**Project Report For**

**Credit Card Fraud Detection**

|  |  |
| --- | --- |
| Name | Amit Padgaonkar |
| Student ID | 862188375 |
| Date of Submission | 08-Jun-20 |

# Project Description

Credit card fraud happens when a fraudster or a thief tries to use stolen credit card to make unauthorized purchases or take out cash advances. Credit card fraud is a cause of financial losses worth billions every year. Apart from financial loss, it also erodes the public trust in the ways modern financial industry and institutions [1]. To address these issues, all organizations in the payment processing industry are trying to implement systems to reduce or eliminate fraud.

One of the very effective ways to tackle this issue in real time is through discovery of patterns in fraudulent transactions. Machine Learning can be utilized very effectively to understand the patterns of fraud and prevent it in its track in real-time.

To build this model, I am going to use the credit card transactions dataset provided by ULB (Université Libre de Bruxelles) [version 3] which is hosted at Kaggle [2].

# About the Dataset

The dataset provided by ULB, contains 284,407 credit card transactions made by European cardholders in a couple of days of September, 2013. The dataset has already been pre-processed by applying Principle Component Analysis (PCA). All the features in the dataset are numeric. Also, the original features and background information has been masked for confidentiality reasons. We know only about 3 features, Class & Time & Amount, which have been provided in their original form. Let us begin by exploratory analysis of the dataset

# Exploratory Analysis

The dataset has 284,807 records and 31 columns. All the values in the dataset are numeric. This saves us time required for conversion from text to numeric values. Also, the dataset has no null values. This obviates the need to impute the missing values.

The fraud transactions are indicated by column Class value 1 and non-fraudulent transactions are indicated by column Class value 0. The model that we will build will try to predict this target variable. From this information, let us see the distribution of our data based on target variable class.

**Fraudulent (Class 1) vs Non-Fraudulent Transactions (Class 0)**



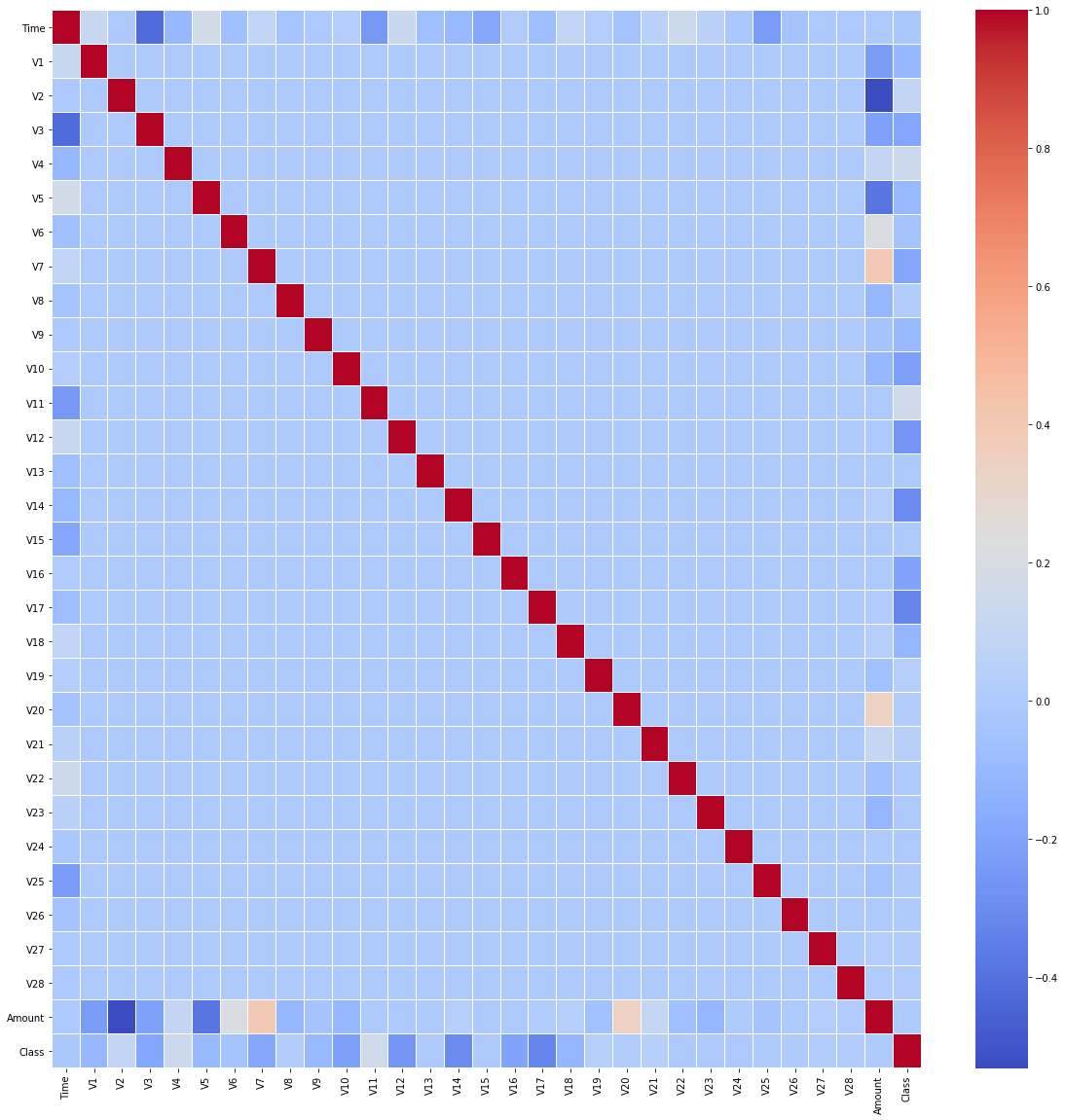
The counts of Fraudulent and legitimate transactions are

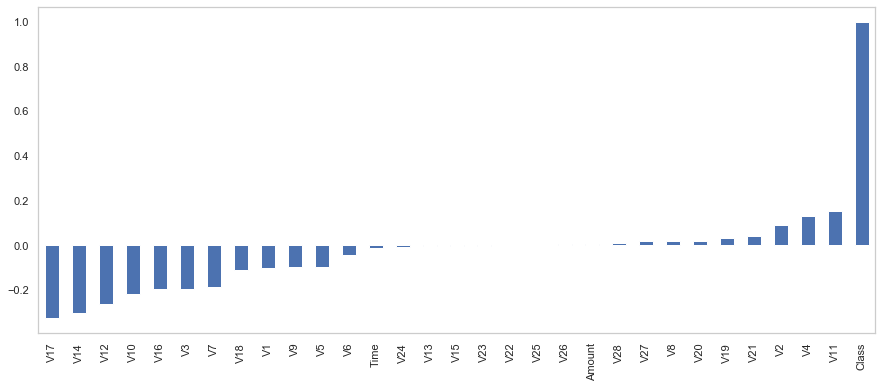
|  |  |  |
| --- | --- | --- |
| **Total** | **Fraudulent** | **Legitimate** |
| 284807 | 492 | 284,315 |

In the chart, we can barely see any bar standing up to represent fraudulent transactions. From the chart and from the count we can see that this is a heavily imbalanced dataset where only **0.172%** of the transactions belong to the positive class.

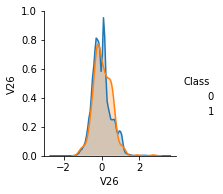
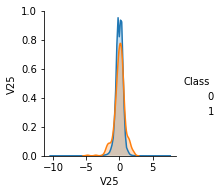
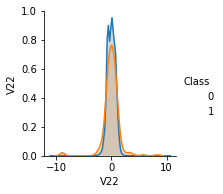
**This is a heavily imbalanced dataset** where we will be looking for needles in the haystack.

Now let us try to identify correlations in our dataset to identify the impact of the features on the target variable.





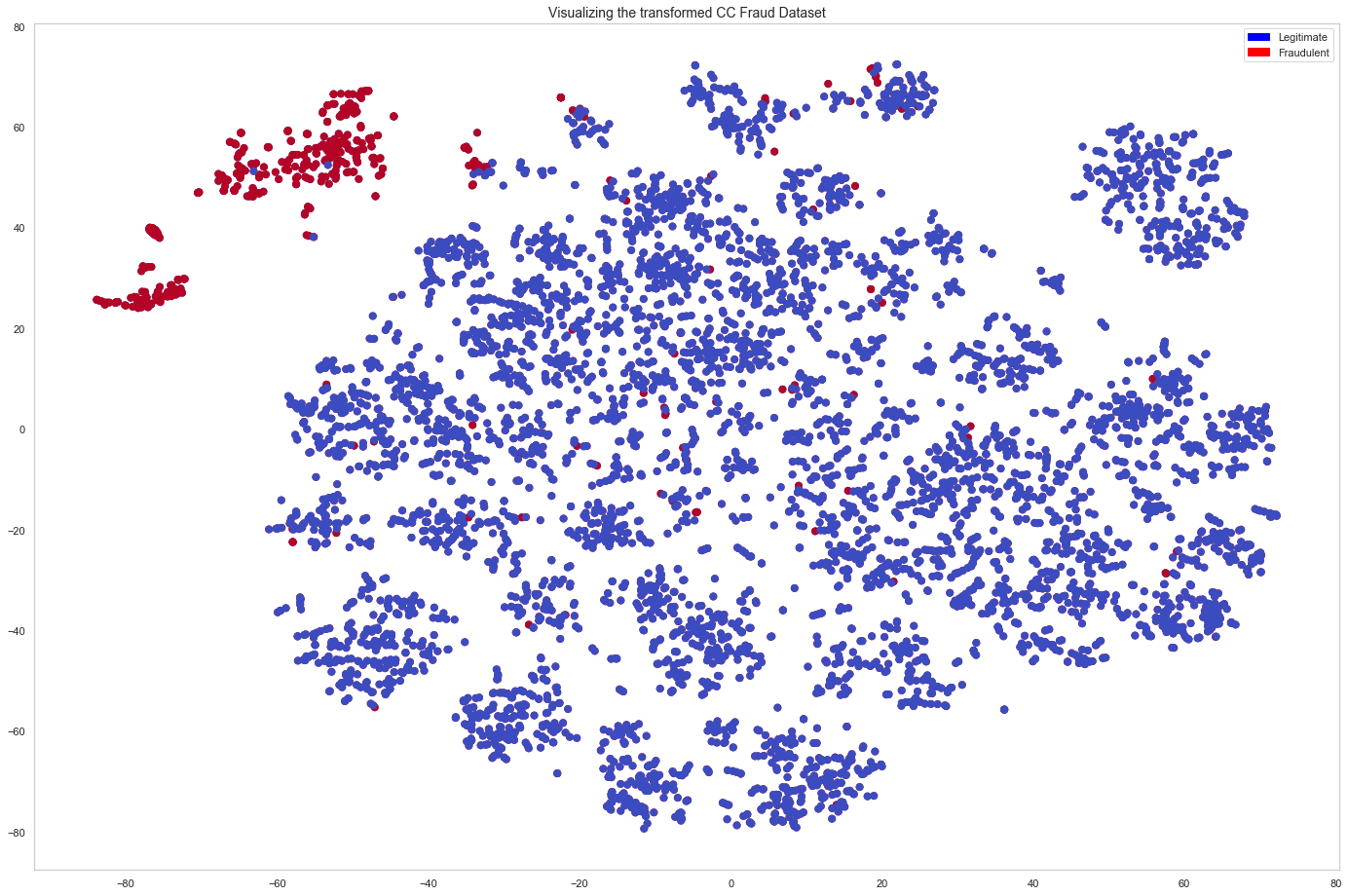
We can see that V22, V25 & V26 have very minimal to none impact on the target variable. Let us also see the plot of distribution of values for V22, V25 & V26 which are created using Seaborn visualization library.



Based on the distribution, we can see that these values do not have significant differences for Class 1 (fraudulent) and Class 0 (legitimate) transactions.

Since our data has 31 dimensions, it would be difficult for us to visualize it in 2 or 3 dimensional chart. In order to overcome this limitation, we will use t-SNE, which is a nonlinear dimensionality reduction technique used for embedding high dimensional data in low dimensional space for visualization [4]. We will reduce the data to 2 dimensions and plot it.

To plot this data we will use all the available fraudulent transactions (492) a subsample of the legitimate transactions (about 2%)



**Data Visualization in 2-dimensional space using T-SNE**

From the chart we can clearly see some clear clusters of fraudulent (**red**) and non-fraudulent (**blue**) transactions. However, we can also see quite a few red dots interspersed in the sea of blue. A simple eyeballing the chart tells us that about 10-20% of the fraudulent transactions can present challenges in identification.

# Challenges

From our exploratory data analysis we can now list our challenges in creating a successful model for fraud detection.

* **Futility of Traditional ML Algorithms**

In our data exploration phase, we observed that our dataset is heavily imbalanced. Most of the machine learning algorithms are designed to work on classification data with equal number of observations [4]. As the instances of minority class are very rare, the algorithms tend to treat them as outliers and simply ignore them [5]. However, it is this positive class (fraud cases) that we are interested in identifying. Since class distribution is not balanced, most traditional machine learning algorithms will perform poorly.

* **Right Metric for Evaluation**

An additional challenge for out heavily imbalanced dataset is coming up with right metric to determine the effectiveness of the model. Since only 0.172% of the transactions are fraudulent, even a no-skill prediction technique will produce an accuracy of 99.82%. This means that we will have to use a metric apart from accuracy to evaluate our model [6].

* **Lack of Data for Minority Class**

Typically, more data is better as it provides better representation of combination and variance features and their mapping to class labels. With more data, machine learning algorithms learn better. However, in our dataset, we have only 492 minority class transactions, which is far from sufficient for any ML algorithm to create effective to learn to separate it from majority class transactions. This means we will need to find ways to bring balance to our dataset. [7]

* **Perfect Separation of Fraudulent Transactions is Very Difficult**

In the data exploration phase, we saw the chart created by t-SNE technique to visualize our data in 2-dimension. Although, we can see some distinct red & white clusters, we can also see that a lot of fraudulent & legitimate transactions were interspersed with each other. This means that identifying these difficult to distinguish fraudulent transactions cannot be done without creating a large number of false positives, reducing the accuracy of our algorithm in turn.

# Data Pre-Processing

In the data pre-processing stage we will try to address some of the challenges mentioned above using the visualizations from data exploration phase

* **Exclude Insignificant Columns**

We saw that V22, V25 & V26 have very minimal to none impact on the target variable. We will simply drop these columns

* **Use Logarithmic Scale for Amount Column**

The amount values vary widely from 0 to 25691.16. To better suit these values to our algorithm we will scale them logarithmic scale

* **Stratified Data Sampling**

For training & testing, we will need to split the data. We saw that we have very few samples of minority class (492). If we use a regular splitting method, we can have very few samples in our validation. In order to avoid that, we need to split the dataset in such a way that it maintains the same class distribution in each subset. This technique is known as Stratified Sampling. [8]

When we split the data using 70/30 split using stratified sampling, our dataset looks good.

|  |  |  |
| --- | --- | --- |
|  | Train | Test |
| Legitimate(Class 0) | 199020 | 85295 |
| Fraudulent(Class 1) | 344 | 148 |

* **Scaling the Dataset**

It is a common requirement for most Machine Learning algorithms to standardize the data. Since we have negative values and we are dealing with a classification problem, the appropriate scaler for our data is **StandardScaler.** StandardScaler transforms the data in such a way that it has a mean of 0 and standard deviation of 1. [9]

* **Oversampling the Minority Class Data**

We have an imbalance in our dataset in terms of minority and majority class samples. If the dataset is heavily skewed, i.e. samples of one class are significantly less than the other class, then the ML algorithms will tend to treat the minority class as insignificant. To address this issue, we will need to increase the number of samples available for minority class. This approach is called Oversampling [10].

Perhaps the most popular oversampling method is Synthetic Minority Oversampling Technique (SMOTE). SMOTE works by selecting examples that are close in the feature space, drawing a line between examples in the feature space and drawing a new sample as point along that line. There are extensions of SMOTE, involves creating samples that are difficult to classify using algorithms such as K-Nearest Neighbors. Borderline Oversampling is an extension to SMOTE that fits an SVM to the dataset and uses the decision boundary as defined by the support vectors as the basis for generating synthetic examples. Since we want to train our algorithm on difficult samples to increase accuracy and sensitivity, we will use **SVMSMOTE,** which is based on support vector machine algorithm to generate new minority class instances near borderline with SVM [11].

Using SVMSMOTE, we have equal counts for samples in both majority and minority class



# Performance Evaluation Metrics

A classifier is only as good as the metric used to evaluate it. For classification problems, metrics involve comparing the expected class label to the predicted class label or interpreting the predicted probabilities for the class labels for the problem. This Challenge is made even more difficult when there is a skew in the class distribution. The reason for this is that many of the standard metrics become unreliable or even misleading when classes are imbalanced. As an example, in our case, we can see that 99.82% of the samples belong to the majority class. Even an algorithm with no-skill would be able to reach a very high accuracy in this case while misclassifying all of the minority class samples. However, it is the prevention of fraud or correct identification of minority class samples that we are most interested in.

However, the model that will identify most of the fraud transactions will also tend to reject a lot of legitimate transactions distorting the user experience**. If we reject a lot of legitimate transactions, the manual review of these transactions will probably cost more than the possible loss of sum in the fraud that is prevented. Also, if users find our system too restrictive, they may simply choose not to do business with us. Hence, it really becomes a BALANCING ACT to tune the model in such a way that fraud is prevented, however most of transactions of the genuine users go through smoothly.** So we will have to increase the True Positives while decreasing the False Negatives (more important) and False Positives.

For the legitimate transactions to flow through smoothly, we will need to have **high accuracy,** which will an important metric for us. However, more than that we need to have an emphasis on detecting the fraudulent transactions correctly which is indicated by number of True Positives. To know whether, we are identifying high number of fraudulent transactions correctly, the right metric would be **Recall Aka Sensitivity**.

Another important metric that we will take a look at is **AUC-ROC** Curve [12]. It is probability curve that plots True Positive Rate against False Positive Rate at various thresholds and separates signal from noise. Area Under the Curve (AUC) is the ability of algorithms to distinguish between classes and it is used as summary of ROC Curve.

Apart from that we will be calculating other metrics such as Precision, F1-Score, F2-Measure as well, however we will be focused mostly on Recall (Sensitivity), Area Under the Curve (AUC) and Accuracy. I will list the formulas for these for convenience

Sensitivity = (True Positive)/ (True Positive + False Negative)

Accuracy = (TP + TN)/ (TP + TN + FP + FN)

Precision = (True Positive) / (True Positive + False Positive)

F-Measure = (2 \* Precision \* Recall) / (Precision + Recall)

F2-Measure = ((1 + 2^2) \* Precision \* Recall) / (2^2 \* Precision + Recall)

Now that we have done everything necessary to create & evaluate an algorithm suitable for our problem, let’s get right into creating some algorithms and evaluate their performance against our model.

# Algorithms

Most machine learning algorithms assume that all misclassification errors are equal. However, this is not true in case of imbalanced classification where missing a positive class (fraud case) is worse than incorrectly identifying a legitimate transaction as fraudulent. For our dataset, we will need algorithm(s) where we can assign a punishment to misclassifying the positive class. If we use traditional classification algorithms such as k-Nearest Neighbor or SVM, they may not be able to provide us these options for fine-tuning to satisfy our performance metrics.

For our purposes we will consider broadly 2 types of algorithms for evaluation

* **Ensemble Algorithms**

Ensemble algorithms are advanced algorithms which help machine learning by combining several models. This approach creates better results than what a single model would predict. Bagging is an ensemble algorithm that works with multiple models on different subsets of training dataset and then combines the results. [13] Random Forest is an extension of Bagging. Although they are not directly suitable for imbalanced classifications, there are tweaks that can be done to enhance their performance substantially.

* **Cost-Sensitive Algorithms**

Cost-Sensitive Algorithms is a field of machine learning which takes into account

the fact that cost of predicting values incorrectly for different classes can be significantly different. The techniques developed for cost-sensitive learning can be used for imbalanced classification problems. [14]

## Ensemble Algorithms

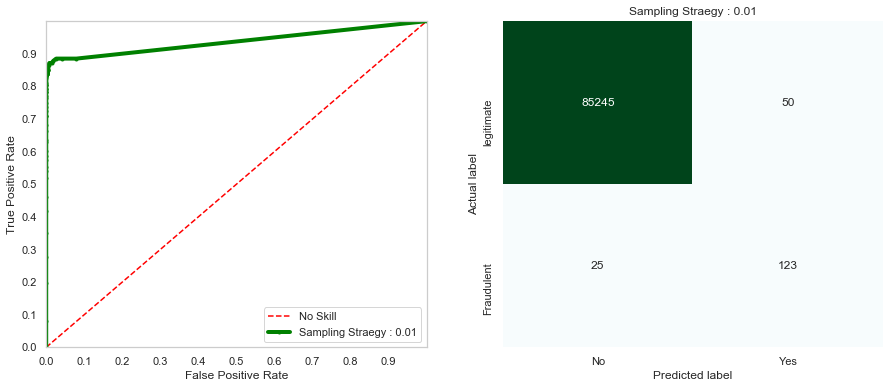
### **Algorithm – Balanced Bagging Classifier**

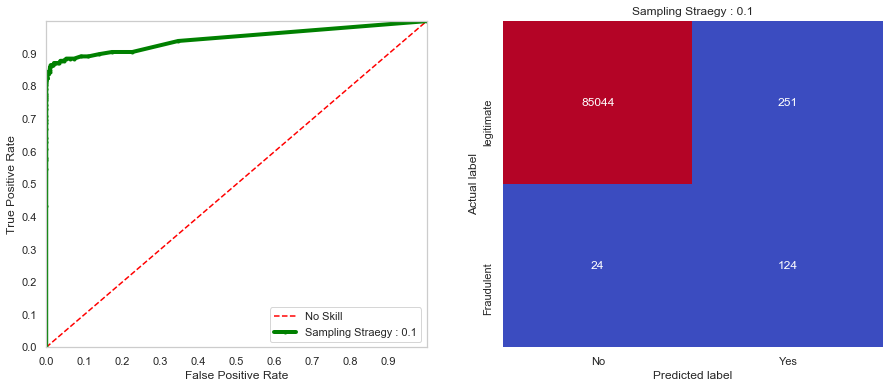
Bootstrap Aggregation or Bagging is an ensemble machine learning algorithm. It selects random samples from the dataset with replacement and create one weak learner to fit the data on each sample. Finally the predictions from fit weak learners are combined to make a single prediction.

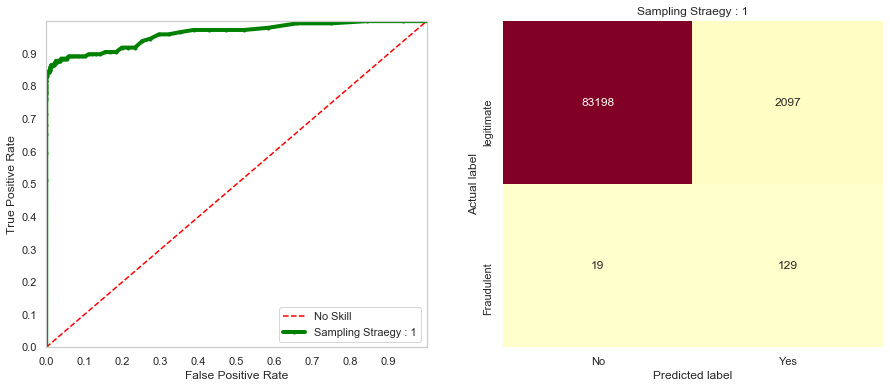
A simple Bagging algorithm will create bootstrap samples that will not take into account skewed class distribution for imbalanced classification datasets. One of the ways is to apply data sampling on bootstrap sample prior to fitting the weak learner model. This is done in the imbalanced-learn library’s BalancedBaggingClassifier class. It provides a version of bagging that uses random undersampling strategy on majority class to balance the 2 classes. In practice it is controlled by the parameter sampling\_strategy, which specifies the desired ratio of number of samples in the minority class to the number of samples in the majority class. [15]

**Results**

We have used different values for sampling-strategy and was able to get following results.







|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sampling Strategy | Confusion Matrix | | Recall (Sensitivity) | AUC-ROC | Accuracy |
| 0.01 | 85245 | 50 | 0.831081 | 0.937494 | 0.999122 |
| 25 | 123 |
|  |  |  |  |  |  |
| 0.1 | 85044 | 251 | 0.837838 | 0.945874 | 0.996781 |
| 24 | 125 |
|  |  |  |  |  |  |
| 1 | 83198 | 2097 | 0.871622 | 0.961538 | 0.975235 |
| 19 | 129 |

Balanced Bagging Classification has provided reasonably good results, both balancing

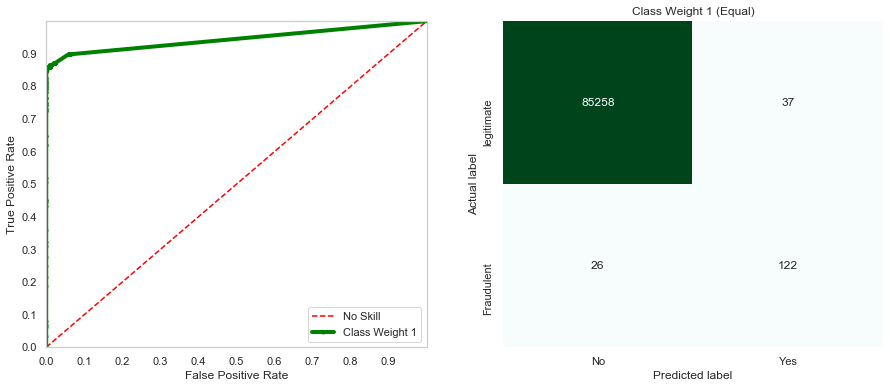
Accuracy and Sensitivity. Moreover, by tweaking the value of sampling strategy, we can change the number of majority classes used for training the algorithm, thereby changing the output parameters.

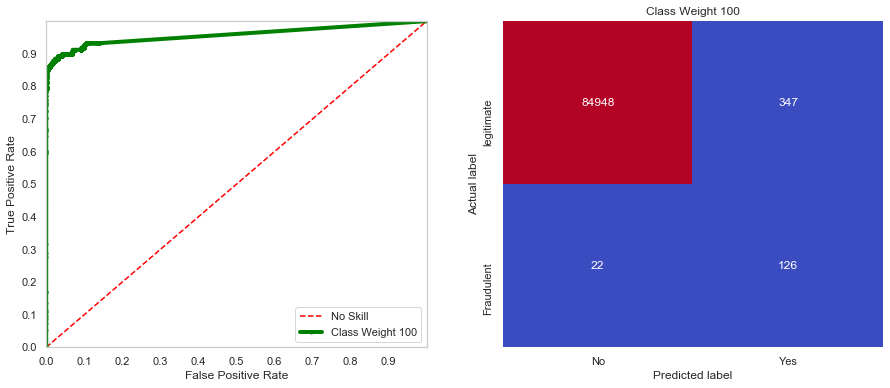
### **Algorithm – Random Forest Classifier with Class Weighing**

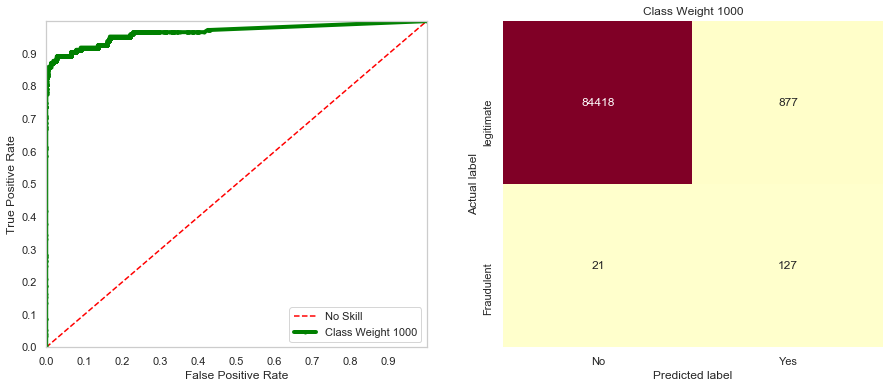
Random Forest is an extension of Bagging. Similar to Bagging, Random Forest selects bootstrap samples from the training dataset to fit a decision tree. However, instead of selecting all features, only a random subset of features is chosen for each bootstrap sample. [16] This makes the decision-trees more independent thus improving the ensemble prediction.

Random Forest can be very easily modified to change the weight used for each class. Weight of each class is used to calculate the impurity score at chosen split point. The calculation can be changed so that minority class is favored while allowing some more number of false positives for majority class. In practice, this is done through a parameter *class\_weight* which specifies the importance given to a specific class value.

**Results**

****

****

****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Class\_Weight | Confusion Matrix | | Recall (Sensitivity) | AUC-ROC | Accuracy |
| {0:1, 1:1} | 85258 | 37 | 0.831081 | 0.915347 | 0.999321 |
| 26 | 122 |
|  |  |  |  |  |  |
| {0:1, 1:100} | 84948 | 347 | 0.851351 | 0.923483 | 0.995365 |
| 22 | 126 |
|  |  |  |  |  |  |
| {0:1, 1:1000} | 84118 | 877 | 0.871622 | 0.930172 | 0.988519 |
| 21 | 127 |

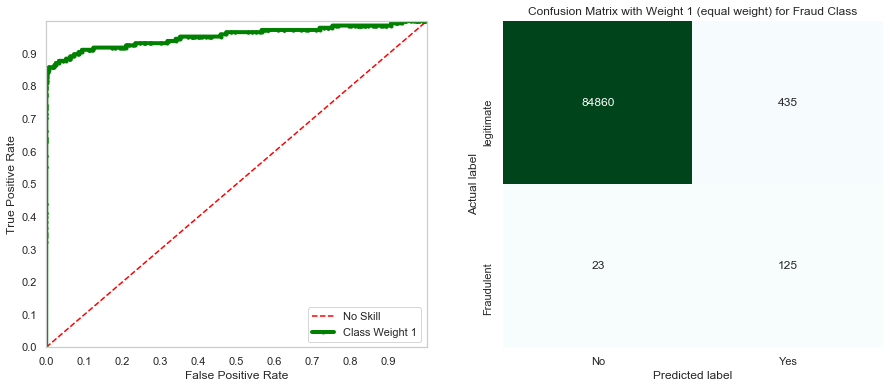
Using Random Forest Classifier with Class Weighing we achieved better results than Balanced Bagging Classifier. Moreover, using the class weights, we can easily tune the Accuracy & Recall. By changing the weights assigned to each class, we can calibrate the desired outcome as per the business requirements.

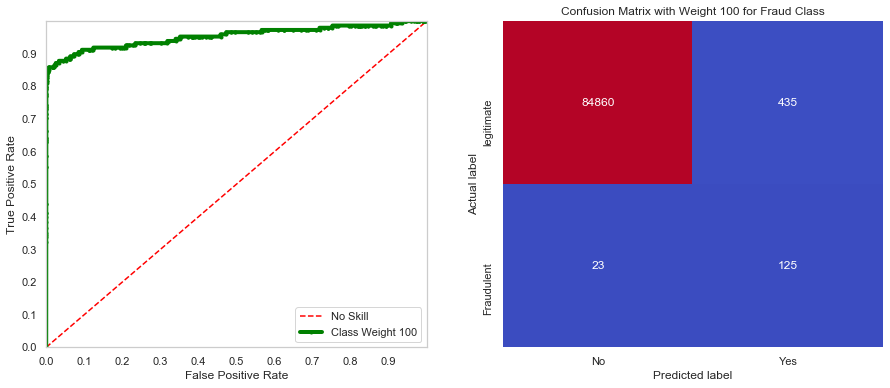
## **Cost-Sensitive Algorithms**

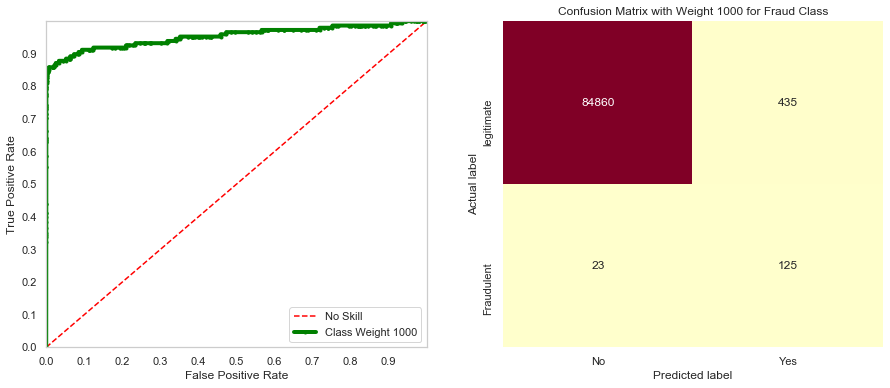
### **Algorithm – Weighted Logistic Regression**

Logistic Regression algorithm models the probability of a class. Logistic Regression does not support imbalanced classification directly. [17] However, the training algorithm used to fit the logistic regression can specify the class weight configuration to influence the logistic regression coefficients. The class weight specified determines the importance that is given to update the model coefficients.

**Results**







|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Class\_Weight | Confusion Matrix | | Recall (Sensitivity) | AUC-ROC | Accuracy |
| {0:1, 1:1} | 84860 | 435 | 0.844595 | 0.953544 | 0.994640 |
| 23 | 125 |
|  |  |  |  |  |  |
| {0:1, 1:100} | 84860 | 435 | 0.844595 | 0.953544 | 0.994640 |
| 23 | 125 |
|  |  |  |  |  |  |
| {0:1, 1:1000} | 84860 | 435 | 0.844595 | 0.953544 | 0.994640 |
| 23 | 125 |

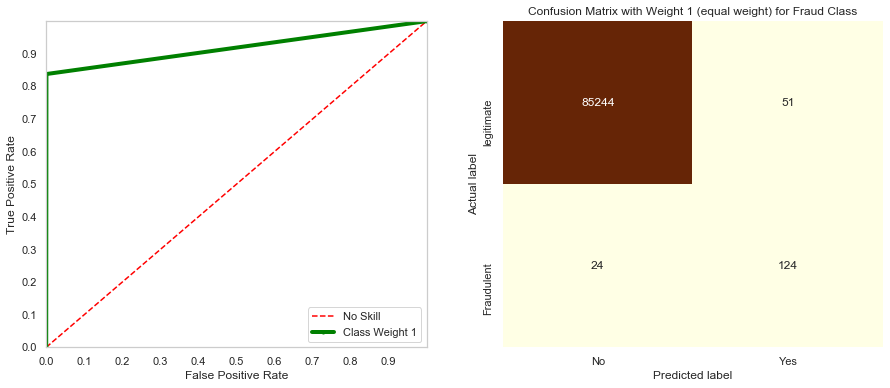
Using Logistic Regression with class weights has been disappointing. Not only the metrics such Accuracy, Recall, AUC-ROC were not as good as other algorithms, the change in class weights didn’t have any influence on the results. We have evaluated this algorithm as a possible candidate solution and it fails to pass the scrutiny.

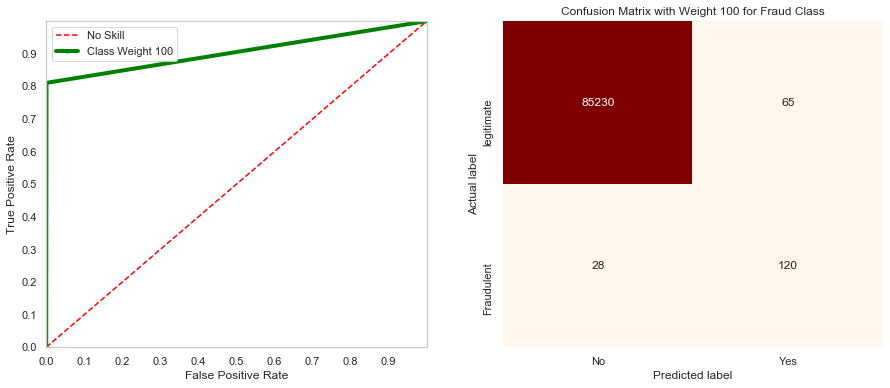
### **Algorithm – Cost Sensitive Decision Trees**

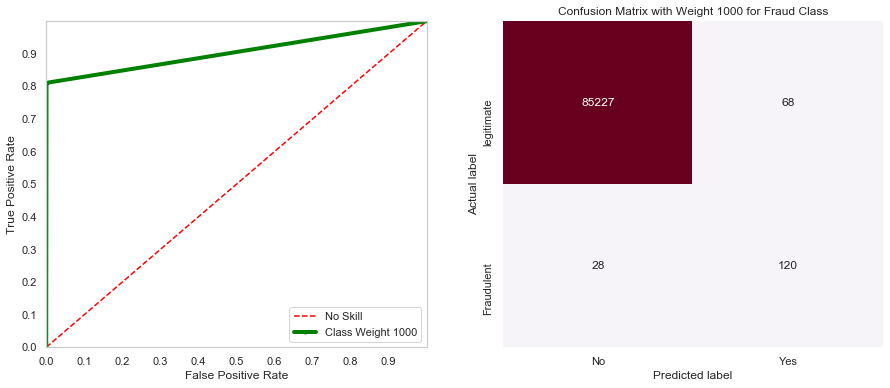
The Decision Tree algorithm has been designed to perform well on balanced datasets. The nodes of the tree chose the best separation to divide into 2 groups. However, when the dataset is imbalanced, the criteria used to select split point will see good separation (resulting in good accuracy), while the minority class examples will get ignored.

To overcome this issue, we need to modify the criteria used to calculate the impurity at the split points by assigning different importance to each class. Similar to other decision tree based algorithms we saw, we pass a parameter class\_weight which is used to calculate the purity score at the nodes [18].

**Results**

****

****

****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Class\_Weight | Confusion Matrix | | Recall (Sensitivity) | AUC-ROC | Accuracy |
| {0:1, 1:1} | 85244 | 51 | 0.837838 | 0.91862 | 0.999099 |
| 24 | 124 |
|  |  |  |  |  |  |
| {0:1, 1:100} | 85230 | 65 | 0.797297 | 0.905024 | 0.998865 |
| 28 | 120 |
|  |  |  |  |  |  |
| {0:1, 1:1000} | 85227 | 68 | 0.810811 | 0.905007 | 0.998771 |
| 28 | 120 |

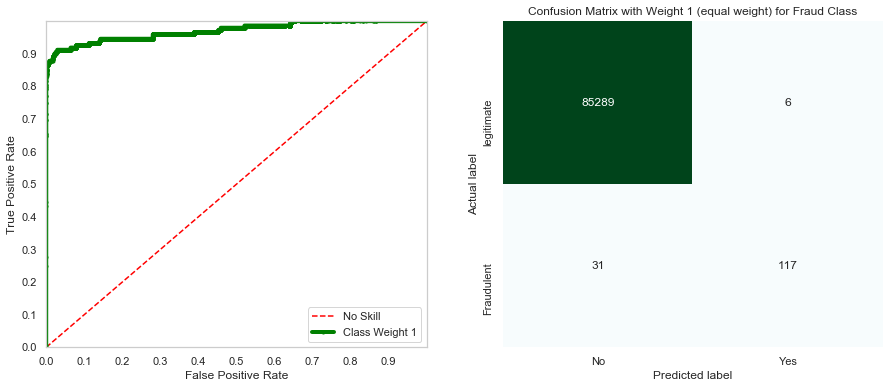
Although, the Accuracy of the algorithm was high, overall the results were contrary to the expectations. Increase in the class weight for the minority class did not result in corresponding increase in Recall and AUC-ROC.

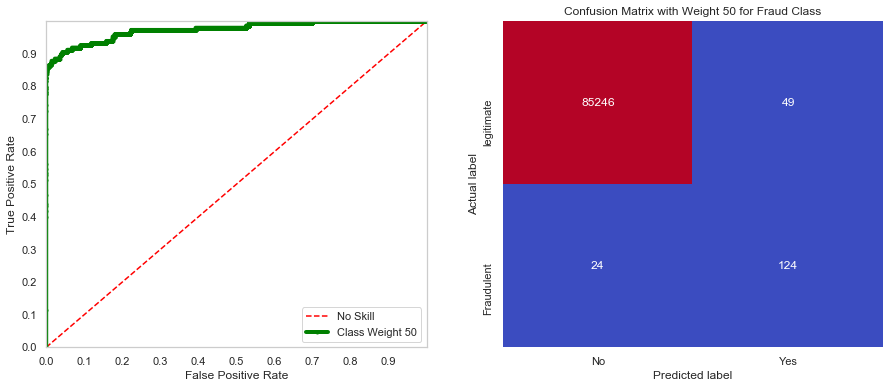
### **Algorithm – Cost Sensitive Gradient Boosting XGBoost**

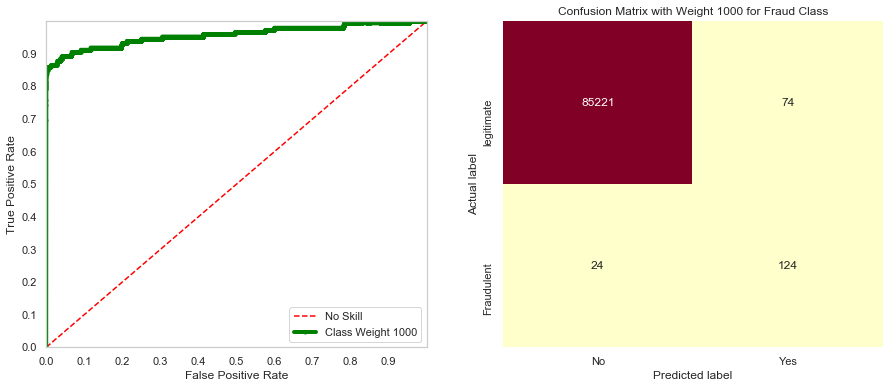
XGBoost (Extreme Gradient Boosting) algorithm is an efficient implementation of the stochastic gradient boosting machine learning algorithm and is very effective for a range of regression and classification predictive modelling problems. XGBoost provides access to a range of model hyperparameters to provide control over the training process [19]. XGBoost can work well even on imbalanced datasets, however it offers to fine-tune the training algorithm to pay more attention to the misclassification of the minority class. The hyperparameter it offers for this purpose is *scale\_pos\_weight*. [20]

With *scale\_pos\_weight* , we can control the balance of positive and negative weights. The other parameters also available with XGBoost are max\_delta\_step which is used to constrain the value of delta step of each leaf output. This helps in controls the updates.

**Results**

****

****

****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Scale\_Pos\_Weight | Confusion Matrix | | Recall (Sensitivity) | AUC-ROC | Accuracy |
| 1 | 85289 | 6 | 0.790541 | 0.970665 | 0.999567 |
| 31 | 117 |
|  |  |  |  |  |  |
| 50 | 85275 | 49 | 0.824324 | 0.974359 | 0.999146 |
| 24 | 122 |
|  |  |  |  |  |  |
| 1000 | 85221 | 74 | 0.837838 | 0.957629 | 0.998853 |
| 24 | 124 |

XGBoost, true to its reputation, has given **very solid results**. Even when the class weights were increased, the accuracy remained very high (above 0.9992). This is the only algorithm so far which has been able to achieve this feat for this dataset. Also, with the increase in the scale\_pos\_weight parameter, we were able to tweak the results to increase the recall and AUC-ROC while at the same time accuracy levels remained solidly in the high range.

### **Algorithm – Cost Sensitive Artificial Neural Network**

Artificial Neural Networks are machine learning algorithms that perform very well on wide range of problems. Neural Networks use backpropagation algorithm that involves calculating errors and updating model weights. However, these methods treat each class the same. This results in models treating majority class preferentially compared to the minority class for imbalanced datasets. [21]

However, this can be changed by tuning the neural networks to treat minority class preferentially compared to majority class. This can be achieved by changing the weightage of error scores based on their class [22]. A large error weight is applied to minority class examples as they are more important for imbalanced classification

The actual tuning is done by passing a parameter *class\_weight* while performing a fit on the training data. This hyperparameter assigns different weights to majority class and minority class samples, so that the model treats these 2 differently.

Also, Neural Network is especially suited for this specific dataset for following reasons

1. *Elimination of Need for Feature Engineering*

Artificial Neural Networks (ANN) have ability to learn and model complex relationships. This is especially the case when their prior relationship is not well-known. ANNs try to learn from features of the data in an incremental manner and which usually need to be identified by a domain expert in traditional machine learning. Here, **we have an anonymized dataset** which makes it harder to carry out any business analysis on the data. Hence, it makes sense to allow ANN to figure out these relationships for us.

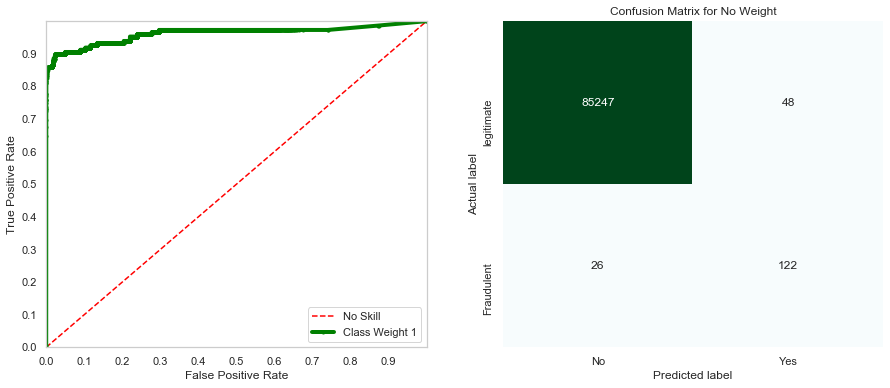
1. *High Level of Configurability*

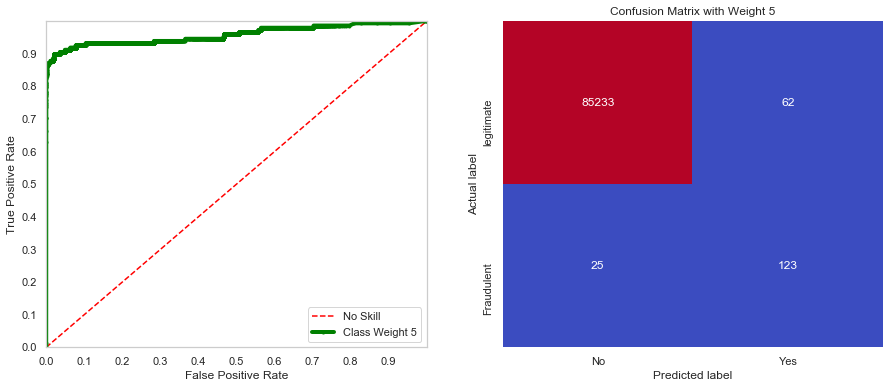
ANNs have achieved accuracies that are far beyond that of classical ML methods in many domains including speech, natural language, vision, and playing games. In many tasks, classical ML can’t even compete with the Neural Networks. Also, we will see in the results that by changing the class weights, the results can be tweaked ad infinitum. This was not reasonably possible with rest of the models that we have seen so far.

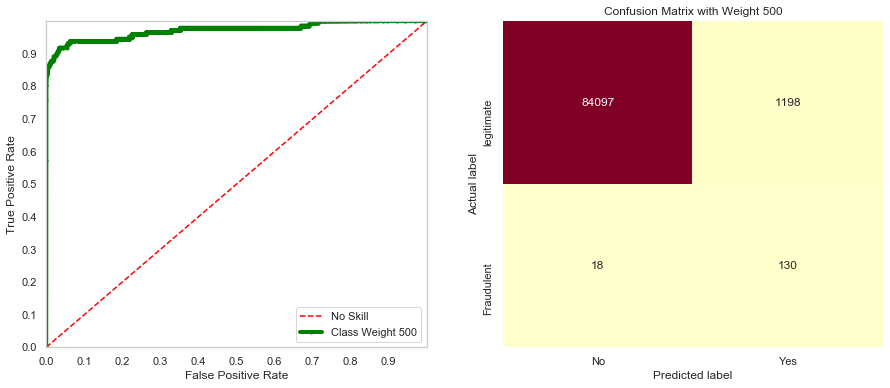
The outline of the 3 ANN models that we have created is below

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Activation  Function | Cost Function |
| Input Layer | No of input features | tanh |  |
| 2 Hidden Layers | 32,32 | tanh |  |
| Output Layer | 1 | sigmoid | binary\_crossentropy |
| Regularization | Dropout regularization (0.4) | | |
| Optimizer | Adam | | |
| Cost Sensitive Weights | This will cause the model to pay more attention to fraud class (1)  {0:1, 1:1} : No Weight  {0:1, 1:5} : Weight 5 for fraud class  {0:1, 1:500} : Weight 500 for fraud class | | |
| Early Stopping | Precision (Although FNs are costly, FPs are also costly for punishing the legitimate transactions. Also, model showed significantly higher accuracy [about 99 %]. The precision is made the parameter for early stopping. | | |

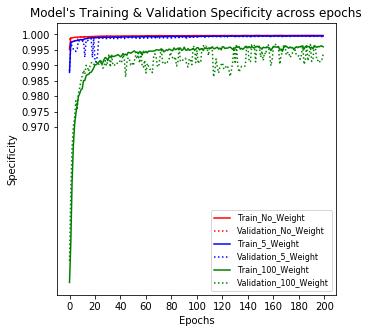
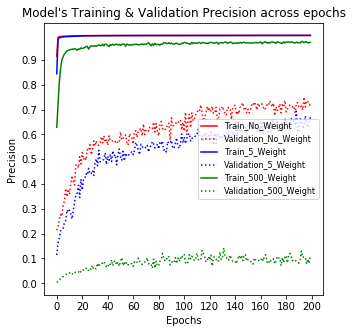
**Results**

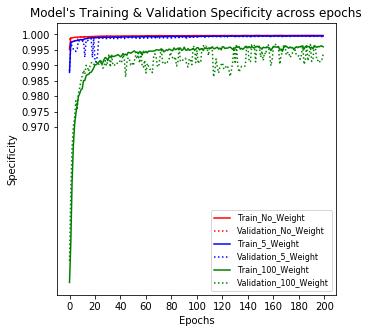
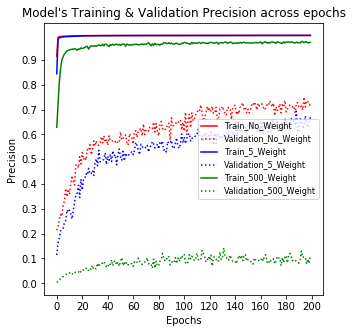






Since the Artificial Neural Networks run across epochs, we were able to get some interesting charts of changes in accuracy, sensitivity, precision and specificity across epochs.



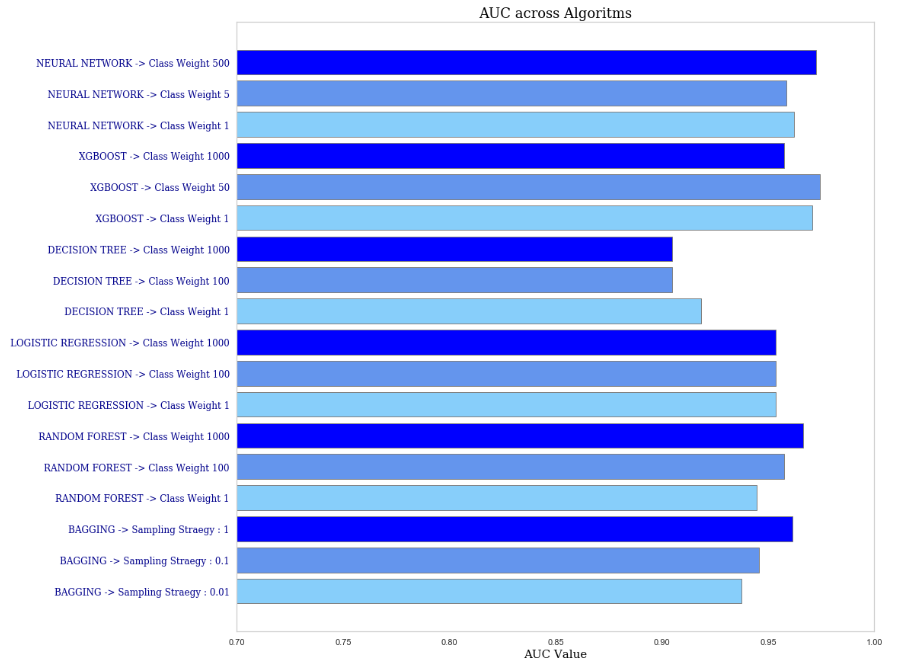


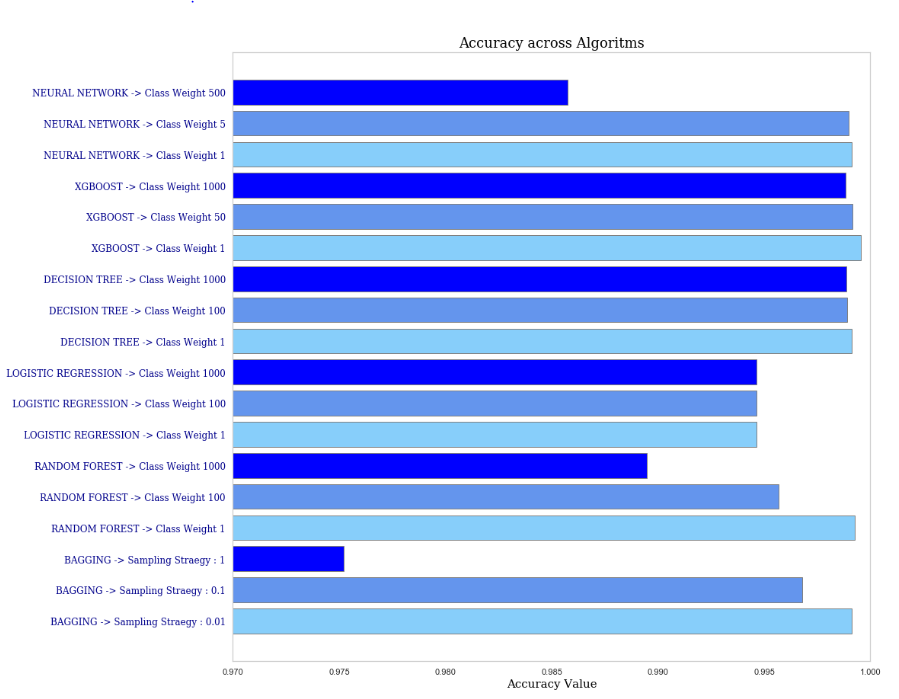
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Class\_Weight | Confusion Matrix | | Recall (Sensitivity) | AUC-ROC | Accuracy |
| {0:1, 1:1} | 85247 | 48 | 0.824324 | 0.962304 | 0.999134 |
| 26 | 122 |
|  |  |  |  |  |  |
| {0:1, 1:5} | 85233 | 62 | 0.831081 | 0.958601 | 0.998982 |
| 25 | 123 |
|  |  |  |  |  |  |
| {0:1, 1:500} | 84097 | 1198 | 0.878378 | 0.972720 | 0.985768 |
| 18 | 130 |

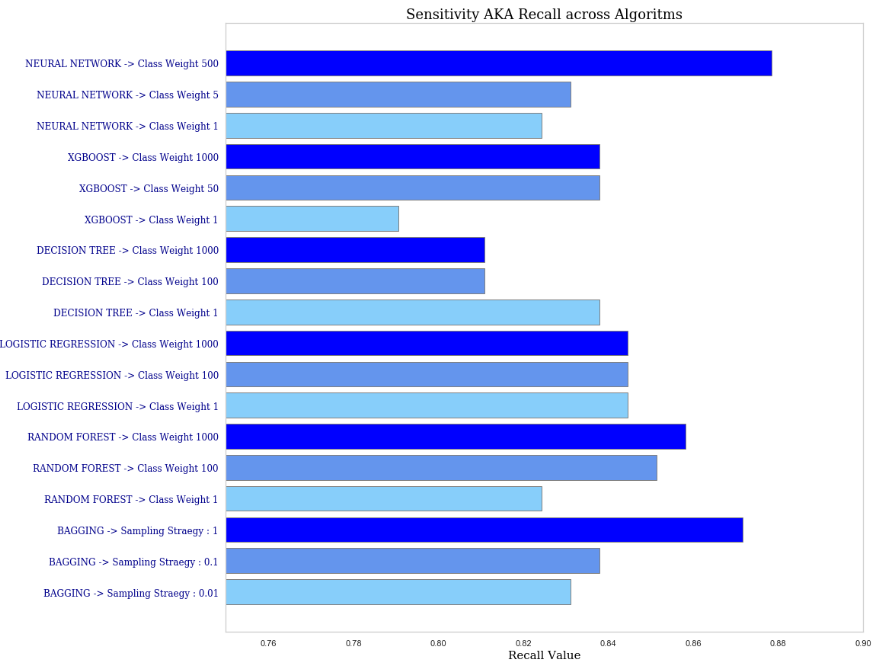
Artificial Neural Networks have given us spectacular results. We have achieved great accuracy as well great Recall & AUC-ROC values. These values have been eclipsed by results from XGBoost Classifier. However, the level of configurability that we were not able to achieve with other algorithms. If we decide that our primary goal is go after the fraud and suppress it, we can achieve it simply by increasing the weight assigned to the fraud class. On the contrary, if we decide for higher accuracy, we can reduce the weight associated with fraud class and ANN model is more predictable oblige. So far, **ANN is the best functional solution** for our dataset.

# Results Summary

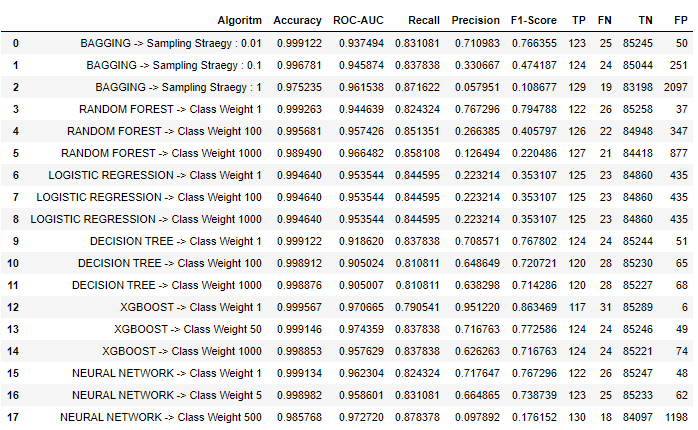
Now that we have run all the algorithms, let us see how they fare in comparison to each other in terms of the key metrics such as Recall, ROC-AUC and Accuracy







Finally, execution of 6 algorithms with 3 different class weight variants is summarized in the table below.

**Results Table**

# Conclusion

The goal of this project was to create a suitable algorithm that can be used to prevent credit card fraud. To come up with a solution, we followed the usual steps required to build a ML solutions viz

* Exploratory Analysis
* Identifying the Challenges
* Data Pre-Processing
* Identifying Suitable Algorithms
* Build Models
* Compare Results

The greatest challenge for this dataset was identified as the heavily skewed dataset with fraud class samples consisting of less 0.2% of the total transactions. This makes it a quintessential binary classification problem for an imbalanced dataset.

During data exploration phase, through visualization we observed that due to the nature of the data, it would be impossible to catch all the fraudulent transactions.

Once the nature of the problem was identified correctly, we took necessary steps bring the dataset in balance using oversampling technique, SVMSMOTE which is a modification of SMOTE [23]. Once that we had sufficient data, we identified appropriate algorithms that can handle the class imbalance through hyperparameter tuning. We used 6 algorithms

* Balanced Bagging Classifier
* Random Forest Classifier with Class Weighing
* Cost Sensitive Logistic Regression
* Cost Sensitive Decision Trees
* Cost Sensitive Gradient Boosting (XGBoost)
* Cost Sensitive Artificial Neural Network

When the algorithms were trained and tested against the dataset with different class weights, we had interesting results. Except for the notably poor performers such as Logistic Regression, all the algorithms had good accuracy. However, the key metric ROC-AUC was best achieved with XGBoost Classifier and Artificial Neural Network. And another metric Recall, which determines the number of fraudulent transactions correctly identified as a percentage of total fraudulent transactions was best achieved by Random Forest Classifier, Bagging Classifier and Artificial Neural Network.

Overall, Artificial Neural Network scored well across all parameters. Also, most notably, ANNs were most responsive and predictably tuned with the hyperparameters. This feature gives the modeler very high flexibility to fine-tune the model as per business requirements. Due to this, **ANN are the best solution for this dataset**.

And, one more thing I would like to mention here is the importance of Data Pre-Processing step. [24] With the oversampling and scaling of data, performance of all algorithms improved markedly. With this, I can emphatically state that Data Pre-Processing and Model Selection are the most important steps.

# References

[1] Leah Hendry (2016). “Bank didn't do enough to prevent online credit card fraud” Retrieved from https://www.cbc.ca/news/canada/montreal/vincenzo-lingordo-credit-card-fraud-bank-1.3900001

[2] Machine Learning Group, ULB (2018). “Credit Card Fraud Detection Anonymized credit card transactions labeled as fraudulent or genuine” Retrieved from https://www.kaggle.com/mlg-ulb/creditcardfraud

[3] Andre Violante (2018) “An Introduction to t-SNE with Python Example” Retrieved from https://towardsdatascience.com/an-introduction-to-t-sne-with-python-example-5a3a293108d

[4] Paula Branco, Luıs Torgo, Rita P. Ribeiro1(2015) "A Survey of Predictive Modelling under Imbalanced Distributions" Retrieved from https://web.cs.dal.ca/~ltorgo/publication/2015\_btr15/

[5] Roberta Pollastro (2020) "How to handle Class Imbalance Problem" Retrieved from https://medium.com/quantyca/how-to-handle-class-imbalance-problem-9ee3062f2499

[6] Gustavo E. A. P. A. Batista, Ronaldo C. Prati , Maria Carolina Monard (2004) "Study Of The Behavior Of Several Methods For Balancing Machine Learning Training" Retrieved from https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.58.7757&rep=rep1&type=pdf

[7] Haibo He, Yunqian Ma *Imbalanced Learning: Foundations, Algorithms, and Applications* Hardcover. New York: Wiley, 2013.

[8] Victor E. Irekponor (2019) "CREATING AN UNBIASED TEST-SET FOR YOUR MODEL USING STRATIFIED SAMPLING TECHNIQUE" Retrieved from https://blog.usejournal.com/creating-an-unbiased-test-set-for-your-model-using-stratified-sampling-technique-672b778022d5

[9] Jason Brownlee (2020) "How to Use StandardScaler and MinMaxScaler Transforms in Python" Retrieved from https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/

[10] Max Kuhn, Knell Johnson *Applied Predictive Modeling*. New York: Springer, 2018.

[11] Nguyen, Hien M., Eric W. Cooper, and Katsuari Kamei (2011). “Borderline over-sampling for imbalanced data classification.” Retrieved from https://pdfs.semanticscholar.org/5c0b/e11c0dfb22a50b59570a06768d0d86188057.pdf

[12] Cheng G. Weng Josiah Poon (2006) "A New Evaluation Measure for Imbalanced Datasets" Retrieved from https://www.researchgate.net/profile/Josiah\_Poon/publication/221338017\_A\_New\_Evaluation\_Measure\_for\_Imbalanced\_Datasets/links/5566437d08aefcb861d198ed/A-New-Evaluation-Measure-for-Imbalanced-Datasets.pdf

[13] Yun Qian, Yanchun Liang, Mu Li, Guoxiang Feng, Xiaohu Shi (2011) "A resampling ensemble algorithm for classification of imbalance problems" Retrieved from http://xuebalib.oss-cn-shanghai.aliyuncs.com/xuebalib.com.37042.pdf

[14] Charles X. Ling, Victor S. Sheng (2008) "Cost-Sensitive Learning and the Class Imbalance Problem" Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.164.4418&rep=rep1&type=pdf

[15] Imblearn (2016) “BalancedBaggingClassifier” Retrieved from https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.ensemble.BalancedBaggingClassifier.html

[16] Leo Breiman (2001) "Random Forests" Retrieved from https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf

[17] Dinesh Yadav (2020) "Weighted Logistic Regression for Imbalanced Dataset" Retrieved from https://towardsdatascience.com/weighted-logistic-regression-for-imbalanced-dataset-9a5cd88e68b

[18] Kai Ming Ting, "An instance-weighting method to induce cost-sensitive trees," in IEEE Transactions on Knowledge and Data Engineering, vol. 14, no. 3, pp. 659-665, May-June 2002, doi:10.1109/TKDE.2002.1000348.

[19] Tianqi Chen, Carlos Guestrin (2016) "XGBoost: A Scalable Tree Boosting System" Retrieved from https://www.kdd.org/kdd2016/papers/files/rfp0697-chenAemb.pdf

[20] XGBoost Docs (N.A.) "Notes on Parameter Tuning" Retrieved from https://xgboost.readthedocs.io/en/latest/tutorials/param\_tuning.html

[21] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng and P. J. Kennedy, "Training deep neural networks on imbalanced data sets," 2016 International Joint Conference on Neural Networks (IJCNN), Vancouver, BC, 2016, pp. 4368-4374, doi: 10.1109/IJCNN.2016.7727770.

[22] Matjaz Kukar, Igor Kononenko (1998) "Cost-Sensitive Learning with Neural Networks" Retrieved from https://pdfs.semanticscholar.org/bdef/7eb9b62e2a12b870957879f7a097b41f6012.pdf?\_ga=2.146525431.250973509.1597029313-870585035.1597029313

[23] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “Smote: synthetic minority over-sampling technique,” JAIR, vol. 16, pp. 321–357, 2002

[24] Ryan Miller (2019) "Data Preprocessing: what is it and why is important" Retrieved from https://ceoworld.biz/2019/12/13/data-preprocessing-what-is-it-and-why-is-important/