a confusion matrix is available, which means that their values depend on the threshold used for classification defined by. Changing the threshold corresponds to use different misclassification costs. The Receiving Operating Characteristic (ROC) curve is a well-known assessment technique that allows evaluating the performance of a classifier over a range of different thresholds. It is obtained by plotting TPR against FPR

where each point of the curve corresponds to a different classification threshold. A classifier K is said to be more accurate than a classifier W in the ROC space only if the curve of K always dominates the curve of W. The best classifier corresponds to the point (0,1) in the ROC space (no false negatives and no false positives), while a classifier predicting at random would have performances along the diagonal connecting the bottom left corner to the top right. When there is not a clear winner (e.g. classifier K dominates W only in one part of the ROC space), the comparison is usually done by calculating the Area under the ROC Curve (AUC).

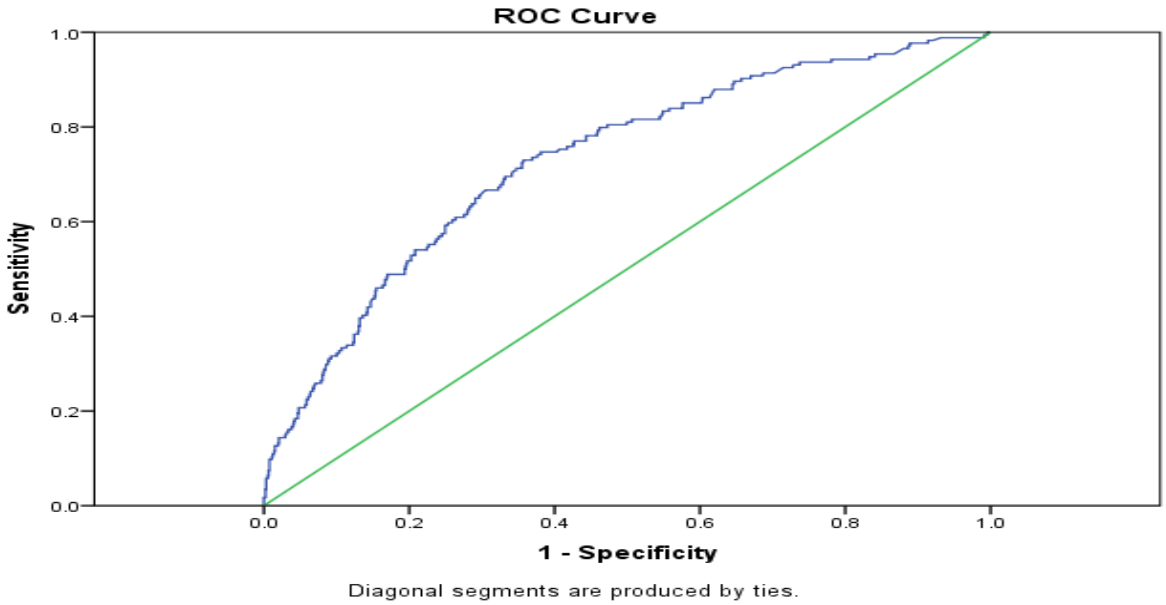


Fig 6. ROC Curve

AUC is also a well-accepted measure for unbalanced datasets and it has become the de facto standard in classification. However, AUC is insensitive to class priors since both TPR and FPR do not change with a different class ratio. Precision-Recall (PR) curves are instead sensitive to changes in the class distribution, but there is there a strong connection between ROC and PR curves. A curve dominates in ROC space if and only if it dominates in PR space, and algorithms that optimize the area under PR curve are guaranteed to optimize the area under ROC curve. When evaluating the output of a classifier it is also important to assess the quality of the estimated probabilities. A well-known measure of quality is Brier Score (BS). BS is a measure of average squared loss between the estimated probabilities and

the actual class value. It allows evaluating how well probabilities are calibrated, the lower the BS the more accurate are the probabilistic predictions of a model.

2.3 Sampling Techniques

The main challenge when it comes to modelling fraud detection as a classification problem comes from the fact that in real world data, the majority of transactions are not fraudulent. Investment in technology for fraud detection has increased over the years so this shouldn't be a surprise, but this brings us a problem: imbalanced data.

Our dataset 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 imbalanced, with the positive class (frauds) accounting for 0.072% of all transactions.

Learning from unbalanced datasets is a difficult task since most learning algorithms are not designed to cope with a large difference between the numbers of cases belonging to different classes. There are several methods that deal with this problem and we can distinguish between methods that operate at the data and algorithmic levels. At the data level, the unbalanced strategies are used as a pre-processing step to rebalance the dataset or to remove the noise between the two classes, before any algorithm is applied. At the algorithmic level, algorithms are themselves adjusted to deal with the minority class detection. Data level methods can be grouped into five main categories: sampling, ensemble, cost-based, distance-based and hybrid. Within algorithmic methods instead we can distinguish between:

i) classifiers that are specifically designed to deal with unbalanced distribution and
ii) classifiers that minimize overall classification cost.

The latter are known in the literature as cost-sensitive classifiers. Both data and algorithm level methods that are cost-sensitive target the unbalanced problem by using different misclassification costs for the minority and majority class. At the data level, cost-based methods sample the data to reproduce the different costs associated to each cost.

Typically, sampling methods are used to rebalance the datasets, because studies have shown that standard classifiers have better performances when trained on a balanced training set Sampling techniques do not take into consideration any class information in removing or

adding observations, yet they are easy to implement and to understand. Undersampling consists in downsizing the majority class by removing observations at random. In an unbalanced problem it is realistic to assume that many observations of the majority class are redundant and that by removing some of them at random the resulting distribution should not change much. However, the risk of removing relevant observations from the dataset is still present, since the removal is done in an unsupervised manner. In practice, this technique is often adopted since it is simple and speeds up the learning phase.

Oversampling consists in up-sizing the small class at random decreasing the level of class imbalance. By replicating the minority class until the two classes have equal frequency, oversampling increases the risk of over fitting by biasing the model towards the minority class. Other drawbacks of this approach are that it does not add any new informative minority examples and that it increases the training time. This can be particularly ineffective when the original dataset is fairly large.

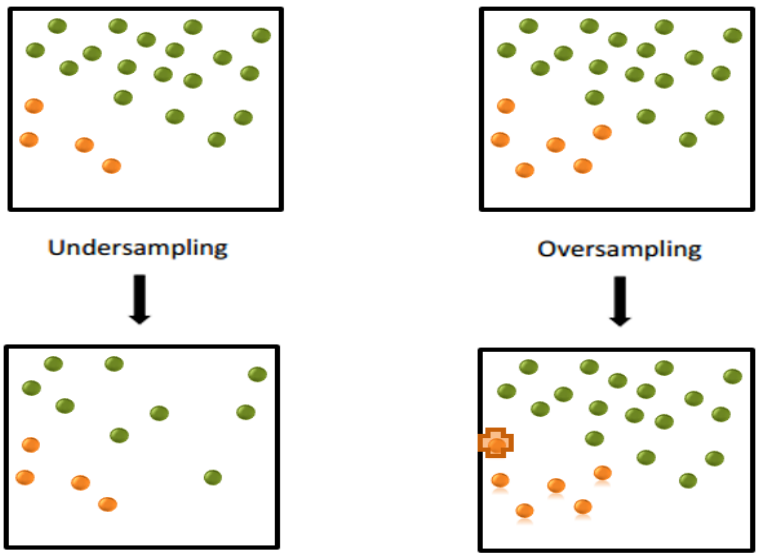


Fig 7. Under sampling and Oversampling

3. Methodology

In this section we describe the working conditions of a real-world Fraud Detection System (FDS). The hierarchy of layers in a FDS, each controlling whether the transactions is genuine or should be rather reported as a fraud: i) Terminal, ii) Transaction Blocking Rules, iii) Scoring Rules, iv) Data Driven Model, v) Investigators.

The first four elements of the FDS are fully automatized, while the last one requires human intervention and it is the only non-automatic and offline part of the FDS. Automatic tools have to decide whether the transaction request (or the transaction attempt) has to be approved in Real Time (i.e., decision has to be taken immediately) or in Near Real Time (i.e. decisions can be taken in a short time). Blocking and scoring rules are Expert Driven Rules (EDR), i.e. rules designed by investigators based upon their experience. On the contrary, the DDM uses annotated transactions as source of information to extract knowledge about fraudulent and genuine patterns. In the following we will explain the role played by each component of the FDS.

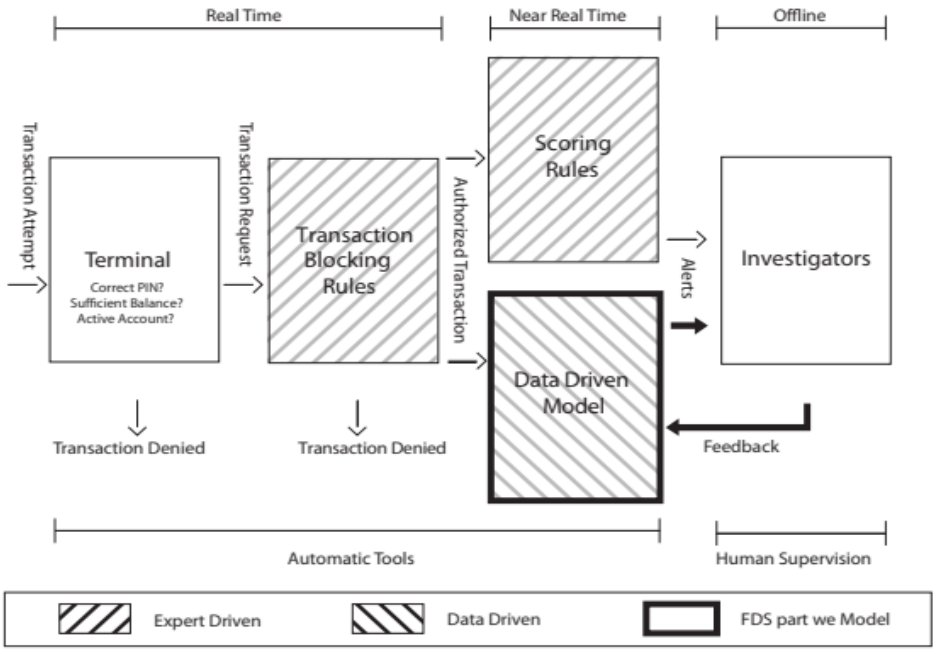


Fig 8. The layers of Fraud Detection System

3.1 FDS Layers:

The Terminal represents the first layer of conventional security checks in a FDS. At this stage security checks such as correct PIN code, number of attempts, status of the card (active or blocked), sufficient balance and expenditure limit provide a first filter of the transactions that can get through. These conditions are evaluated in real time and if any of them is not satisfied, the transaction is denied. All transactions passing this first filter raise a transaction request to the second layer controlled by the blocking rules.

Transaction Blocking Rules are designed by experienced investigators and they can block a transaction request before it is authorized. These rules operate in Real Time, and are very precise, they deny a transaction request only if it clearly represents a fraud attempt. Blocking Rules are if-then (-else) rules that associate the class fraud or genuine to a transaction when specific conditions are met. An example can be the following: "IF a PIN-based transaction is done in Europe AND in Asia within 5 min from the same cardholder THEN denies the transaction", since the customer cannot physically be in Europe and Asia over a short time interval. Transaction requests passing Blocking Rules are authorized and become transactions (the payment is executed). Before each authorized transaction is sent to the successive security layer of FDS it is enriched with aggregated information (e.g. average expenditure, number of transactions in the same day of the same cardholder, etc.). The resulting feature vector describing the transaction is then analysed by both Scoring Rules and DDMs. These controls are performed in Near Real Time (typically less than 6s) since there is no need to provide an immediate response given that the transaction has been already authorized.

Scoring Rules are EDR defined by investigators based upon their experience that act like a rule-based classifier. They assign a score to each authorized transaction: the larger the score, the more likely the transaction is to be a fraud. In practice scoring rules contain simple, human-understandable conditions (e.g. IF Internet transaction in fiscal paradise and amount > 1000$, THEN fraud score = 0.9) and as such can be easily designed by investigators or other experts. These rules are easy to understand and fast to deploy, but require manual revision when their performance drops. As a consequence, these rules can be expensive to maintain.

Data Driven Model (DDM) layer relies on a predictive model to assign a fraud score to each transaction. Usually this phase uses ML algorithms that return for each transaction an estimate of the probability to be a fraud. These algorithms can learn complex correlations in

the data using large volume of data with high dimensionality. They are usually more robust than EDR, but most of them are black box, i.e. it is not possible to convert them into rules that are easy to interpret. DDMs are able to consider all the information associated to a transaction, while EDR returns conditions based on few features of the transactions. Investigators are fraud experts that check fraud alerts, i.e. transactions that have received a high fraud score by either EDR or DDM. These suspicious transactions are displayed in a Case Management Tool (CMT) where investigators can see the associated fraud score, check where they come from (EDR or DDM) and annotate them as genuine or fraud after verification. In a realistic scenario only few alerts can be checked given a limited number of investigators. For this reason, they typically investigate only transactions with the highest fraud score. The role of investigators is to contact the cardholder, check if the transaction is fraudulent or not and annotate it with its correct label. This process can be long, tedious, and at the same time the number of transactions to check is usually very large, so it is important to minimize false alerts. The labelling process generates annotated transactions that are used by the DDM. In the following we use the term feedbacks to refer to these labelled transactions.

Expert driven systems are updated manually, while the data-driven component is updated automatically. Alerts generated today define the feedbacks that will be used to train an algorithm that detects the frauds of tomorrow. This means that Alert-Feedback Interaction (AFI) governs the type of information available to the algorithm that regulates the data driven phase.

Typically, DDMs require a large set of samples to train an accurate model, so the ML algorithms used in the data driven layers are trained at the end of the day, when a sufficient batch of feedbacks are available. Feedbacks of alerted transactions are given by investigators during the day, but only at the end of the day we have all the feedbacks for the alerts generated by the FDS. Therefore, the DDM is usually updated every day at midnight and the detection model is then applied to all the transactions occurring in the next days.

The focus of this project is to improve the DDM part of the FDS and to model the interaction between Data Driven Methods based on ML and investigators. In particular, we will consider a scenario where the DDM is the only element of the FDS responsible for the alerts given to fraud experts and the algorithms are able to learn from the feedback provided. In the

remainder of the thesis we will use the term FDS to indicate only the FDS layers we model: DDM and investigators.

3.2 Data Driven Model

In this phase of FDS model, we compared and contrasted the existing algorithms with 2 new algorithms.

We have used a total of 5 algorithms in our project-

1. Logistic Regression

2. K-Nearest Neighbours

3. Decision Tree

4. Random Forest

5. XgBoost

3.2.1 XGBoost:

XGBoost has become a widely used and really popular tool among Kaggle competitors and Data Scientists in industry, as it has been battle tested for production on large-scale problems. It is a highly flexible and versatile tool that can work through most regression, classification and ranking problems as well as user-built objective functions. As an open-source software, it is easily accessible and it may be used through different platforms and interfaces. The amazing portability and compatibility of the system permits its usage on all three Windows, Linux and OS X. It also supports training on distributed cloud platforms like AWS, Azure, GCE among others and it is easily connected to large-scale cloud dataflow systems such as Flink and Spark. Although it was built and initially used in the Command Line Interface (CLI) by its creator (Tianqi Chen), it can also be loaded and used in various languages and interfaces such as Python, C++, R, Julia, Scala and Java.

Its name stands for eXtreme Gradient Boosting; it was developed by Tianqi Chen and now is part of a wider collection of open-source libraries developed by the Distributed Machine Learning Community (DMLC). XGBoost is a scalable and accurate implementation of gradient boosting machines and it has proven to push the limits of computing power for boosted trees algorithms as it was built and developed for the sole purpose of model

performance and computational speed. Specifically, it was engineered to exploit every bit of memory and hardware resources for tree boosting algorithms.

3.2.2 Techniques used for Under-sampling and Oversampling

- Random Oversampling - Random Oversampling involves supplementing the training data with multiple copies of some of the minority classes. Oversampling can be done more than once (2x, 3x, 5x, 10x, etc.) This is one of the earliest proposed methods, that is also proven to be robust. Instead of duplicating every sample in the minority class, some of them may be randomly chosen with replacement.
- Random Under sampling- Randomly removes samples from the majority class, with or without replacement. This is one of the earliest techniques used to alleviate imbalance in the dataset, however, it may increase the variance of the classifier and may potentially discard useful or important samples
- Tomek Links Under sampling - Tomek links remove unwanted overlap between classes where majority class links are removed until all minimally distanced nearest neighbour pairs are of the same class. This way, if two instances form a Tomek link then either one of these instances is noise or both are near a border. Thus, one can use Tomek links to clean up overlap between classes. By removing overlapping examples, one can establish well-defined clusters in the training set and lead to improved classification performance.
- Cluster Centroid under sampling - In Majority Under-sampling, unimportant (or not-so-important) instances is removed among majority samples. In CCMUT, the demarcation of instances as important and unimportant is done by using the concept of Clustering on Feature-Space Geometry. Clustering is an Unsupervised Learning Approach. But CCMUT, only uses the concept of finding cluster centroid (clusters are created encircling data-points belonging to the majority class), as already instances are labelled. The cluster centroid is found by obtaining the average feature vectors for all the features, over the data points belonging to the majority class in feature space. After finding the cluster centroid of the majority class, the instance belonging to the cluster (majority class), which is farthest from the cluster centroid in feature space, is considered to be the most unimportant instance. On the contrary, the instance belonging to the majority class, which is nearest to the cluster centroid in feature space, is considered to be the most important instance. So, in CCMUT, instances belonging to the majority class are removed on the basis of their

importance and number of samples to be under-sampled depends upon the % of Under-sampling or CCMUT.

- SMOTE - There are a number of methods available to oversample a dataset used in a typical classification problem (using a classification algorithm to classify a set of images, given a labelled training set of images). The most common technique is known as SMOTE: Synthetic Minority Over-sampling Technique. To illustrate how this technique works consider some training data which has *s* samples and *f* features in the feature space of the data. Note that these features, for simplicity, are continuous. As an example, consider a dataset of birds for classification. The feature space for the minority class for which we want to oversample could be beak length, wingspan, and weight (all continuous). To then oversample, take a sample from the dataset, and consider its *k* nearest neighbours (in feature space). To create a synthetic data point, take the vector between one of those *k* neighbours, and the current data point. Multiply this vector by a random number *x* which lies between 0, and 1. Add this to the current data point to create the new, synthetic data point. Many modifications and extensions have been made to the SMOTE method ever since its proposal
- SMOTE + Tomek Links- In this we combine SMOTE and Tomek link approach to improve accuracy.

3.3 Other layers of FDS

Fraud Detection problems are typically addressed in two different ways. In the static learning setting, a detection model is periodically retrained from scratch (e.g. once a year or month). In the online learning setting, the detection model is updated as soon as new data arrives. Though this strategy is the most adequate to deal with issues of non-stationary, little attention has been devoted in the literature to the unbalanced problem in a changing environment. Another problematic issue in credit card fraud detection is the scarcity of available data due to confidentiality issues that give little chance to the community to share real datasets and assess existing techniques.

The first part aims at making an experimental comparison of several state-of-the-art algorithms and modelling techniques on one real dataset, focusing in particular on some open questions like: Which machine learning algorithm should be used? Is it enough to learn a model once a month or it is necessary to update the model every day? How many transactions

are sufficient to train? Should the data be analyzed in their original unbalanced form? If not, which is the best way to rebalance them? Which performance measure is the most adequate to assess results?

The second part builds upon this result and proposes an algorithm solution for unbalanced data streams. In unbalanced data streams, state-of-the-art techniques use instance propagation and decision trees to cope with the unbalanced problem. However, it is not always possible to either revisit or store old instances of a data stream. We test our framework on several streaming datasets with unbalanced classes and concept drift.

FDSs typically relay on classification algorithms to identify transactions at risk of fraud that generate alerts, but are not able to integrate the feedback that investigators provide on the alerts raised by the FDS. As a consequence, most FDSs available in the literature ignore Alert-Feedback Interaction (AFI), making the unrealistic assumption that all transactions are correctly labelled by a supervisor. With a limited number of investigators only a restricted quantity of alerts can be checked, which means a small set of labelled transactions returned as feedback. Non-alerted transactions are a large set of unsupervised samples that can be either fraud or genuine. Additional labelled observations are obtained by means of cardholders that report unauthorized transactions. The number of customers reporting frauds not detected by the FDS is usually small and hard to model since cardholders have different habits when it comes to check the transcript of credit card transactions given by the bank. Then, every day in addiction to investigators' feedback, we have historical supervised samples for which the labels can safely assumed to be correct after some time. In summary, we can distinguish between two types of supervised samples: i) feedbacks provided by investigators and ii) historical transactions whose labels are received with a large delay. We will call the latter delayed samples to stress the fact that their label is available only after a while. In this formulation we assume that the FDS is updated everyday at midnight and the detection model is then applied to all the transactions occurring the following day. Feedbacks of alerted transactions are given by investigators during the day and by the end of the day we have all the feedbacks for the alerts generated by the FDS. In these settings, the ML algorithm learns from the batch of feedbacks available at the end of the day and is not trained from each transaction incrementally, i.e. the algorithm is trained only when a sufficient batch of supervised samples is provided.

3.4About the 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.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide 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 used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

```
Fraud = dataset[dataset['Class']==1]
Normal = dataset[dataset['Class']==0]
count_classes = pd.value_counts(dataset['Class'], sort = True)
count_classes.plot(kind = 'bar', rot=0)
plt.title("Transaction Class Distribution")
LABELS = ["Normal", "Fraud"]
plt.xticks(range(2), LABELS)
plt.xlabel("Class")
plt.ylabel("Frequency");
```

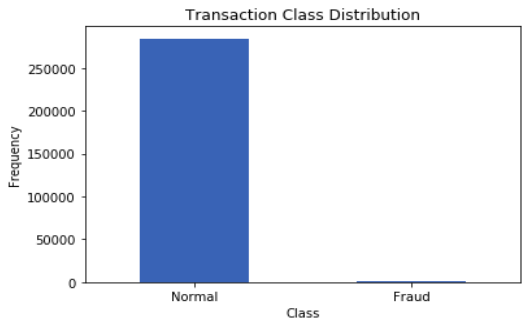


Fig.9. Dataset Overview

How different are the amount of money used in different transaction classes?

```
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Amount per transaction by class')
bins = 50
ax1.hist(Fraud.Amount, bins = bins)
ax1.set_title('Fraud')
ax2.hist(Normal.Amount, bins = bins)
ax2.set_title('Normal')
plt.xlabel('Amount ($)')
plt.ylabel('Number of Transactions')
plt.xlim((0, 20000))
plt.yscale('log')
plt.show();
```

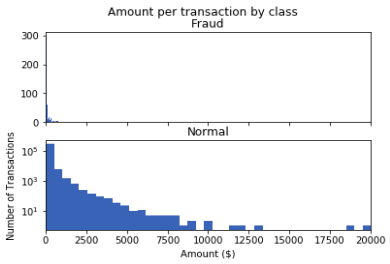


Fig.10. Amount variable

Do fraudulent transactions occur more often during certain time frame?

```
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Time of transaction vs Amount by class')
ax1.scatter(Fraud.Time, Fraud.Amount)
ax1.set_title('Fraud')
ax2.scatter(Normal.Time, Normal.Amount)
ax2.set_title('Normal')
plt.xlabel('Time (in Seconds)')
plt.ylabel('Amount')
plt.show()
```

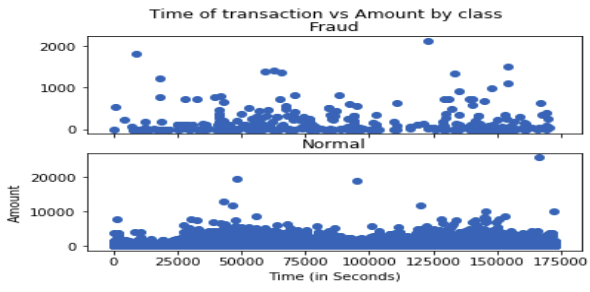
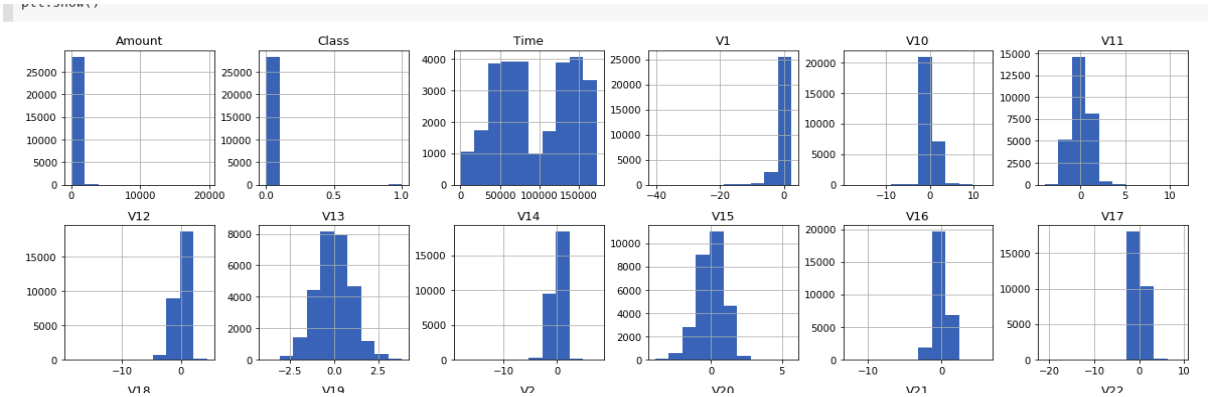


Fig.11. Time variable

Plot histogram of each parameter



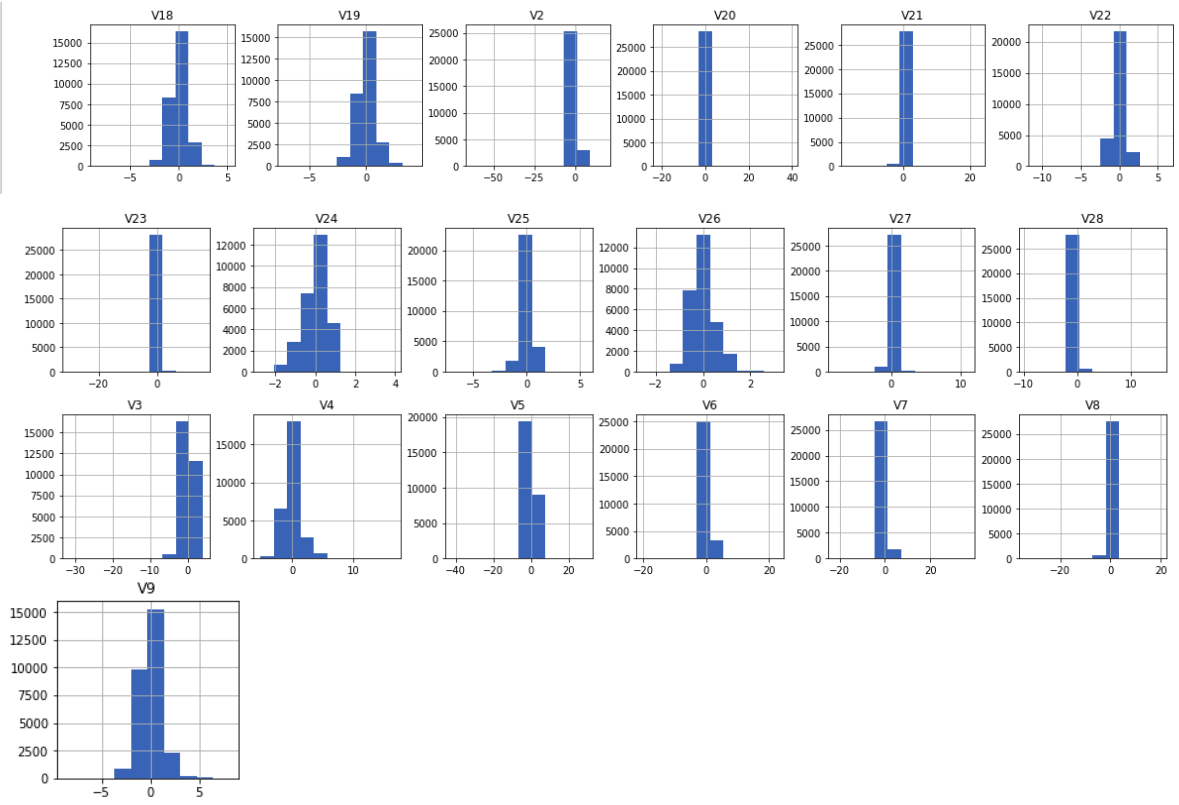
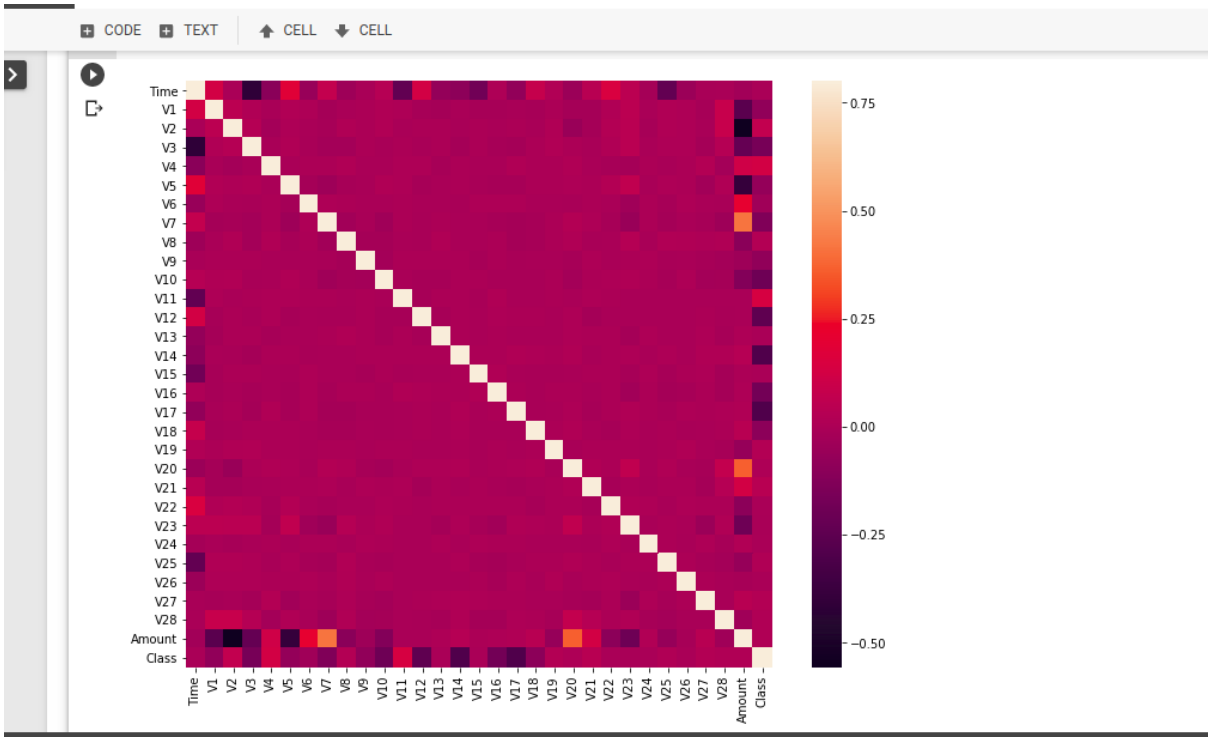


Fig.12. Histogram of each paramter

Fig. 13. Correlation Matrix based on Heatmaps



4.Experimental Results

The first result would be of under or over sampling process. The warping due to the instance selection procedure in under or over sampling is essentially equivalent to the bias that occurs with a change in the priors when class-within distributions remain stable. With undersampling, we create a different training set, where the classes are less unbalanced. However, if we make the assumption that the training and testing sets come from the same distribution, it follows that the probability estimates obtained after undersampling are biased. As a result of undersampling, the posterior probability $\sim p_s$ is shifted away from the true distribution, and the optimal separation boundary moves towards the majority class so that more cases are classified into the minority class. Finally, we would consider a highly unbalanced dataset (Credit-card), where the minority class accounts for only 0.172% of all observations. In this dataset, we expect that the large improvement in accuracy obtained with undersampling will be coupled with poor calibrated probabilities (large BS).

In these experiments we would test the techniques for unbalanced. In particular, we considered the following techniques: undersampling. We would start by performing a CV for each technique with different classification algorithms: Random Forest and Decision Tree etc . For each dataset we would compute the average G-mean in the CV and then calculate the average accuracy over all the datasets. We would also include the performances in the case of unbalanced datasets as benchmark. In this study, we expect to see that the combination of RF and undersampling appears to return the largest accuracy.

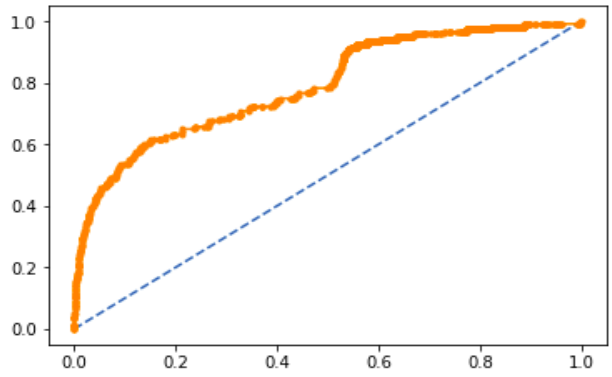
The strategy names follow a structure built on the following options:

- Algorithm used (RF,LR,Xgboost etc)
- Sampling method (Under or over or both)
- Model update frequency (One, Daily, 15days, weekly)
- Learning approach (Update, Propagate and Forget)

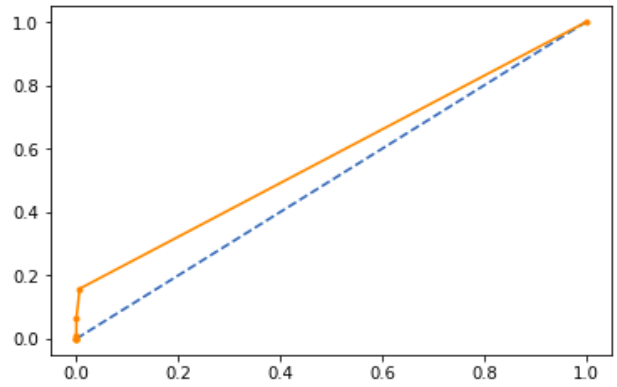
The various screenshots of different algorithms and comparison are as follows-

Fig 14. Normal algorithms

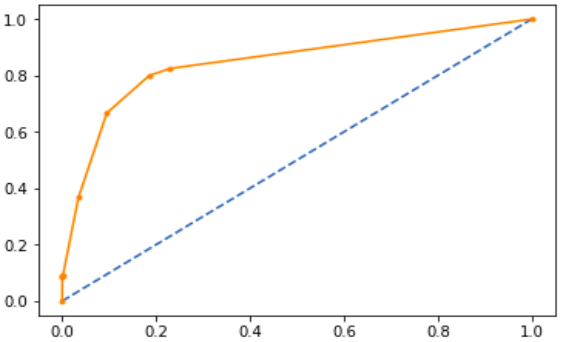
```
Confusion matrix for Normal logistic regression
[[71074     8]
 [  120     0]]
AUC for Normal Logistic Regression: 0.792
```




```
Confusion matrix for NormalKNN
[[71074     8]
 [  119     1]]
AUC for Normal KNN: 0.576
```

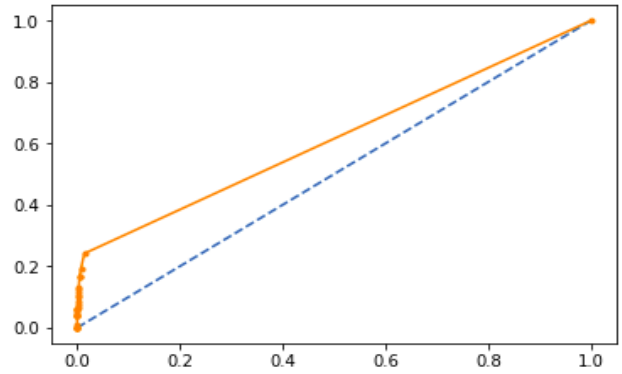



```
Confusion matrix for Normal Decision Tree
[[71082     0]
 [  120     0]]
AUC for Normal Decision Tree: 0.844
```

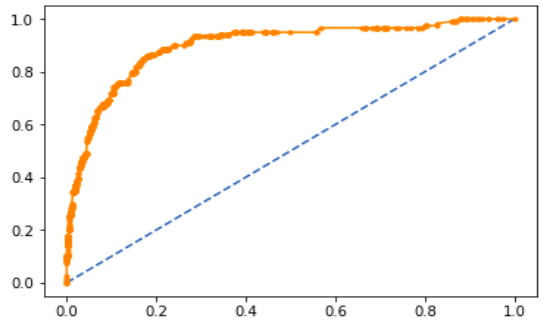



```
[ ]
```

```
Confusion matrix for Normal Random Forest
[[71047    35]
 [  115     5]]
AUC for Normal Random Forest: 0.615
```



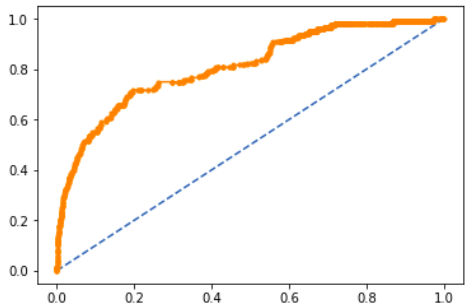

```
Confusion matrix for Normal XGBoost
[[71078     4]
 [  118     2]]
AUC for Normal XGboost: 0.899
```



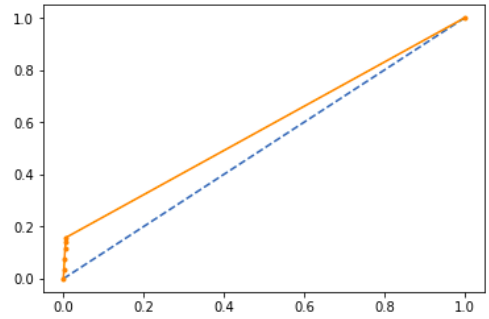

```
[ ]
```

Fig 15. Random Oversampled algorithm

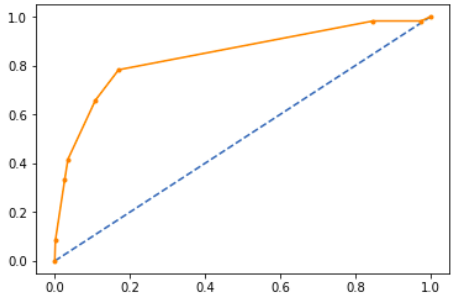
```
Confusion matrix for Random Oversampling + Logistic Regression
[[59456 11626]
 [   41    79]]
AUC for Random Oversampling + Logistic Regression: 0.812
```



```
Confusion matrix for random oversampling + KNN
[[59456 11626]
 [   41    79]]
AUC for Random oversampling + KNN: 0.576
```

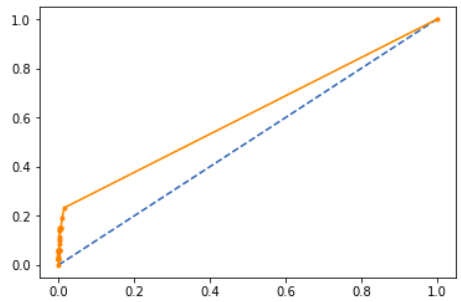


```
Confusion matrix for Random oversampling +Decision Tree
[[58993 12089]
 [   26    94]]
AUC for random oversampling+Decision Tree: 0.841
```



```
[ ]
```

```
Confusion matrix for Random Oversampling +Random Forest
[[71032    50]
 [  113     7]]
AUC for Random oversampling+Random Forest: 0.610
```




```
[ ]
```

```
Confusion matrix for Random oversampling + XGBoost
[[62134  8948]
 [   24    96]]
AUC for Random oversampling + XGBoost: 0.897
```

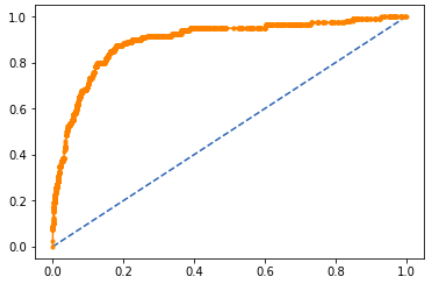
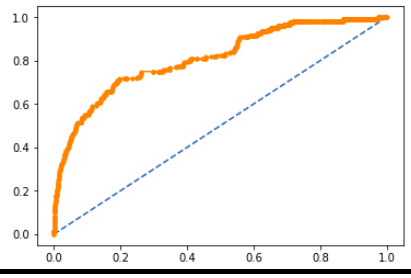
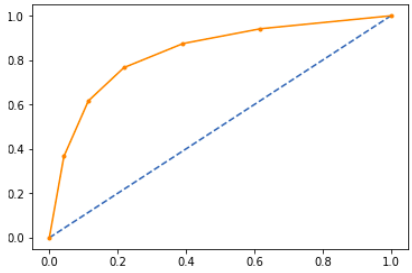


Fig 16. Random Undersampled Algorithms

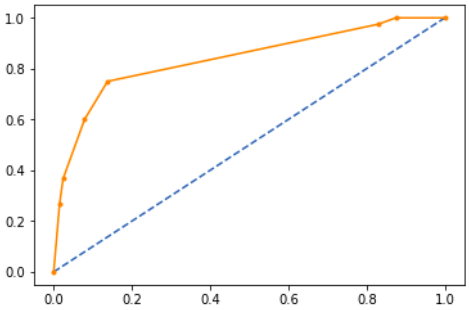
```
Confusion matrix for Random Undersampling + Logistic Regression
[[57339 13743]
 [   35    85]]
AUC for Undersampling +Logistic Regression: 0.812
```



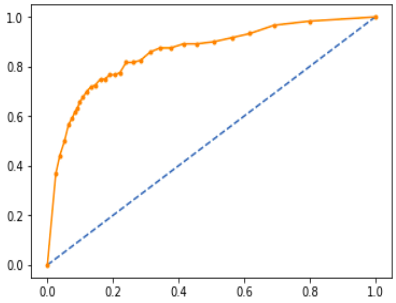
```
onfusion matrix for Random Undersampling+KNN
[55546 15536]
[   28    92]]
UC for random Undersampling+KNN: 0.834
```




```
Confusion matrix for Random Undersampling+Decision Tree
[[61288  9794]
 [   30    90]]
AUC for Random Undersampling+Decision Tree: 0.838
```




```
Confusion matrix for Random Undersampling+Random Forest
[[57573 13509]
 [   28    92]]
AUC for Random Undersampling+Random Forest: 0.856
```




```
Confusion matrix for random undersampling + XGBoost
[[58576 12506]
 [   26    94]]
AUC for Random Undersampling + XGboost: 0.886
```

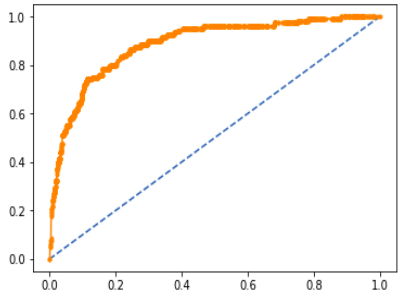
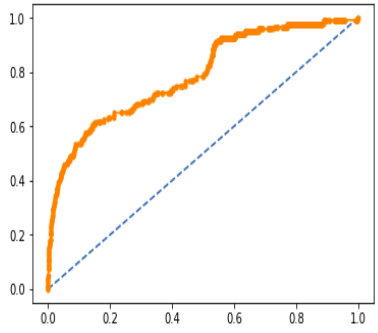
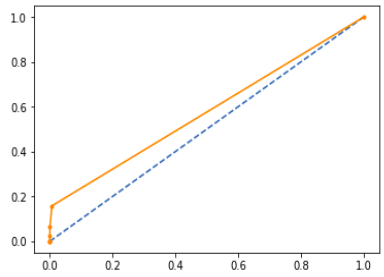


Fig 17. Tomek Links Undersampled Algorithms

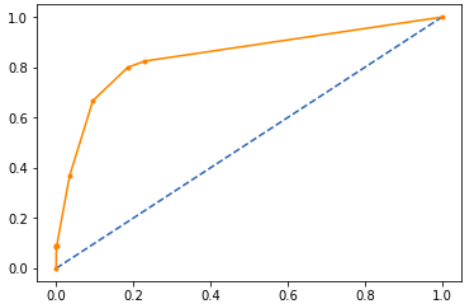
```
Confusion matrix for Undersampling(TL) + Logistic Regression
[[71074     8]
 [  120     0]]
AUC for Undersampling(TL) +Logistic Regression: 0.790
```



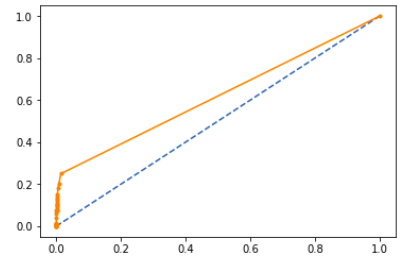

```
Confusion matrix for Undersampling(TL)+KNN
[[71073     9]
 [  117     3]]
AUC for Undersampling(TL)+KNN: 0.576
```




```
Confusion matrix for Undersampling(TL)+Decision Tree
[[71082     0]
 [  120     0]]
AUC for Undersampling(TL)+Decision Tree: 0.844
```



```
Confusion matrix for Undersampling(TL)+Random Forest
[[71033    49]
 [  111     9]]
AUC for Undersampling(TL)+Random Forest: 0.619
```




```
[ ]
```

```
Confusion matrix for Undersampling (TL)+ XGBoost
[[71076     6]
 [  111     9]]
AUC for Undersampling (TL)+ XGboost: 0.899
```

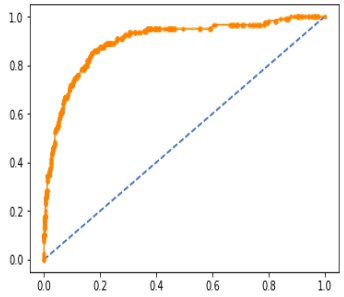
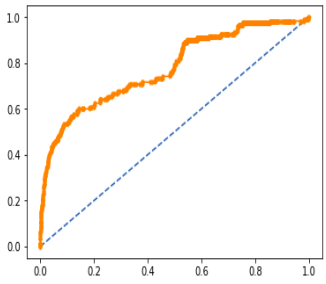
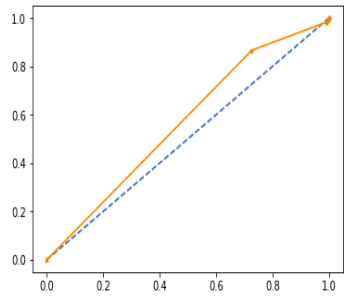


Fig 18. Cluster Centroids Undersampling Algorithms

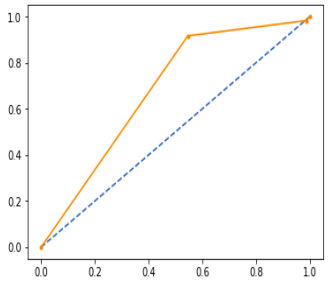
```
  FutureWarning)
Confusion matrix for Undersampling(CC) + Logistic Regression
[[  432 70650]
 [    1   119]]
AUC for Undersampling(CC) +Logistic Regression: 0.780
```



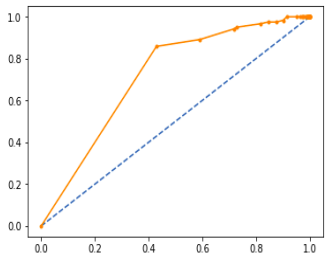
```
Confusion matrix for Undersampling (CC) +KNN
[[  230 70852]
 [    1   119]]
AUC for Undersampling(CC)+KNN: 0.568
```



```
Confusion matrix for Undersampling(CC) +Decision Tree
[[  834 70248]
 [    2   118]]
AUC for Undersampling(CC)+Decision Tree: 0.682
```



```
Confusion matrix for Undersampling(CC)+Random Forest
[[ 1135 69947]
 [    0   120]]
AUC for Undersampling (CC)+Random Forest: 0.718
```



```
Confusion matrix for Undersampling (CC)+XGBoost
[[ 1424 69658]
 [    1   119]]
AUC for Undersampling (CC)+XGboost: 0.777
```

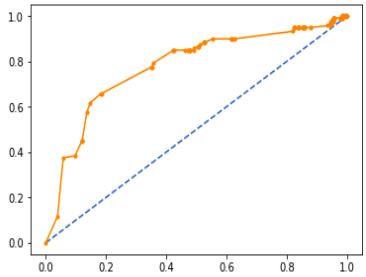
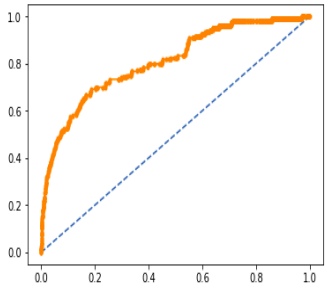
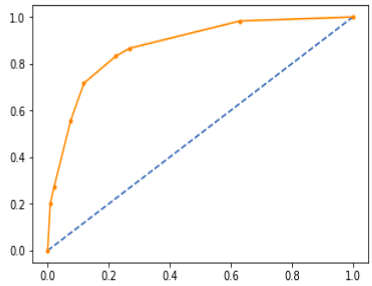


Fig 19. SMOTE

```
  FutureWarning)
Confusion matrix for OVersampling(SMOTE) + Logistic Regression
[[58796 12286]
 [   40    80]]
AUC for OVersampling(SMOTE) +Logistic Regression: 0.811
```




```
Confusion matrix for OVersampling(SMOTE)+Decision Tree
[[55157 15925]
 [   20   100]]
AUC for OVersampling(SMOTE)+Decision Tree: 0.874
```




```
Confusion matrix for OVersampling(SMOTE) + XGBoost
[[60942 10140]
 [   17   103]]
AUC for OVersampling(SMOTE) + XGboost: 0.932
```

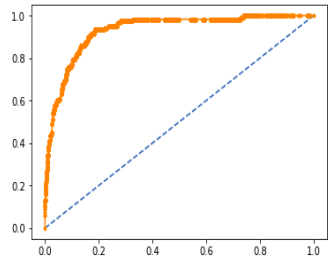
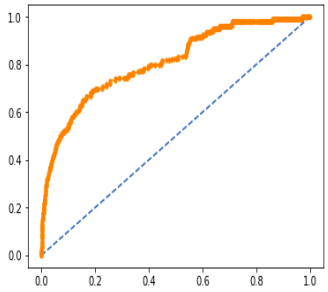
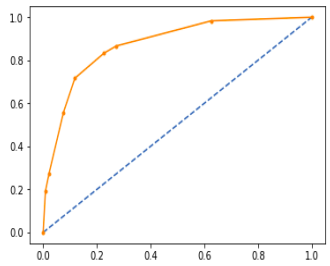


Fig 20. SMOTE+ TL

```
Confusion matrix for OVerUNDERsampling(SMOTE+TL) + Logistic Regression
[[58679 12403]
 [   40    80]]
AUC for OVerUNDERsampling(SMOTE+TL) + Logistic Regression: 0.811
```




```
Confusion matrix for OVerUNDERsampling(SMOTE+TL) + Decision Tree
[[54963 16119]
 [   20   100]]
AUC for OVerUNDERsampling(SMOTE+TL) + Decision Tree: 0.874
```




```
Confusion matrix for OVerUNDERsampling(SMOTE+TL) + XGBoost
[[60998 10084]
 [   18   102]]
AUC for OVerUNDERsampling(SMOTE+TL) + XGboost: 0.932
```

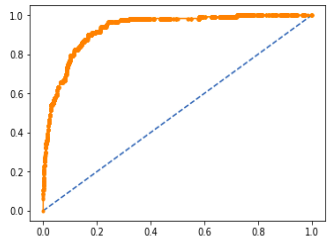


Fig 21. Comparison of Normal Algorithms

```
print ("Accuracy comparsion for Normal Algorithms")

print ("----------------------------------------")

print("Logistic Regression | %.3f" % auc1)
print("K-Nearest neighbour | %.3f" % auc2)
print("Decision Tree       | %.3f" % auc3)
print("Random Forest       | %.3f" % auc4)
print("XGBoost             | %.3f" % auc5)
```

```
Accuracy comparsion for Normal Algorithms
----------------------------------------
Logistic Regression | 0.792
K-Nearest neighbour | 0.576
Decision Tree       | 0.844
Random Forest       | 0.615
XGBoost             | 0.899
```

Fig 22. Comparison of random oversampled algorithms

```
print ("Accuracy comparsion for Random Oversample Algorithms")

print ("----------------------------------------")

print("Logistic Regression | %.3f" % auc6)
print("K-Nearest neighbour | %.3f" % auc7)
print("Decision Tree       | %.3f" % auc8)
print("Random Forest       | %.3f" % auc9)
print("XGBoost             | %.3f" % auc10)
```

```
Accuracy comparsion for Random Oversample Algorithms
----------------------------------------
Logistic Regression | 0.812
K-Nearest neighbour | 0.576
Decision Tree       | 0.841
Random Forest       | 0.610
XGBoost             | 0.897
```

Fig 23. Comparison of random undersampled algorithms

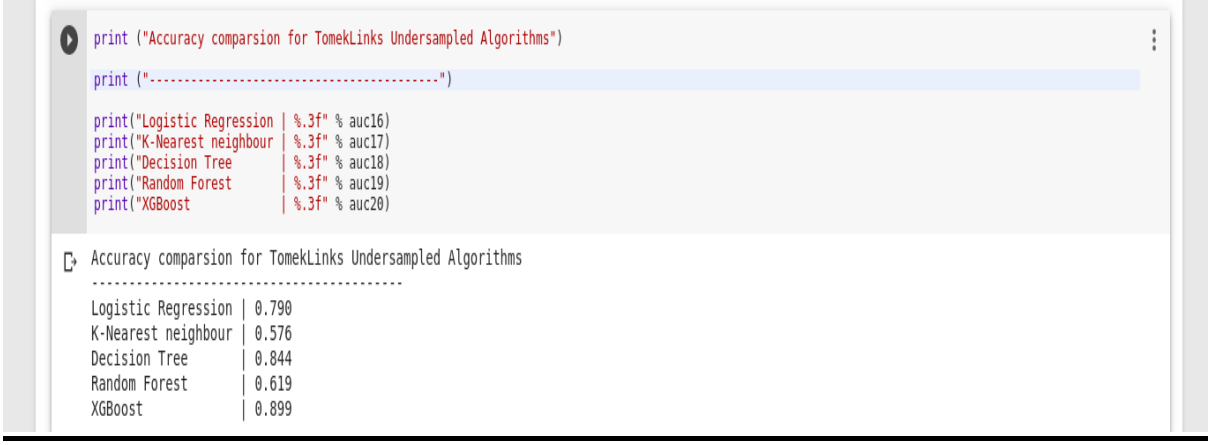
```
print ("Accuracy comparsion for Random Undersampled Algorithms")

print ("----------------------------------------")

print("Logistic Regression | %.3f" % auc11)
print("K-Nearest neighbour | %.3f" % auc12)
print("Decision Tree       | %.3f" % auc13)
print("Random Forest       | %.3f" % auc14)
print("XGBoost             | %.3f" % auc15)
```

```
Accuracy comparsion for Random Undersampled Algorithms
----------------------------------------
Logistic Regression | 0.812
K-Nearest neighbour | 0.834
Decision Tree       | 0.838
Random Forest       | 0.856
XGBoost             | 0.886
```

Fig 24. Comparison of Tomek Links undersampled algorithms



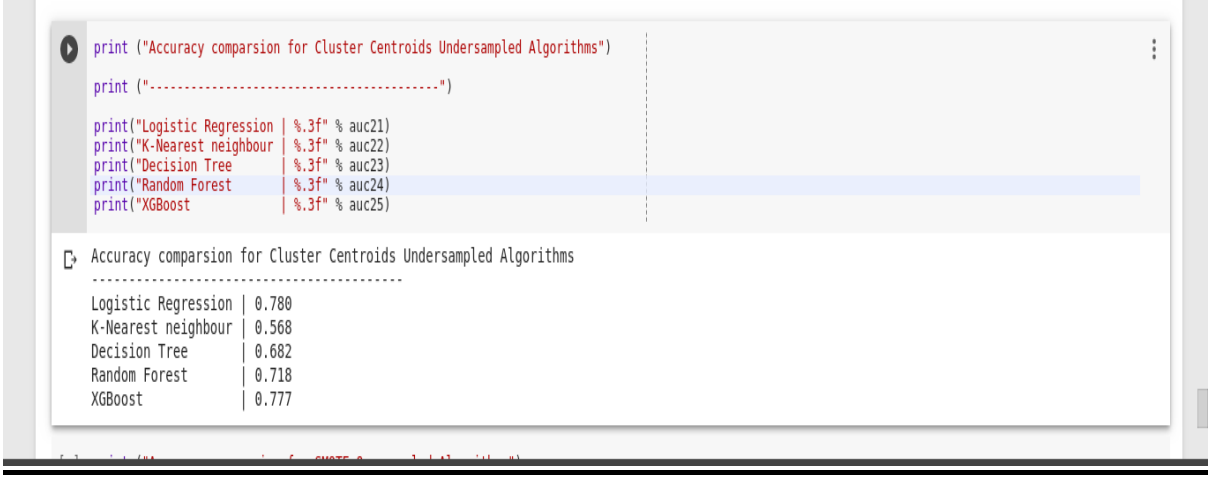
```
print ("Accuracy comparsion for TomekLinks Undersampled Algorithms")

print ("----------------------------------------")

print("Logistic Regression | %.3f" % auc16)
print("K-Nearest neighbour | %.3f" % auc17)
print("Decision Tree       | %.3f" % auc18)
print("Random Forest       | %.3f" % auc19)
print("XGBoost             | %.3f" % auc20)
```

```
Accuracy comparsion for TomekLinks Undersampled Algorithms
----------------------------------------
Logistic Regression | 0.790
K-Nearest neighbour | 0.576
Decision Tree       | 0.844
Random Forest       | 0.619
XGBoost             | 0.899
```

Fig 25. Comparison of Cluster centroids undersampled algorithms



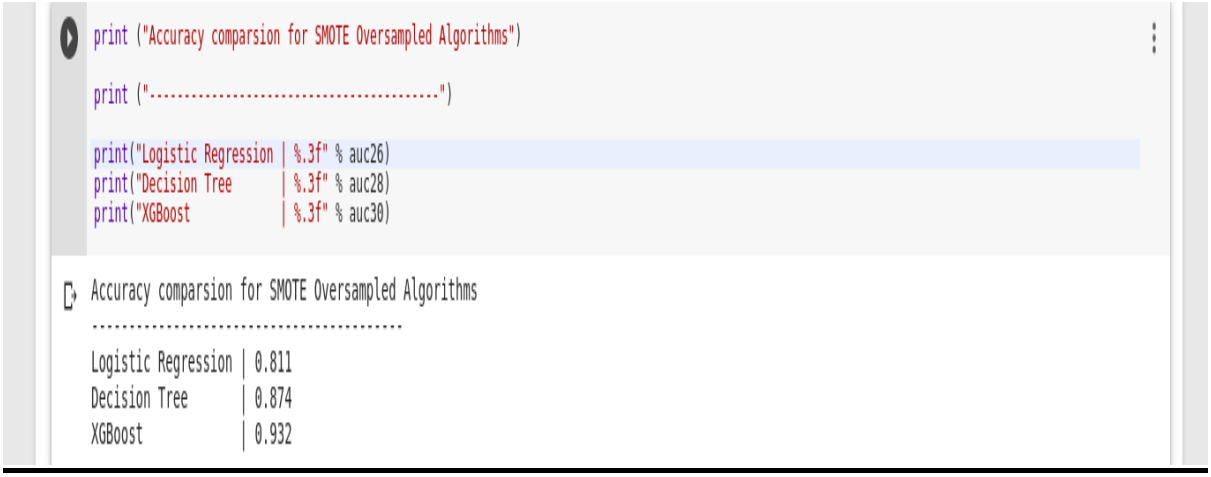
```
print ("Accuracy comparsion for Cluster Centroids Undersampled Algorithms")

print ("----------------------------------------")

print("Logistic Regression | %.3f" % auc21)
print("K-Nearest neighbour | %.3f" % auc22)
print("Decision Tree       | %.3f" % auc23)
print("Random Forest       | %.3f" % auc24)
print("XGBoost             | %.3f" % auc25)
```

```
Accuracy comparsion for Cluster Centroids Undersampled Algorithms
----------------------------------------
Logistic Regression | 0.780
K-Nearest neighbour | 0.568
Decision Tree       | 0.682
Random Forest       | 0.718
XGBoost             | 0.777
```

Fig 26. Comparison of SMOTE oversampled algorithms



```
print ("Accuracy comparsion for SMOTE Oversampled Algorithms")

print ("----------------------------------------")

print("Logistic Regression | %.3f" % auc26)
print("Decision Tree       | %.3f" % auc28)
print("XGBoost             | %.3f" % auc30)
```

```
Accuracy comparsion for SMOTE Oversampled Algorithms
----------------------------------------
Logistic Regression | 0.811
Decision Tree       | 0.874
XGBoost             | 0.932
```

Fig 27. Comparison of SMOTE+TL algorithms

```
print ("Accuracy comparsion for SMOTE+TL Algorithms")

print ("----------------------------------------")

print("Logistic Regression | %.3f" % auc31)
print("Decision Tree       | %.3f" % auc33)
print("XGBoost             | %.3f" % auc35)
```

```
Accuracy comparsion for SMOTE+TL Algorithms
----------------------------------------
Logistic Regression | 0.811
Decision Tree       | 0.874
XGBoost             | 0.932
```

Fig 28. Comparison of Logistic Regression

```
print ("Accuracy comparsion for Logistic Regression")

print ("----------------------------------------")

print("Normal                  | %.3f" % auc1)
print("Random Oversample       | %.3f" % auc6)
print("Random Undersample      | %.3f" % auc11)
print("Tomeklinks undersample  | %.3f" % auc16)
print("Cluster centroids       | %.3f" % auc21)
print("SMOTE                   | %.3f" % auc26)
print("SMOTE+TL                | %.3f" % auc31)
```

```
Accuracy comparsion for Logistic Regression
----------------------------------------
Normal                  | 0.792
Random Oversample       | 0.812
Random Undersample      | 0.812
Tomeklinks undersample  | 0.790
Cluster centroids       | 0.780
SMOTE                   | 0.811
SMOTE+TL                | 0.811
```

Fig 29. Comparison of KNN

```
print ("Accuracy comparsion for KNN")

print ("----------------------------------------")

print("Normal                  | %.3f" % auc2)
print("Random Oversample       | %.3f" % auc7)
print("Random Undersample      | %.3f" % auc12)
print("Tomeklinks undersample  | %.3f" % auc17)
print("Cluster centroids       | %.3f" % auc22)
```

```
Accuracy comparsion for KNN
----------------------------------------
Normal                  | 0.576
Random Oversample       | 0.576
Random Undersample      | 0.834
Tomeklinks undersample  | 0.576
Cluster centroids       | 0.568
```

Fig 30. Comparison of Decision Tree

```
print ("Accuracy comparsion for Decision Tree")

print ("----------------------------------------")

print("Normal                  | %.3f" % auc3)
print("Random Oversample       | %.3f" % auc8)
print("Random Undersample      | %.3f" % auc13)
print("Tomeklinks undersample  | %.3f" % auc18)
print("Cluster centroids       | %.3f" % auc23)
print("SMOTE                   | %.3f" % auc28)
print("SMOTE+TL                | %.3f" % auc33)
```

```
Accuracy comparsion for Decision Tree
----------------------------------------
Normal                  | 0.844
Random Oversample       | 0.841
Random Undersample      | 0.838
Tomeklinks undersample  | 0.844
Cluster centroids       | 0.682
SMOTE                   | 0.874
SMOTE+TL                | 0.874
```

Fig 31. Comparison of Random Forest

```
print ("Accuracy comparsion for Random Forest")

print ("----------------------------------------")

print("Normal                  | %.3f" % auc4)
print("Random Oversample       | %.3f" % auc9)
print("Random Undersample      | %.3f" % auc14)
print("Tomeklinks undersample  | %.3f" % auc19)
print("Cluster centroids       | %.3f" % auc24)
```

```
Accuracy comparsion for Random Forest
----------------------------------------
Normal                  | 0.615
Random Oversample       | 0.610
Random Undersample      | 0.856
Tomeklinks undersample  | 0.619
Cluster centroids       | 0.718
```

Fig 32. Comparison of XGboost

```
print ("Accuracy comparsion for XGBoost")

print ("----------------------------------------")

print("Normal                  | %.3f" % auc5)
print("Random Oversample       | %.3f" % auc10)
print("Random Undersample      | %.3f" % auc15)
print("Tomeklinks undersample  | %.3f" % auc20)
print("Cluster centroids       | %.3f" % auc25)
print("SMOTE                   | %.3f" % auc30)
print("SMOTE+TL                | %.3f" % auc35)
```

```
Accuracy comparsion for XGBoost
----------------------------------------
Normal                  | 0.899
Random Oversample       | 0.897
Random Undersample      | 0.886
Tomeklinks undersample  | 0.899
Cluster centroids       | 0.777
SMOTE                   | 0.932
SMOTE+TL                | 0.932
```

5. Discussions and Conclusion

Fraud detection is a particularly challenging and complex task. Fraudulent activities are rare events that are hard to model and in constant evolution. The large volume of transactions happening everyday demands automatic tools to support investigation, and the human resources devoted to investigations have to concentrate on the most suspicious cases. This report investigated how machine learning algorithms could be used to address some of these issues. In particular, we focused on the design of a framework that is able to report the riskiest transactions to investigators by means of algorithms that can deal with unbalanced and evolving data streams.

Now we will discuss future research directions .The future is the design of mechanisms based on machine learning and big data mining techniques that allow to automatically detecting attacks and fraudulent behaviours in large amounts of transactions. Developing a real-time framework that is able to compare in parallel a large number of alternative models in terms of nature (expert-based or data driven), features (e.g. history or customer related), data (e.g. supervised or unsupervised), predictive methods (e.g. decision trees, neural networks), scalability (e.g. conventional versus Map/Reduce implementation) and quality criteria (e.g. readability vs. accuracy). The framework is expected to be scalable with respect to the amount of data and resources available. The project will also investigate the exploitation of network data (social networks, communication networks, etc.) for fraud, privacy and security purposes. The use of network data is currently a highly studied field, subject of much recent work in various areas of science.

Now we want to discuss how the results of this project could be valorised by the industries. The FDS in production is currently adopting a static approach, i.e. a classification model updated once/twice a year. The results show to the company that there is a clear performance gain when the model is updated more frequently, e.g. once a week or every 15 days. We also used different classification algorithms. In particular, XgBoost has emerged as the best algorithm in many simulations. We actually used 5 different algorithms i.e. Logistic Regression, K nearest neighbours, Decision tree, Random forest and XGBoost Algorithms. We concluded that XGboost performed far better than other algorithms in all scenarios.

The results of also showed that, in case of fraud detection, the performance of a classifier can be significantly improved when sampling methods are used to rebalance the two classes.

Given the large imbalance ratio and number of transactions, over sampling should perhaps be favoured w.r.t. under sampling techniques. We have analysed various different types of oversampling and undersampling techniques. The various types were Random oversampling and SMOTE technique for oversampling and random, tomek links and cluster centroids for undersampling.We also used a combination of SMOTE and Tomek links. We found out that SMOTE and SMOTE with Tomek Links gave very good results as compared to others. Cluster centroids performed the worst.

.

6. References

[1] Masoumeh Zareapoor, Pourya Shamsolmoali, "Application of Credit Card Fraud Detection", International Conference of Intelligent Computing, Communication & Convergence 2015.

[2] S.Vimala, K.C.Sharmili, "Survey Paper for Credit Card Fraud Detection Using Data Mining Techniques", International Journal of Innovative Research in Science, Engineering and Technology September 2017

[3] Andrea Dal Pozzolo, "Adaptive Machine Learning for Credit Card Fraud Detection Thesis"

[4] Snehal Patil et al, "Credit Card Fraud Detection Using Decision Tree Induction Algorithm",International Journal of Computer Science and Mobile Computing, Vol.4 Issue.4, April- 2015.

[5] Dr R.Dhanapal, Gayathiri.P, "Credit Card Fraud Detection Using Decision Tree For Tracing Email And IP", IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 5, No 2, September 2012.

[6] Raghavendra Patidar, Lokesh Sharma, "Credit Card Fraud Detection Using Neural Network", International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-1, Issue-NCAI2011, June2011.

[7] Dinesh L. Talekar, K. P. Adhiya, "Credit Card Fraud Detection System: A Survey", International Journal Of Modern Engineering Research (IJMER),Vol. 4, Iss.9, Sept.2014.

[8] Mr.P.Matheswaran, Mrs.E.SivaSankari ME, Mr.R.Rajesh, "Fraud Detection in Credit Card Using DataMining Techniques, Volume II, Issue I", IJRSET-February- 2015.

[9] Yogesh Bharat Sonawane , Akshay Suresh Gadgil , Aniket Eknath , Niranjan Kamalakar Jathar , "Credit Card Fraud Detection Using Clustering Based Approach", IJARIIE-ISSN(O)-2395-4396 Vol-2 Issue-6 2016.