Machine Learning Engineer Nanodegree

Credit Card Fraud Detection

Shakti Misra

July 9th, 2019

Table of Contents

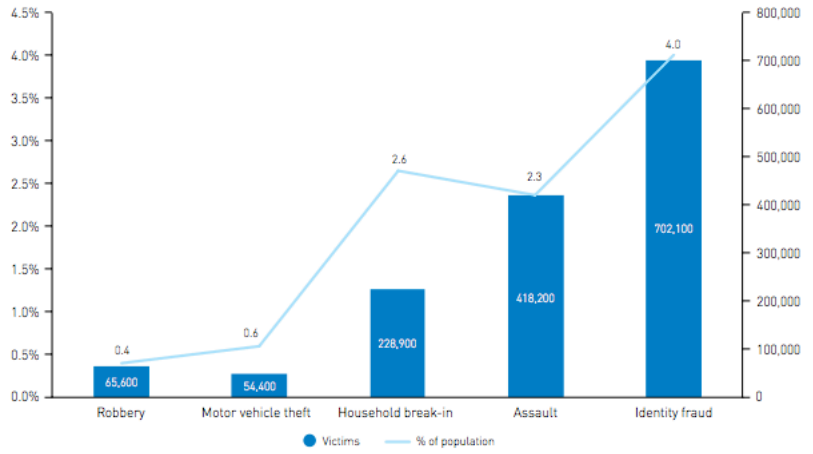
I. Definition

In this section we will discuss the basic overview of the project.

Project Overview

Rise of credit card is on all time high and people use it for various reasons. With this rise in usage the chances of fraud is also increasing. Credit card and identity are now a days available in black market. The image below shows that these kind of identity threat is much high than any other crime now a days.

Figure 33: Number of victims and proportion of population or household, by offence type (n and %)



Sources: ABS 2015, and ABS 2012.

Identity threat (Courtesy wikipedia*)*

Working in expense management industry it is quite important to help out customers with solutions where these kind of transactions can be identified and flagged. Considering the credit card billing is integrated with our solution and is shown up in our portals it will be a useful feature. Even it will be useful when the credit card is insured. In which case the amount can be claimed from the insurance company if possible.

Problem Statement

Fraud is defined as the use of one's asset for personal use through misbehaviour. Fraud is increasing drastically with globalization and modern technology which results in major loss to regular user and larger organizations. Fraud detection refers to the act of identifying frauds as early as possible.

In this project we will be using historical credit card data for recognizing fraudulent credit card transaction from the transaction data so that the customer can be alerted and/or protected from the fraud.

With the rise in credit card usage and transaction it will be impossible for us to do this activity manually. Hence we will be taking help of machine learning to help us in identifying credit card fraudulent transactions.

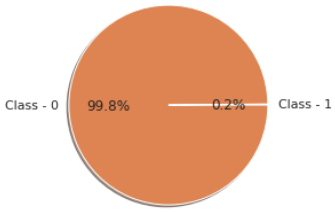
While doing this activity our goal is not only to device a proper algorithm but also to see that the efficiency of the algorithm is in a desirable range. In real world application this is very important too. Out of millions of transaction records the algorithm should be able to find out suspected fraudulent records and flag them. These records generally then go to auditors who can actually verify if the record is a real fraudulent record or not. Apart from that these systems can be used to send alert to actual users. This is when we get mails of fraudulent transactions or get calls from credit card representatives from banks to confirm if the transaction is fraudulent or a valid transaction.

Our goal should be following:

- Classify all of the fraudulent cases as fraud
- Classify most of the non-fraud transaction as non-fraud
- It is okay to classify few of the non-fraud cases as fraud, but it should be very few. This means it's okay to classify all of the fraudulent cases + few non fraud cases.
- The algorithm should be efficient

Metrics

In Machine Learning, performance measurement is an essential task. To come up with the metrics we have to first see the distribution of the data at hand on which we need to make the prediction. The fraud cases in general are very less as shown in the diagram



The data is highly imbalanced so accuracy is not a good measure for this problem as the cost of **False negative** is far sever than that cost of **False positive**. We cannot even use accuracy for this case as the data is highly skewed. So, if we predict all values as valid still the accuracy will be quite high. Which is not a desirable trait of evaluation. Confusion matrix will not provide a good measure of the evaluation. The total number of fraud cases is much less (~0.2%); and variation in the confusion matrix will not be a good measure.

In scenarios like this two diagnostics tools are more helpful in coming to a conclusion.

- **Area Under the Curve *(AUC)* Receiver Operating Characteristics *(ROC)*** A ROC curve is plotting True Positive Rate (TPR) against False Positive Rate (FPR). The ROC area under curve (ROC AUC) is just the area under the ROC curve. AUC - ROC curve is a performance measurement for classification problem like the one we are doing here. ROC is a probability curve and AUC represents degree of separability. It tells how much model is capable of distinguishing between classes. Higher value means the model is a good model. So in this case high value of the curve will tell that our model is better at finding the fraud to non-fraud separation.

$$TruePositiveRate(TPR) = \frac{TP}{TP + FN}$$

$$FalsePositiveRate(FPR) = \frac{FP}{FP + TN}$$

- **Precision Recall Curve *(PR)* Area Under the curve *(AUC)*** A PR curve is plotting Precision against Recall. The precision recall area under curve (PR AUC) is just the area under the PR curve. Higher value means the model is better. The PR plot is a model-wide measure for evaluating binary classifiers and closely related to the ROC plot.

$$Prescision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

The difference between the ROC-AUC and PR-AUC is how each treats true negatives (TN). If true negatives are not that meaningful to the problem at hand or they are very low in number PR-AUC is going to be more useful. On the other hand ROC-AUC is going to be more useful. For our problem at hand we will be giving more stress on the PR-AUC but we will be plotting both the curves and see how they vary. For our problem at hand both Precision and Recall at hand are useful as we are having a class imbalance problem where only few of the Class 1 are present and lot more Class 0 are present. The reason we will take into account **ROC-AUC** is that we will be employing some methods to balance the dataset. Once we do that the dataset will be balanced and hence AUC-ROC can be applied. So it will be interesting to see how both of the curves look in this case. AUC-ROC will be the primary method on which we will be measuring the performance of our algorithm.

II. Analysis

In this section we will be analysing how we want to proceed in this project, what kind of insight we are getting from the data and how we can refine it. We will also see what kind of algorithm we will be using on the data. This section details more things bit in theory. Next sections will be more result oriented. Most of the theory will be covered in this section.

Data Exploration

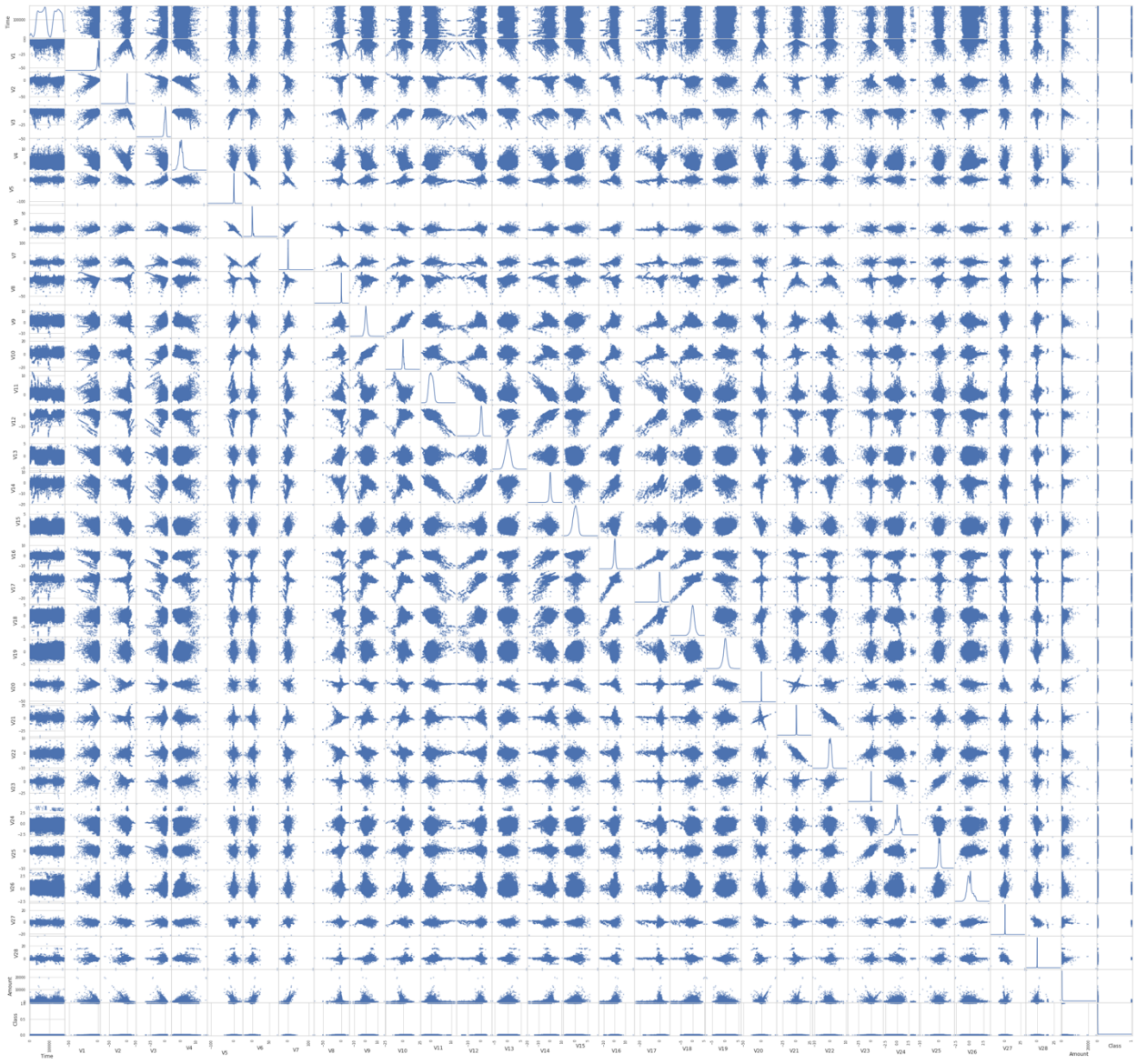
Data exploration is the primary step to get a basic understanding of the data. This step helps in gaining some basic knowledge about the data so that we can pick and choose various method to refine the data and apply machine learning methods.

General overview

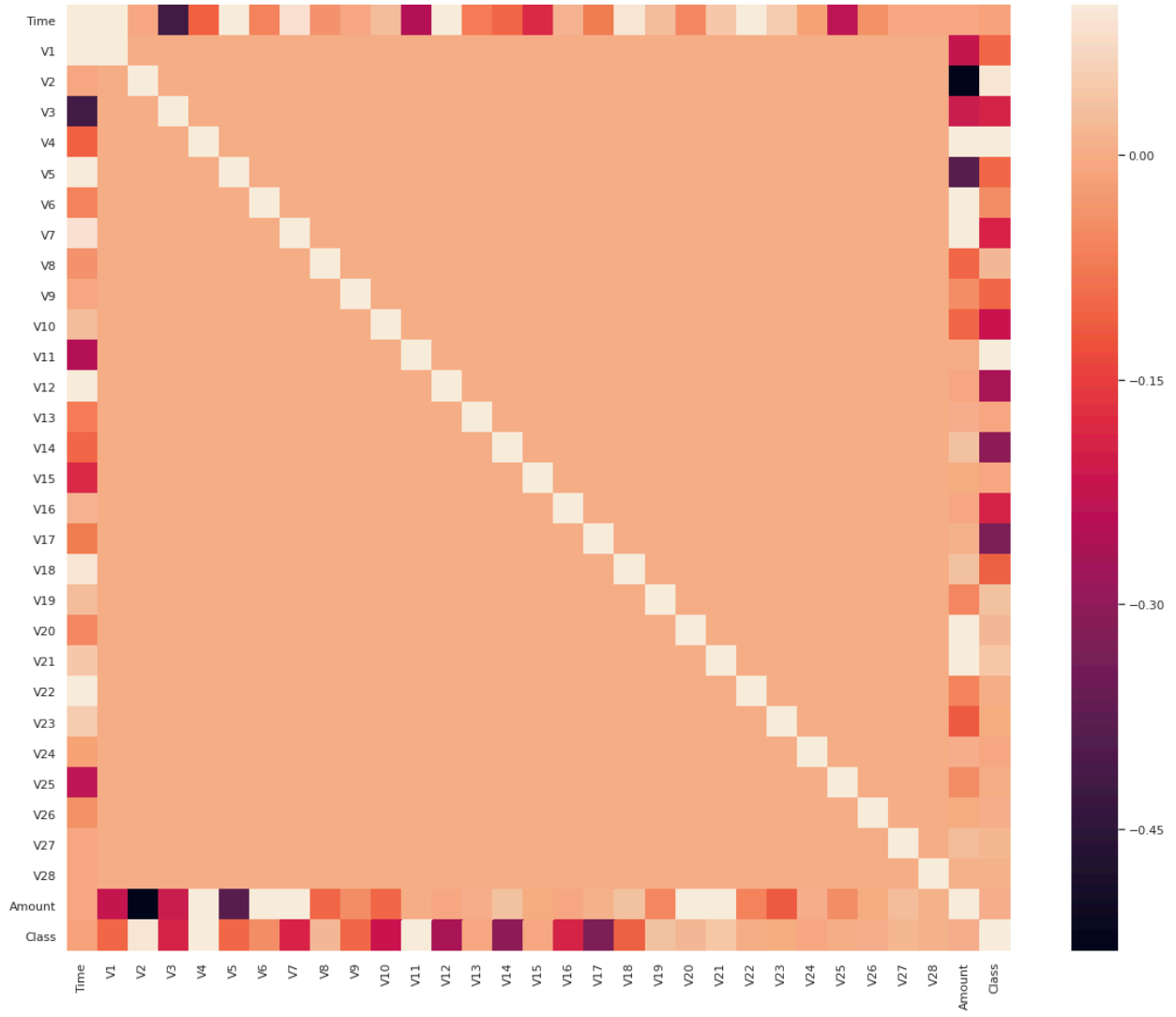
The datasets contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 fraud/284807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions. It contains only numerical input variables which are the result of a PCA transformation. 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'**. The Class field of the data tells if the transaction is actually a fraudulent (Class = 1) or regular transaction (Class = 0). One good thing about this data is that there are no null values. So we can use all of the data set for our training and analysis.

Further analysis

Considering the variables are not something I could make much sense of plotting the correlation matrix was of not much use as we see below:



Only few variables seems to vary with others, but I will not say that this gives out some good info just visually plotting them.

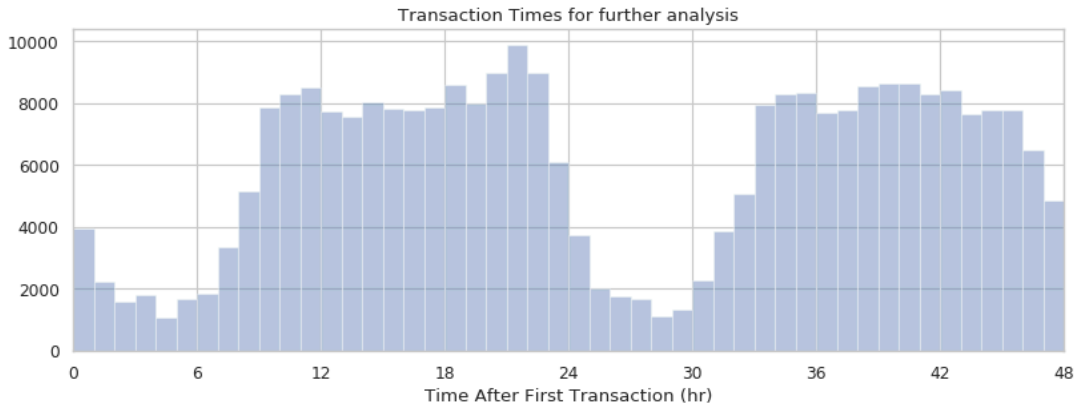


The correlation matrix heat map gives some idea about the relation of the data.

- The values from $V1-V28$ do not show a great deal of correlation. Most of the values linger around zero
- The Class field is showing some correlation with each of the fields. Same is true for the Amount field. But considering we need to predict Class and it shows some correlation it looks promising to explore
- Further we can see that V12, V14, V17 have a strong negative correlation and then V11, V2 seems to have some strong positive correlation
- *Time* has strong correlation with V11, V15, V25. Not really sure if time has any real relation to the frauds committed

Variation of Time

Further investigation shows that the transaction has been done over a period of 2 days. For that we have to convert the Time into hours. Below graph shows that:

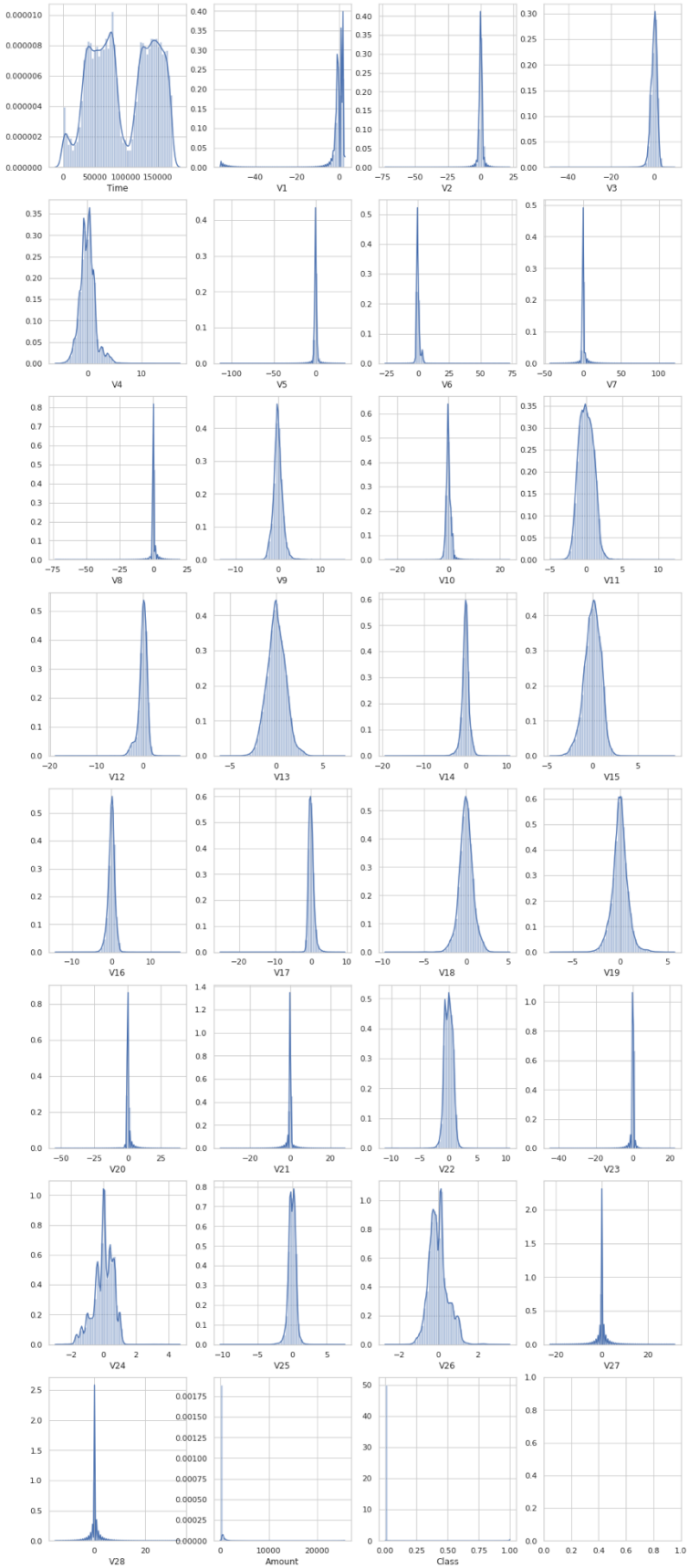


From here we can see that the transaction happens at some peak time over 2 days period. I guess this is the day time for the place where transaction was captured. The transactions reach a peak each day and then they reduce.

Distribution of data

The distribution of data is given below:

- From below graph we can see that most of the data is *normally distributed*
- Some data points are bit skewed, so we may have to modify them so the distribution is proper. This will help our learning algorithm perform better


Time
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16
V17
V18
V19
V20
V21
V22
V23
V24
V25
V26
V27
V28
Amount
Class

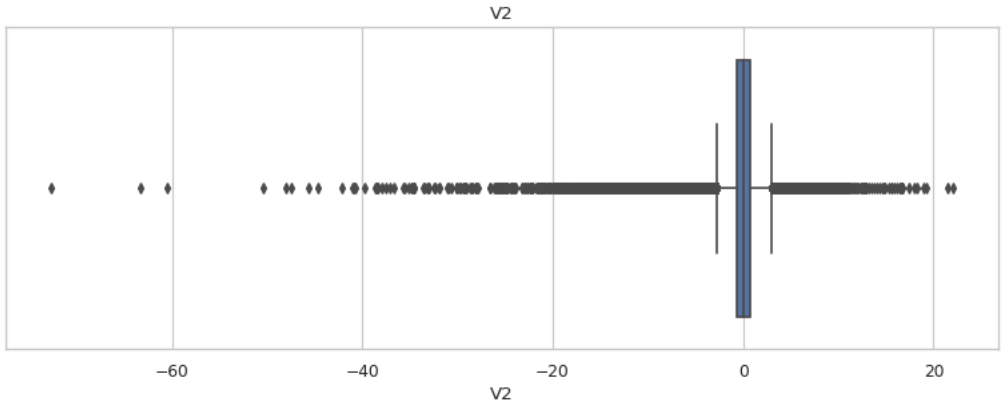
Exploratory Visualization

In this section we will provide some more insight into the data we have to model.

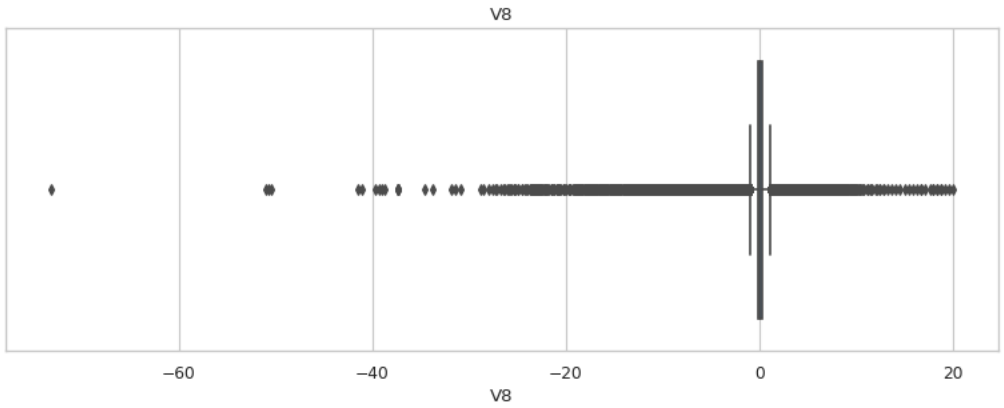
One very important aspect of the data that at hand is that the fraudulent transaction are view less. This is obvious from the data distribution also.

As we discussed in the previous section some of the data features are also bit skewed. Few places I have read that this kind of skewness may not impact the performance quite a lot but just to be in the safer side we will rectify it. To view some of the data points that is skewed we can see the box plots:

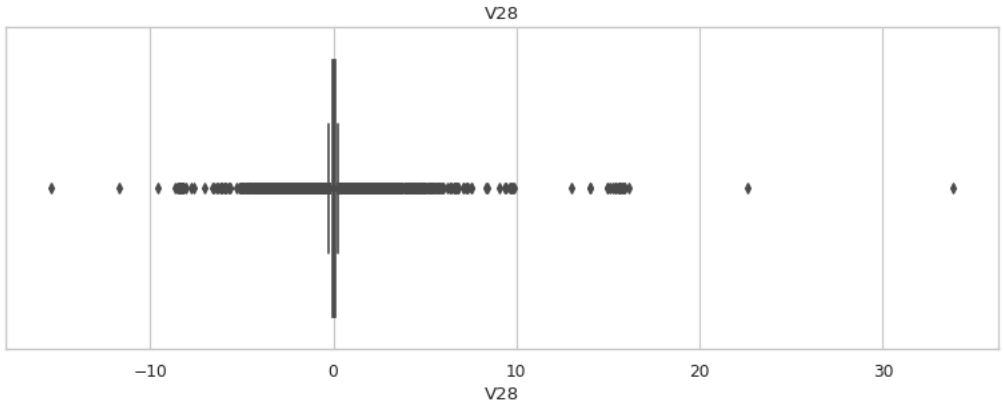
```
plot_box(data = data, var = 'V2')
```



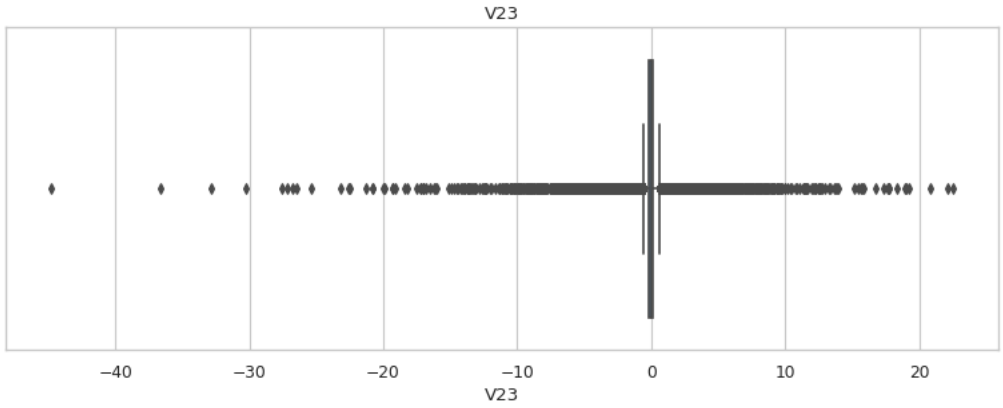

```
plot_box(data = data, var = 'V8')
```



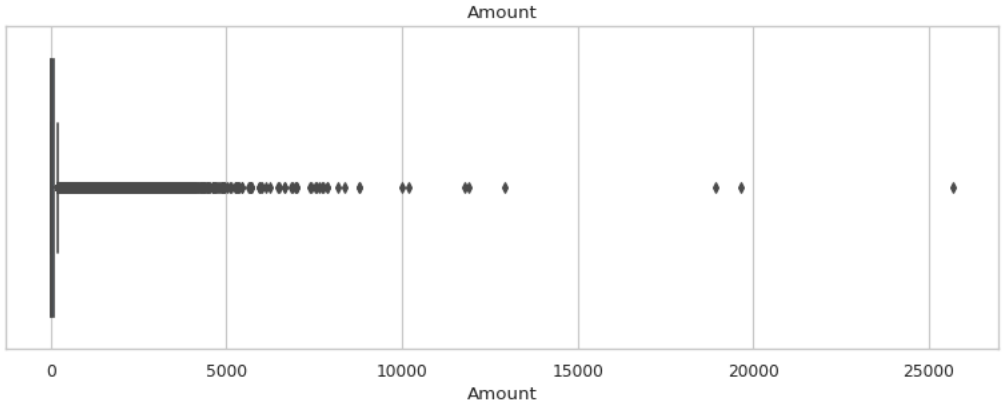
```
plot_box(data = data, var = 'V28')
```




```
plot_box(data = data, var = 'V23')
```




```
plot_box(data = data, var = 'Amount')
```

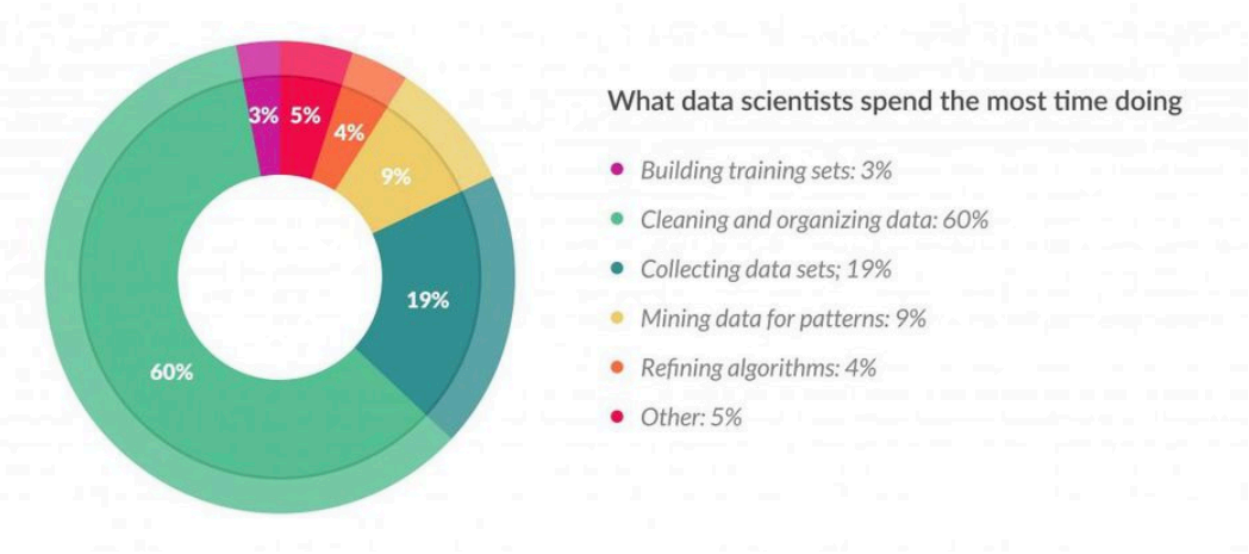


The above box plots show us that these variables are having outliers. So these are some of the values we need to correct before we apply any machine learning.

Algorithms and Techniques

The aim of the project is to automatically detect fraudulent transaction with best possible measures. The primary method to establish a base model is Logistic Regression but before trying that I wanted to quickly play around with some other algorithms to see how they perform after I did some massaging of the data. I saw that these algorithms are quick to perform so I get a fair idea on how these methods work. Some links to the algorithms and their significance is given in the reference section.

Just to give a glimpse here is the time spent in various phases of the data analysis and prediction:



Data cleansing and Scaling the data

As we saw in the previous section some of the data fields are bit skewed. So we will have to polish the data fields a bit. We will be evaluating and employing the following methods to normalize the data.

- **MinMaxScaler:** The MinMaxScaler is the probably the most famous scaling algorithm, and follows the following formula for each feature
- **MaxAbsScaler:** Scale each feature by its maximum absolute value
- **StandardScaler:** The StandardScaler assumes your data is normally distributed within each feature and will scale them such that the distribution is now centred around 0, with a standard deviation of 1
- **RobustScaler:** The RobustScaler uses a similar method to the Min-Max scaler but it instead uses the interquartile range, rather than the min-max, so that it is robust to outliers.
- **Normalizer:** The normalizer scales each value by dividing each value by its magnitude in n-dimensional space for n number of features

- **QuantileTransformer:** QuantileTransformer applies a non-linear transformation such that the probability density function of each feature will be mapped to a uniform distribution. In this case, all the data will be mapped in the range [0, 1], even the outliers which cannot be distinguished anymore from the inliers
- **PowerTransformer:** PowerTransformer applies a power transformation to each feature to make the data more Gaussian-like

Use anomaly detection to detect fraud

Small amount fraud transaction data are like anomaly. So below methods are used to see if we can effectively detect fraud. This is just to establish a crude baseline even before we can normalize the data and apply methods like Logistic Regression to establish a solid baseline.

- **IsolationForest:** The IsolationForest isolates observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature
- **LocalOutlierFactor:** this is an unsupervised Outlier Detection using Local Outlier Factor. It measures the local deviation of density of a given sample with respect to its neighbours. It's closely related to k-nearest neighbours where distance is used to estimate the local density

Resampling

The data that we have is very skewed in terms of the Class variable that we need to predict. This is a demerit when we are training the model. A dataset is imbalanced if the classification categories are not approximately equally represented. In this case the data sets are predominately composed of "non fraud" sets with only a small percentage of "abnormal" or "fraud examples. To overcome this we can do the following:

- Collect more data, in this case collecting more data is not an option. This is not a feasible strategy all the time
- Change the performance matrix by calculating confusing matrix, getting the precision, recall and F1 score
- Modify loss function
- Resampling the dataset
- Ensemble methods We will be sticking with the resampling methods. We will go through few of them briefly and them choose the best one to pick

In resampling the dataset we will need to pre-process the dataset in certain ways so that we can help the learning algorithm increase the efficiency.

Few ways to do it is as follows:

- **Undersampling:** This method balances the dataset by reducing the size of the majority class. This method is used when quantity of data is sufficient. By keeping all samples in the minority class and randomly selecting an equal number of samples in the majority class, a balanced new dataset can be retrieved for further processing. In this method we are effectively reducing the size of the majority class. So data loss happens. Here are few ways to do it.
    - Random under sampling
        - Randomly remove 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
    - ClusterCentroids
        - Cluster centroids is a method that replaces cluster of samples by the cluster centroid of a K-means algorithm, where the number of clusters is set by the level of under sampling
    - NearMiss
        - NearMiss selects the majority class samples which are close to some minority class samples. In this method, majority class samples are selected while their average distances to three closest minority class samples are the smallest
- **Oversampling:** This is used when the quantity of data is insufficient. It tries to balance dataset by increasing the size of minority samples. Here we have to generate some synthetic samples in some way or the other. Below are few ways of achieving this
    - 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
    - SMOTE (Synthetic minority oversampling technique)
        - SMOTE synthesizes new minority instances between existing (real) minority instances
- **Combine:** We can also combine the above methods to achieve results
    - SMOTE + ENN and SMOTE + Tomek

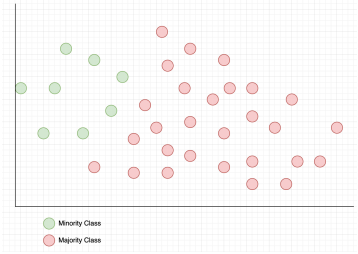
To baseline I will implement a hardcoded under sampling method and also measure the performance of that method. Then after evaluating other methods I will be choosing the best for the job at hand. Considering I am using the hand coded baseline method and SMOTE variants in my processing I will briefly explain them in details here.

Hand coded Under sampling

Check all the data points that have Class as 1 (fraud). Let the number of Class 1 points be x, Now randomly select the same x number of points from all the points where Class is 0 (Non fraud). Merge these two set of data points. So the total number of data points will be 2x. The problem we have here is we will be discarding all the other data points from Class 0. This is a major loss. These points may carry some useful feature vectors. This is just a base line method.

SMOTE

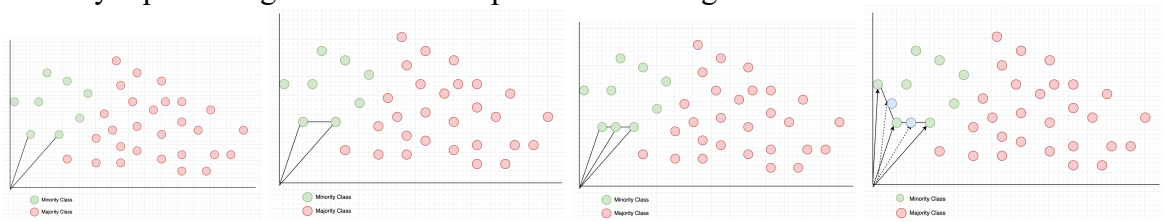
To learn more about the SMOTE variant we need to understand SMOTE first. For that we will consider a simple example as in below diagram.



The way SMOTE works is it tries to synthesize new minority class members. In this case new Green points. SMOTE synthesises new minority instances between existing (real) minority instances, rather than randomly generate them. The process of SMOTE is as follows:

- Identify the feature vector and its nearest neighbour
- Take the difference of the two
- Multiply the difference with a random number between 0 and 1
- Identify a new segment on the line joining the above two points by adding the random number to the feature vector
- Repeat the above for all the identified feature vectors

Visually representing all the above steps from left to right:

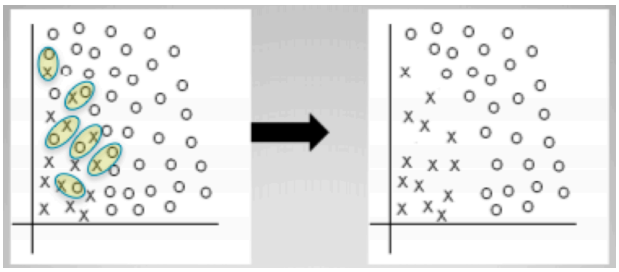


Here the Blue dots are synthesized. This is the basic SMOTE algorithm. In my evaluation I have evaluated another 2 variations of SMOTE. I will briefly outline the other 2 algorithms here too. SMOTE is a over sampling method. The number of neighbours choosen are dependent on the percentage replication needed. For example in above example I am choosing 2 neighbours as I need 200% replication. It is important to note a substantial limitation of SMOTE. Because it

operates by interpolating between rare examples, it can only generate examples within the body of available examples never outside.

SMOTE + Tomek

To understand this we need to understand Tomek links. Tomek links are pairs of instances of opposite classes who are their own nearest neighbours. In other words, they are pairs of opposing instances that are very close together. In this case either is considered a noise of the other one. Tomek's algorithm looks for such pairs and removes the majority instance of the pair. This is essentially a under-sampling technique. It's also used as a data cleansing technique to clean the border between the majority and minority classes.



As we have discussed previously for SMOTE the new data points are synthesized in inside the boundary or convex hull. This can lead to the problem of overfitting the data points. So we can apply SMOTE along with Tomek, taking strong points of both the algorithms. Here is the procedure:

1. Over sample the dataset with SMOTE
2. Identify Tomek links
3. Cleanse the data using Tomek algorithm

SMOTE + ENN

Wilson's Edited Nearest Neighbour Rule (ENN) is a type of Neighbourhood Cleaning Rule to remove majority class examples. ENN removes any example whose class label differs from the class of at least two of its three nearest neighbours. It's kind of similar to the previous method but in a very high level and also it's a under classification method. It works in the following few steps:

1. For each data point *d* in the training set, its three nearest neighbours are found.
2. If *d* belongs to the *majority* class and the classification given by its three nearest neighbours contradicts the original class of *d*, then *d* is removed.
3. If *d* belongs to the *minority* class and its three nearest neighbours misclassify *d*, then the nearest neighbours that belong to the *majority* class are removed.

ENN tends to remove more data points than the Tomek links does, so it is expected that it will provide more in-depth data cleaning. This method also can be used with the SMOTE method in the same 3 steps:

1. Over sample the dataset with SMOTE
2. Identify Tomek links
3. Cleanse the data using ENN algorithm

Baseline with Logistic Regression

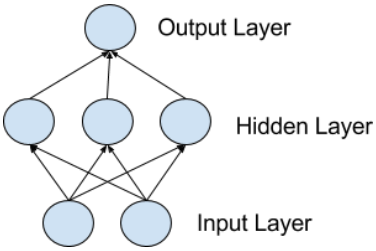
Logistic Regression is a machine learning method for classifying or categorize values. It is one of the most simple and commonly used machine learning algorithms for two-class classification. It is easy to implement and can be used as the baseline for any binary classification problem. Logistic regression describes and estimates the relationship between one dependent binary variable and independent variables. It is a special case of linear regression where the target variable is categorical in nature. Logistic regression uses the logit function or the sigmoid function. Here are the benefits of logistic regression:

- This is simple and efficient
- Quick to execute that the neural network
- Low variance
- It provides probability for the observation

This problem at hand is a classification problem. Apart from this Logistic Regression is made for these kind of problems of categorization. So it only makes sense that we compare a neural network based method with logistic regression.

Multilayer perception

A multilayer perceptron (MLP) is a class of feedforward artificial neural network. A MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer.



MLP can be used for following type of problems:

- Tabular datasets
- Classification prediction problems
- Regression prediction problems

Considering this is a classification prediction problem I guess this will be good enough for our cause. They are very flexible and can be used generally to learn a mapping from inputs to outputs.

This flexibility allows them to be applied to other types of data. Certain times MLP can be used as a baseline before we move to other kind of neural networks.

For our problem at hand we will use the following configuration:

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 16)                496
_________________________________________________________________
dense_17 (Dense)             (None, 18)                306
_________________________________________________________________
dropout_4 (Dropout)          (None, 18)                0
_________________________________________________________________
dense_18 (Dense)             (None, 20)                380
_________________________________________________________________
dense_19 (Dense)             (None, 24)                504
_________________________________________________________________
dense_20 (Dense)             (None, 1)                 25
=================================================================
Total params: 1,711
Trainable params: 1,711
Non-trainable params: 0
_________________________________________________________________
```

Benchmark

In this case we need to detect fraud. So our goal will be to catch all the fraud cases. If we classify any non-fraud case as fraud too that's fine, cause mostly these kind of reports and flags gets verified by expert auditors after they are flagged. So false +ve also is fine in small quantities, but we should not leave any true +ve cases. It is also the case that the cost of misclassifying an abnormal (fraud) example as a non-fraud case is often much higher than the cost of the reverse error. So in other words

- High **recall** on **fraud data (Class = 1)** (Recall/Sensitivity: how many relevant instances are selected)
- High **precision** on **non-fraud data (Class = 0)** (Precision/Specificity: how many selected instances are relevant)

Predictive accuracy is the performance measure generally associated with machine learning algorithms and is defined as Accuracy = (TP+TN)/(TP+FP+TN+FN). When the dataset is balanced has equal error costs, it is reasonable to use the error rate (1-Accuracy) as a performance metric.

This dataset is imbalanced and then we balance it using some resampling methods. As discussed earlier we will be using the ROC-AUC curve as the primary measure of performance. This ROC-AUC curve will be drawn for the Logistic Regression first. To obtain the best ROC-AUC curve we will be doing hyperparameters tuning using the Gridsearch method. This will give us a solid base to start with, as Logistic Regression is made for this kind of sampling problem. After that our job will be to improve on this result. As we discussed earlier the higher the value of the AUC the better the prediction is. Then we will train a neural network and plot the AUR-ROC curve for that too. I will be plotting the PR-ROC just alongside to see how much the results differ.

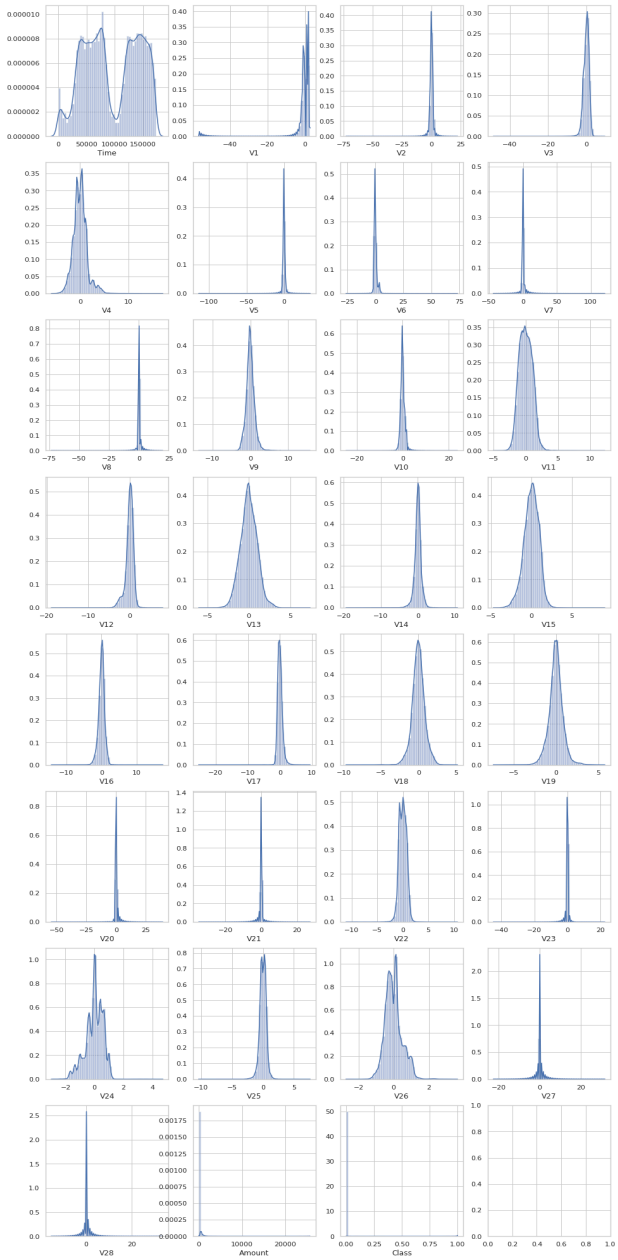
III. Methodology

This section details the methods we have used to come to the conclusion.

Data Pre-processing

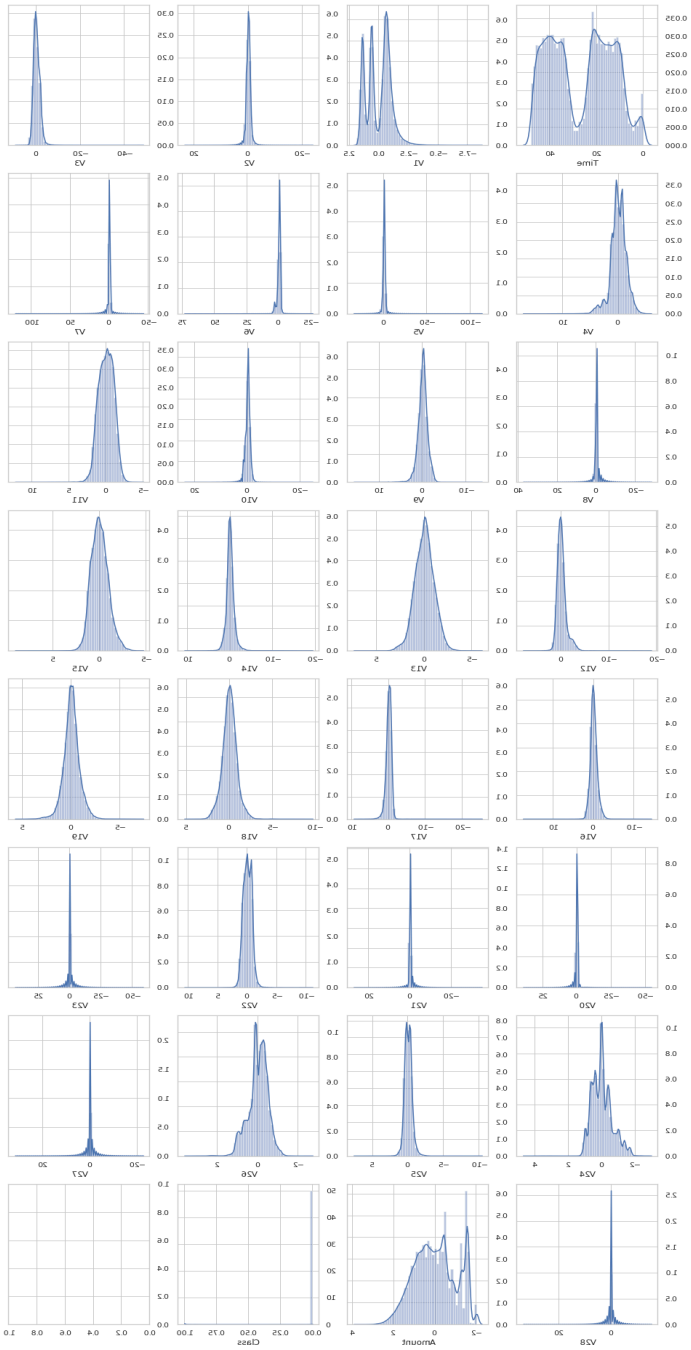
Data normalization

To normalize the data I had to evaluate a few methods as explained above. Data before normalization:



From that graph and the box plots given in the previous sections I had to test few of the normalization on each of them and then plot the distribution plot again. Amount, V1, V2, V8, V23 were normalized. Then I converted the Time field to hours.

After normalization the following distribution plot was generated:



Data resampling

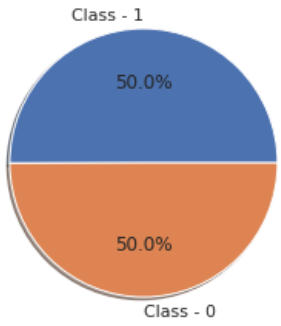
The data is highly imbalanced as we have been noticing for some time. To make the data balanced we have to try out few resampling methods and pick up the best method. To come up with the metrics need apply each of the to the data and test them with a machine learning method and get the metrics. The easiest way that I could do was to use the same Logistic Regression method. The hyper parameters tuning was done using Gridsearch and then the resampled data was used to come up with the below table.

Sampling Type	F1	Precision	Recall	Accuracy	AUC - ROC	AUC - PR	Confusion matrix
base	0.694836	0.860465	0.582677	0.999087	0.791254	0.744027	[[71063, 12], [53, 74]]
Naive Handcoded	0.140625	0.076122	0.921260	0.979916	0.950640	0.655864	[[69655, 1420], [10, 117]]
SMOTE	0.114425	0.061001	0.921260	0.974565	0.947960	0.738917	[[69277, 1798], [10, 117]]
RandomOverSampler	0.126350	0.067826	0.921260	0.977276	0.949318	0.736164	[[69467, 1608], [10, 117]]
NearMiss	0.008704	0.004372	0.960630	0.609702	0.784852	0.417425	[[43290, 27785], [5, 122]]
RandomUnderSampler	0.128751	0.069168	0.929134	0.977571	0.953396	0.683202	[[69487, 1588], [9, 118]]
SMOTEENN	0.111695	0.059451	0.921260	0.973863	0.951475	0.738466	[[69224, 1851], [10, 117]]

Sampling Type	F1	Precision	Recall	Accuracy	AUC - ROC	AUC - PR	Confusion matrix
SMOTETomek	0.114481	0.061033	0.921260	0.974579	0.947960	0.738943	[[69275, 1800], [10, 117]]
ClusterCentroids	0.137413	0.074143	0.937008	0.979017	0.958050	0.726611	[[69589, 1486], [8, 119]]

In this un-balanced data just looking at the F1 score will not be fair. We have to look at AUC - ROC (Area Under the Receiver Operating Characteristics) and AUC - PR (Precision Recall AUC). In case of imbalanced data AUC-PR is important but as mentioned earlier we are also using oversampling method to synthesize new data so our focus metrics will be AUS-ROC. So considering both the values, **SMOTEENN** performs better. Another reason for choosing the SMOTEENN method is due to the fact that it cleans up the cluster boundaries better than SMOTETomek method.

After applying SMOTEENN below is the distribution pie chart. Here we can see that the fraudulent class has also increased in sample size due to the application of the SMOTEEN method. Now our data set is evenly distributed.



Another thing that I had to check after resampling was if the distribution of the data was too much skewed with outliers. I again plotted the distribution graph and check.

Implementation

After picking up the method of choice for resampling the next step was to establish a baseline. After establishing the baseline the next step is to choose a better method and evaluate it.

Baseline method implementation (Logistic Regression)

The problem at hand is a binary problem. We have to choose if a particular transaction is fraudulent or a regular transaction. For these kind of classification problems the Logistic Regression is a good standard as we have discussed earlier. We will go with the Logistic Regression to establish the base methods. Then we can try with neural network based methods. With logistic regression I am using Gridsearch for hyper parameters tuning. The final scores were calculated against the resampled test data and also the original test data.

LogisticRegression is used from sklearn library. Following parameters were used :

- l2 penalty was used as 'newton-cg', 'sag' and 'lbfgs' solvers support only l2 penalties. In general the l2 loss function is a stable function. L2-norm loss function is also known as least squares error. It is basically minimizing the sum of the square of the differences between the target value and the estimated values.
- Lbfgs solver was used. Lgfgs is a limited memory algorithm that can be used in limited memory system as we are using (Collab is a shared resource system). Apart from that this is a popular method in the machine learning forums. I tried using other solvers, but this solver gave me the answer in the quickest time.

To search for the optimal hyper parameter I had to do a gridsearch. The algorithm used from sklearn is GridsearchCV. This does an exhaustive search over specified parameter values for an estimator. The estimator passed in to the algorithm is logisticregression. The parameter which I am trying to estimate here is the *C* parameter from the logistic regression. I am keeping the solver and the penalty constant as given above. The *C* parameter is the inverse of regularization strength and the values it can take is positive float.

To understand the significance of *C* and why we choose it, we need to understand the *regularization*. Here is what it means:

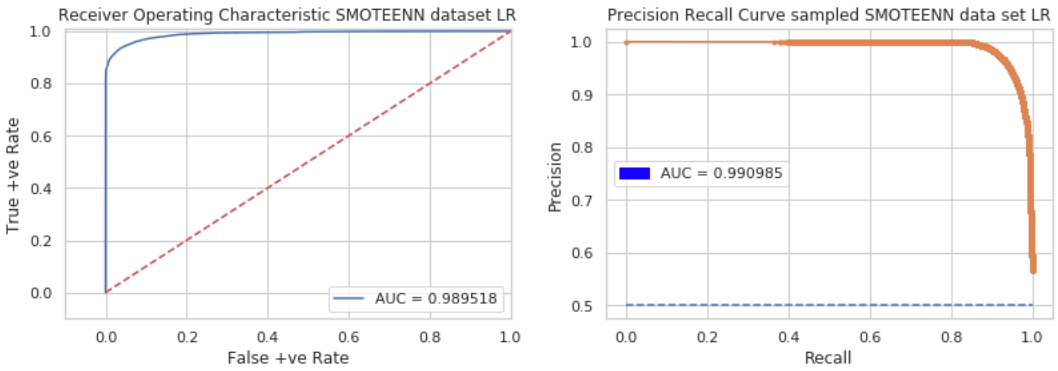
Regularization is applying a penalty to increasing the magnitude of parameter values in order to reduce overfitting. When we train a model such as a logistic regression model, we are choosing parameters that give best fit for the data at hand. When we have lot of independent variable and less data there is a chance to overfit, which is not a desirable trait. To solve this, we also minimize

a function that penalizes large values of the parameters. In many cases the minimization function is $\lambda\sum\theta^2$ here C = 1/ λ.

With a TPU runtime the time taken was almost 750 seconds (12.5 mins). We have to be very careful when we are doing the gridsearch for hyper parameter tuning. On my laptop it did not yield any result even after 2 hours. Even in GPU and TPU it takes significant amount of time. So we have to consider that factor when we are doing this operations. This is the reason I did not go with any other hyperparameters.

As discussed previously we will be using AUC-ROC for determining the best model. Here are the graphs for the baseline method. I will be plotting PR-ROC just for reference too and see if this also improves.

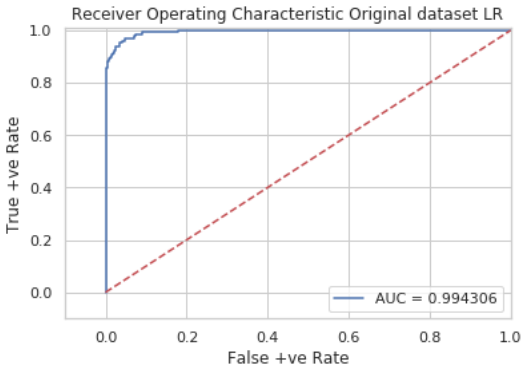
ROC-AUC Curve and PR-AUC Curve

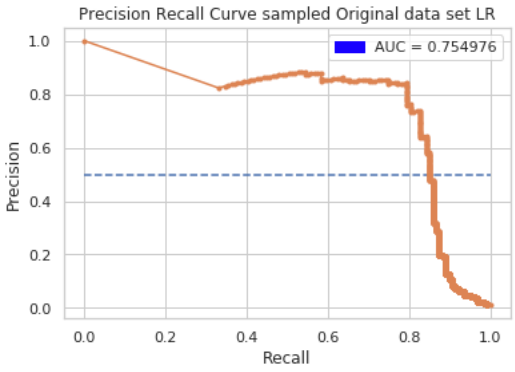


These are the curves after SMOTEENN were applied on the data. As we see the baseline accuracy is quite good. As in the previous section I mentioned the bigger the AUC value the better the results are. Here
ROC−AUC=98.95% and
PR-AUC = 99.09%

Considering we have trained the model with resampled data. I want to check the performance with the original dataset. Here is the graphs:

ROC-AUC Curve and PR-AUC Curve





ROC−AUC=99.43% and
PR-AUC = 75.49

Here in the PR-AUC we see that the precision falls of significantly at 80% recall value which is quite less than that of the SMOTEENN test data where the precision falls nearly at 99% I believe or more.

Neural Network

Now that we have established a base method the next step is to improve on this. We will be employing multilayer neural network to do the job. As mentioned earlier the following neural network was used here with 100 epochs. For the neural network I will be using the Keras library. The backend of the library will be TensorFlow. Using this backend significantly reduces the execution time if the model is executed in a CUDA GPU or TPU.

Models in Keras are defined as a sequence of layers, which is a linear stack of layers. A simple way to work with this is add layers one at a time till we get the desired results. The first thing to get right is to ensure the input layer has the right number of input features which can be specified using the **input_dim** parameter. We have set it to 30 in accordance with the number of parameters we have.

Neural networks are bit tricky. After reading few articles about them what I found is there is a lot of trial and errors we need to do. In general we need to have a network the right size to capture the size of the problem. Another thing I have found helpful is to take neural network from similar problems and modify them to get them working. Regardless this is a time consuming task. Generally I keep the initial **epoch** as low as possible kind of 5-10. This helps me find any kind of runtime errors and get a baseline run. In Collab this is the reason I am using a form to adjust it.

The *Dense* class represents a fully connected layer. The *units* specify the number of nodes and the activation function is specified using *activation*. We will be using the *ReLU* (rectified linear unit) activation function on all the initial layers and *sigmoid* on the output layer. In simple terms the

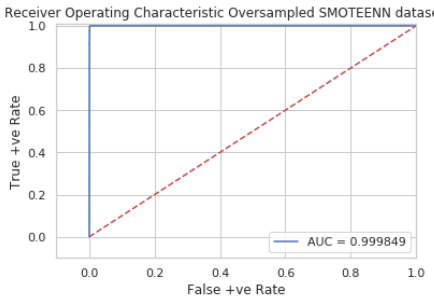
rectified linear activation function (ReLU) is a piecewise linear function that will output the input directly if is positive, otherwise, it will output zero. The reason being using ReLU generates better performance in all layers and it's easy to train. But in the output we need to have a sigmoid to keep the output bound between 0-1. This makes it easy to map it to a probability class.

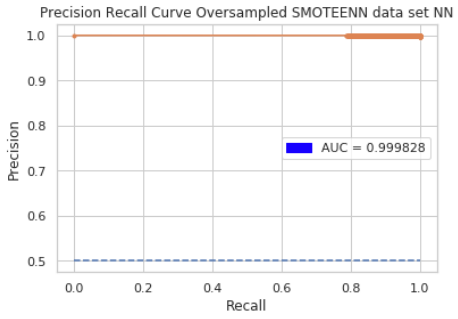
To avoid overfitting we use the Dropout layer. Ensembles of neural networks with different model configurations are known to reduce overfitting. A model can have a large number of different network architectures by randomly dropping out nodes during training. This is called dropout and this is a very computationally cheap and effective regularization method to avoid overfitting. One thing to note is that using dropout means we need more *epochs* for the neural network to learn.

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 16)                496
_________________________________________________________________
dense_17 (Dense)             (None, 18)                306
_________________________________________________________________
dropout_4 (Dropout)          (None, 18)                0
_________________________________________________________________
dense_18 (Dense)             (None, 20)                380
_________________________________________________________________
dense_19 (Dense)             (None, 24)                504
_________________________________________________________________
dense_20 (Dense)             (None, 1)                 25
=================================================================
Total params: 1,711
Trainable params: 1,711
Non-trainable params: 0
_________________________________________________________________
```

Below is the result with the resampled test data:

ROC-AUC and PR-AUC





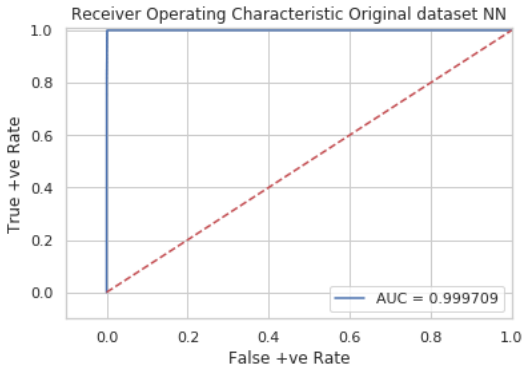
Here the

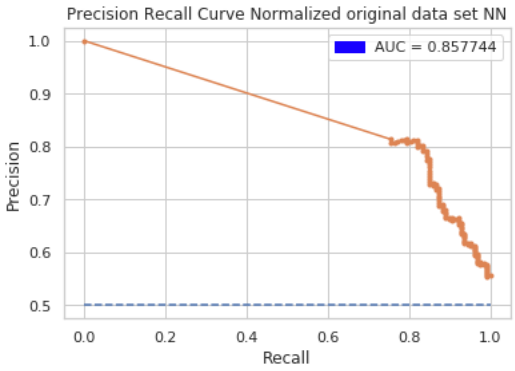
ROC-AUC = 99.97% and

PR-AUC = 99.97%

Even if we test the results with original test data it shows better results:

ROC-AUC and PR-AUC





Here the ROC-AUC = 99.96% and PR-AUC = 85.85%

Refinement

Various refinements were done over the period of the period. A lot of parameters were chosen and kept so as to reduce the running time of the algorithm. These libraries do not perform well in any GPU other than NVidia which supports CUDA. So the running time for any of them is quite a lot. So after spending quite some time trying to run them efficiently in my MAC machine and another desktop with AMD GPU I had to use google Collab. Although Collab is a great place, the sessions don't stay alive long, I guess they stay alive for 12 hours. So every day I have to restart the run to see the results. To reduce the time I have taken some parameters as static and then proceeded further. Although I would have liked to play around more but considering the time, I had to reject the idea.

- To reduce the dependency on the GPU based libraries I tried to use Elasticsearch. But the problem there is the pandassearch library has some serious shortcoming. Only 10000 records can be fetched at a shot from there. Apart from that I did not get enough time to explore the exploration of executing certain algorithms on spark.
- As execution time was crucial I had to add a *timeit* function to keep tab on the execution time.
- First set of refinement was to visualize the data and data was normalized.

- I tried to use some outlier detection to just view how they perform. This was just an additional methods before I could proceed with LinearRegression.
- The second step was to get rid of the unbalance. First I tried to make a balanced dataset using a hand coded quick method. But this method has a lot of data loss. I had to drop almost all of the records as I only choose another set of 487 records to match the same amount of fraud data. That's a huge loss. So I had to go with better methods that's supported by library. The best method that I could find out of them was SMOTEENN.
- Once this was done I had to come up with a baseline so the methods could be further improved. To achieve this I had to use LogisticRegression, but the hyperparameters were done using GridSearch.
- Once a baseline was set I went ahead with a multilayer perceptron to train the model and then find out their performance.
- *lbgs* was considered for all the scorers considering the execution time and as this only supports l2 penalty, hence l2 was used.
- To make the notebook more interactive I have tried using google forms in google Collab notebook. This helps playing around with parameters easier.

IV. Results

In this section we dwell further on the results and what kind of refinements we can do on the results. The main focus of the result will be the ROC-AUC score as we have oversampled our data and normalized it.

Model Evaluation and Validation

As we have discussed earlier for this kind of problems confusion matrix is not a valid validation method. Rather *ROC-AUC* and *PR-AUC* is a valid metric. Considering we have balanced the data ROC-AUC can be the primary metric to evaluate. A good model has AUC near to the 1 which means it has good measure of separability of how good it is at separating different values. A poor model has AUC near to the 0 which means it has cannot separate the results.

For us the NN model gives ROC−AUC=99.97% with the , even with the test data which is not resampled the value is ROC−AUC=99.96%.

Both the ROC and PR curves are tools to help interpret the efficiency of the models for binary forecast. PR Curve is most useful when there is an imbalance. Here the data imbalance is removed. For a model to have perfect skill the curve should be inclined towards 1,1 point. In the above curves for Logistic Regression we see that the precision drops after 0.8 in the neural network model also. Apart from that the AUC value is 85%, which is not a great value. This is even lower in the case of the Logistic regression which is 76% around. So there is definitely some improvement in the neural network model. I believe I will have to play around with the different activation functions to see if it can be improved. I guess the sigmoid models are better at fitting complex examples.

Justification

After normalizing and resampling the data the model that we created performed quite good ROC−AUC= = 98.95%. The challenge was to improve on this data. The neural network based model gave performance ROC−AUC=99.97%. The improvement is around *1%* but still it's an improvement. Initially when I started the project I did not consider normalizing. Gradually while progressing I read more about normalizing and also checked that the data performed well while using the Logistic Regression in case the data was normalized. The other factor that helped was resampling. To come up with the best method to resample I had to test out few of the available resampling methods using the Logistic regression. Both of the above methods paid well at the end. This helped in getting a better score at the end.

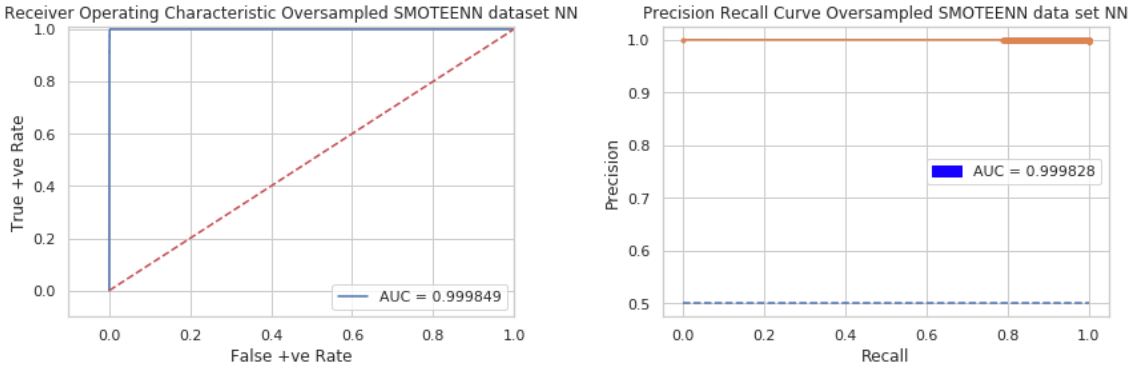
V. Conclusion

This section we come to the conclusion on the methods we have been using and on the results we have got.

Free-Form Visualization

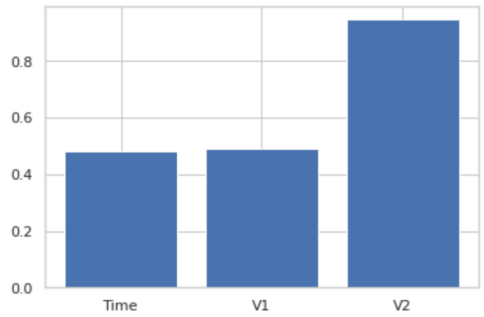
We have already analysed the results that is the ROC−AUC and PR-AUC. Again to emphasise that the ROC−AUC is the primary curve we are interested in as we have resampled the data. If the imbalance was great we would have chosen PR-AUC. Still both the curves have shown improvements using the neural network method than logistic regression as shown here.

ROC-AUC and PR-AUC



Apart from this if we want to see the comparison of the mean absolute error on test data, we can see for the Logistic Regression it is: *0.053711* and for neural network it is: *0.0009181324107884726*

Apart from that we can see what are the first few variables that contribute more are more important. This is called the feature importance. We can see the feature importance for the Logistic Regression:



From here we can see that the most important features are *Time*, *V1* and *V2*.

Reflection

I work in travel and expenses domain. So this project was interesting to me. In this what I have learned will be a baseline for similar fraud detection mechanisms I can use in certain other use cases too like fraud bill detection etc. As an architect this gave me insight on the execution time and resource usages too. This is also very important as cost is important for profit.

The easiest thing in this project is getting the data, but the difficulty in that was the data was already made anonymous. So getting the relation between different fields was not feasible. I tried using correlation metrics and also the heat map. Though the heat map gave vague indication of different field relation it was not trivial what was the relation. But this kind of field anonymization is important due to different government rules like GDPR etc.

Another important aspect of the problem at hand was data normalization. For this I had to plot the data distribution and then find out which all the fields that need to be normalized. I have normalized only the fields which had a lot of skew.

The execution time was a big concern for me. Although I finally used google Collab, it was still difficult sometimes as the run time gets rest at night and if not used for some time. I believe that Kaggle, Azure and AWS even have similar facility. Need to properly check that. Even in commercial cases these adds cost and procuring them is not trivial ass it goes through various approvals. So this I have to keep in mind.

Linear Regression and NN I have used mostly basic configurations as considering the execution time is quite high. I used outlier detection mechanisms also to get a quick look at what the amount of accuracy I can get. This was just to see a baseline execution time and a quick model.

Improvement

I would like to use CNN or RNN also to further see the results. As I mentioned in previous section this project will act as a skeletal project for other experiments I need to do. So I will be improving this project continuously.

As suggested in different other sections need to try to try different activation function for the neural network too. Need to try non-linear activation functions. Apart from that I will have to see and increase the gridsearch hyper parameters too and see if I can get better result with the logistic regression too.

One major improvement that I want to do is use Elasticsearch as the storage. In real world use cases data will hardly be received using csv. The data will come from some database. One such database which I have used with Spark is Elasticsearch. I would like to improve on that. Apart from that another of my interest is using some other libraries that has less dependency on CUDA. Something else I am willing to try is mxnet, caffe2 etc. There are also some mobile platform deep learning libraries. Being an architect this will be important for me to try out different platforms.

VI. Reference

- Solve imbalanced dataset
- 7 techniques to handle imbalance in data
- SMOTE with imbalanced data
- Understanding ROC-AUC
- Difference between ROC-AUC and PR-AUC
- Precision-Recall AUC vs ROC AUC for class imbalance problems
- Class imbalance problem
- Comparison of combination of Under sampling and over sampling
- Introduction to the precision-recall plot
- Anomaly Detection with Isolation Forest & Visualization
- Feature scaling with scikit learn
- Compare the effect of different scalers on data with outliers
- The best explanation of Convolutional Neural Networks on the Internet!
- Multilayer Perceptron (MLP) vs Convolutional Neural Network in Deep Learning
- When to Use MLP, CNN, and RNN Neural Networks
- Fraud Detection using Keras LSTM
- Crash Course On Multi-Layer Perceptron Neural Networks
- Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says
- Awesome Distributed Deep Learning
- Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems
- Resampling techniques and other strategies
- Credit Card Fraud Detection Using Random Forest and Local Outlier Factor
- Credit Card Fraud Detection using Local Outlier Factor
- A Survey on Outlier Detection Techniques for Credit Card Fraud Detection
- Resampling strategies for imbalanced datasets
- Python Machine Learning Cookbook - Second Edition
- LEARNING PATH: Machine Learning & Deep Learning with TensorFlow
- Under-Sampling Approaches for Improving Prediction of the Minority Class in an Imbalanced Dataset