Anomaly Detection

Under the guidance of
Prof. Sri Krishnamurthy
TA Ashwin Dinoria

By: Team 7

Lalit Jain
Lipsa Panda
Sameer Goel

TABLE OF CONTENTS

OBJECTIVE

The goal of report is to detect credit card fraudulent transaction using anomaly detection techniques. We will try different algorithms to track down all the fraudulent transaction.

As fraud transaction to authentic transition ratio is too high, confusion matrix accuracy is not meaningful for unbalanced classification. So are measuring the accuracy using the **Area Under the Precision-Recall Curve** (**AUPRC**).

INPUT DATASET

It is very important to know about this dataset as this dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset present transactions that occurred in two days, where we have **492** frauds out of **284,807** transactions. The dataset is highly unbalanced, the positive class (frauds) account for **0.172%** of all transactions.

https://www.kaggle.com/account/login?ReturnUrl=%2fdalpozz%2fcreditcardfraud%2fdownloads%2fcreditcardfraud.zip

DATA ABOUT DATA

Due to confidentiality issues, input variables are PCA transformed. Features V1, V2, to V28 are the principal components obtained with PCA, the only features which is not been transformed with PCA are '**Time**' and '**Amount**'.

- **Time:** It contains the seconds elapsed between each transaction and the first transaction.
- **Amount:** It is the transaction amount, used for example dependent cost-sensitive learning.
- **Class**: It is the response variable and it takes value 1 in case of fraud and 0 otherwise**.**

Below is the screenshot from the data.

Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
0	-1.3598	-0.0728	2.53635	1.37816	-0.3383	0.46239	0.2396	0.0987	0.36379	0.09079
0	1.19186	0.26615	0.16648	0.44815	0.06002	-0.0824	-0.0788	0.0851	-0.2554	-0.167
	-1.3584	-1.3402	1.77321	0.37978	-0.5032	1.8005	0.79146	0.24768	-1.5147	0.20764
1	-0.9663	-0.1852	1.79299	-0.8633	-0.0103	1.2472	0.23761	0.37744	-1.387	-0.055
2	-1.1582	0.87774	1.54872	0.40303	-0.4072	0.09592	0.59294	-0.2705	0.81774	0.75307
2	-0.426	0.96052	1.14111	-0.1683	0.42099	-0.0297	0.4762	0.26031	-0.5687	-0.3714
4	1.22966	0.141	0.04537	1.20261	0.19188	0.27271	-0.0052	0.08121	0.46496	-0.0993
7	-0.6443	1.41796	1.07438	-0.4922	0.94893	0.42812	1.12063	-3.8079	0.61537	1.24938
7	-0.8943	0.28616	-0.1132	-0.2715	2.6696	3.72182	0.37015	0.85108	-0.392	-0.4104
9	-0.3383	1.11959	1.04437	-0.2222	0.49936	-0.2468	0.65158	0.06954	-0.7367	-0.3668
10	1.44904	-1.1763	0.91386	-1.3757	-1.9714	-0.6292	-1.4232	0.04846	-1.7204	1.62666
10	0.38498	0.61611	-0.8743	-0.094	2.92458	3.31703	0.47045	0.53825	-0.5589	0.30976

Time	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20
0	-0.5516	-0.6178	-0.9914	-0.3112	1.46818	-0.4704	0.20797	0.02579	0.40399	0.25141
0	1.61273	1.06524	0.4891	-0.1438	0.63556	0.46392	-0.1148	-0.1834	-0.1458	-0.0691
	0.6245	0.06608	0.71729	-0.1659	2.34586	-2.8901	1.10997	-0.1214	-2.2619	0.52498
1	-0.2265	0.17823	0.50776	-0.2879	-0.6314	-1.0596	-0.6841	1.96578	-1.2326	-0.208
2	-0.8228	0.5382	1.34585	-1.1197	0.17512	-0.4514	-0.237	-0.0382	0.80349	0.40854
2	1.34126	0.35989	-0.3581	-0.1371	0.51762	0.40173	-0.0581	0.06865	-0.0332	0.08497
4	-1.4169	-0.1538	-0.7511	0.16737	0.05014	-0.4436	0.00282	-0.612	-0.0456	-0.2196
7	-0.6195	0.29147	1.75796	-1.3239	0.68613	-0.0761	-1.2221	-0.3582	0.3245	-0.1567
7	-0.7051	-0.1105	-0.2863	0.07436	-0.3288	-0.2101	-0.4998	0.11876	0.57033	0.05274
9	1.01761	0.83639	1.00684	-0.4435	0.15022	0.73945	-0.541	0.47668	0.45177	0.20371
10	1.19964	-0.6714	-0.5139	-0.095	0.23093	0.03197	0.25341	0.85434	-0.2214	-0.3872
10	-0.2591	-0.3261	-0.09	0.36283	0.9289	-0.1295	-0.81	0.35999	0.70766	0.12599

Time	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	-0.0183	0.27784	-0.1105	0.06693	0.12854	-0.1891	0.13356	-0.0211	149.62	
0	-0.2258	-0.6387	0.10129	-0.3398	0.16717	0.12589	-0.009	0.01472	2.69	
	0.248	0.77168	0.90941	-0.6893	-0.3276	-0.1391	-0.0554	-0.0598	378.66	
1	-0.1083	0.00527	-0.1903	-1.1756	0.64738	-0.2219	0.06272	0.06146	123.5	
2	-0.0094	0.79828	-0.1375	0.14127	-0.206	0.50229	0.21942	0.21515	69.99	
2	-0.2083	-0.5598	-0.0264	-0.3714	-0.2328	0.10591	0.25384	0.08108	3.67	
4	-0.1677	-0.2707	-0.1541	-0.7801	0.75014	-0.2572	0.03451	0.00517	4.99	
7	1.94347	-1.0155	0.0575	-0.6497	-0.4153	-0.0516	-1.2069	-1.0853	40.8	
7	-0.0734	-0.2681	-0.2042	1.01159	0.3732	-0.3842	0.01175	0.1424	93.2	
9	-0.2469	-0.6338	-0.1208	-0.385	-0.0697	0.0942	0.24622	0.08308	3.68	
10	-0.0093	0.31389	0.02774	0.50051	0.25137	-0.1295	0.04285	0.01625	7.8	
10	0.04992	0.23842	0.00913	0.99671	-0.7673	-0.4922	0.04247	-0.0543	9.99	

The datasets contains transactions made by credit cards in September 2013 by European cardholders where there 492 frauds out of 284,807 transactions.

ANOMALY DETECTION: INTRODCUTION

Anomaly detection detects data points in data that does not fit well with the rest of the data. It has a wide range of applications such as fraud detection, surveillance, diagnosis, data cleanup, and predictive maintenance.

However, often it is very hard to find training data, and even when you can find them, most anomalies are 1:1000 to 1:10^6 events where classes are not balanced.

Anomalies or outliers has three types.

1. Point Anomalies. If an individual data instance can be considered as anomalous with respect to the rest of the data (e.g. purchase with large transaction value)
2. Contextual Anomalies, If a data instance is anomalous in a specific context, but not otherwise (anomaly if occur at certain time or certain region. e.g. large spike at middle of night)
3. Collective Anomalies. If a collection of related data instances is anomalous with respect to the entire data set, but not individual values. They have two variations.
    a. Events in unexpected order (ordered. e.g. breaking rhythm in ECG)
    b. Unexpected value combinations (unordered. e.g. buying large number of expensive items)

POINT ANOMALY

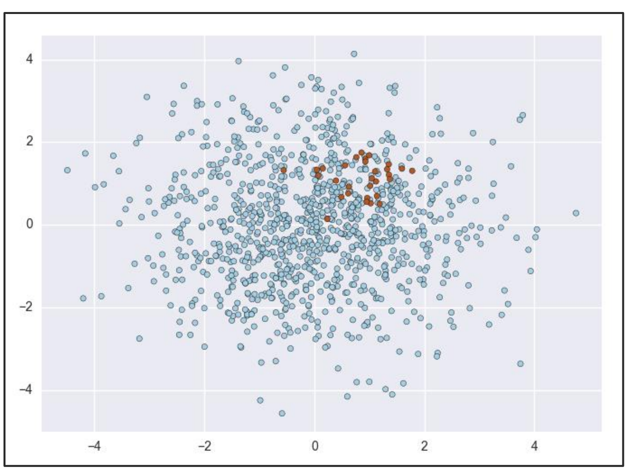
As this is credit card fraud detection, we can narrow down our approach towards point anomaly detection. Anomalies are rare under most conditions. Hence, even when training data is available, often there will be few anomalies exists among millions of regular data points. The standard classification methods such as **SVM** or **Random Forest** will classify almost all data as normal because doing that will provide a very high accuracy score (e.g. accuracy is 99.9 if anomalies are one in thousand).

Generally, the class imbalance is solved using an **ensemble** built by resampling data many times. The idea is to first create new datasets by taking all anomalous data points and adding a subset of normal data points (e.g. as 4 times as anomalous data points). Then a classifier is built for each data set using SVM or Random Forest, and those classifiers are combined using ensemble learning. This approach has worked well and produced very good results.

DATA PROCESSING

We will start our analysis by checking the imbalance or checking the probabilities of our target values.

```
> prop.table(table(dataset$Class))

          0           1
0.998272514 0.001727486
```



We can see that dataset is highly skewed, we have 0.17% fraudulent data points against 99.8% regular data pint. Also the input columns are PCA transformed so we are not require to do any feature engineering with this dataset.

SPLITTING DATASET

We have split the dataset so that we get nearly equal number of 1's in our train and test dataset.

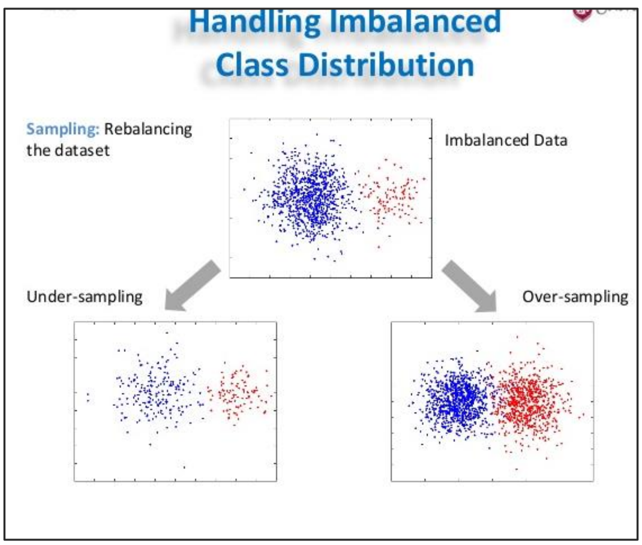
```
> prop.table(table(train$Class))

          0           1
0.998136964 0.001863036
```

```
> prop.table(table(test$Class))

          0           1
0.998675294 0.001324706
```

We can see that train data has 0.18% and test data has 0.13% of the fraud data samples, but this would not be sufficient to train any model, so we need to think of any technique to proportionate our class label.

HANDLING IMBALANCED DATA

We can handle imbalance data using various techniques.



1. **Data sampling:** In which the training instances are modified in such a way to produce a more or less balanced class distribution that allow classifiers to perform in a similar manner to standard classification. **Oversample the minority class, Under sample the majority class, Synthesize new minority classes.**

   E.g. SMOTE, ROSE, EasyEnsemble, BalanceCascade, etc.

2. **Algorithmic modification:** This procedure is oriented towards the adaptation of base learning methods to be more attuned to class imbalance issues

3. **Cost-sensitive learning:** This type of solutions incorporate approaches at the data level, at the algorithmic level, or at both levels combined, considering higher costs for the misclassification

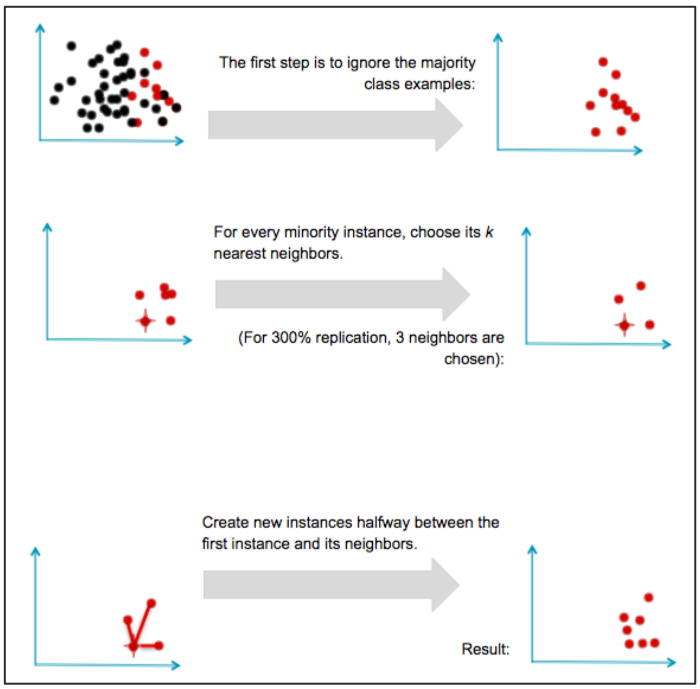
of examples of the positive class with respect to the negative class, and therefore, trying to minimize higher cost errors

E.g. CostSensitiveClassifier.

SMOTE: SYNTHETIC MINORITY OVERSAMPLING TECHNIQUE

We are using SMOTE techniques to handle the imbalance data. Here we are generating artificial anomalies to balance the ratio of class labels.

- New rare class examples are generated inside the regions of existing rare class examples.
- Artificial anomalies are generated around the edges of the sparsely populated data regions Classify synthetic outliers vs. real normal data using active learning.



We are using DMwR library to facilitate oversampling to our dataset. It is called Synthetic Minority Oversampling Technique (SMOTE). New rare class examples are generated inside the regions of existing rare class examples.

The count of test of train is imbalanced as we can see the table of test and train below:

```
> table(train$Class)

     0      1
212696    397
> table(test$Class)

    0     1
71619    95
```

Now we will apply our SMOTE()

```r
## Using SMOTE to handle imbalanced dataset
#set seed for reproducibility
set.seed(7)
#our SMOTed dataset and model using DMwR package
require(DMwR)
train_smote <- SMOTE(Class~., data = train, perc.over = 200, k = 5, perc.under = 200)
set.seed(6)
#putting randomness.
split <- sample(1:nrow(train_smote), nrow(train_smote))
#random train set.
train_smot <- train_smote[split,]


## doing it for testing dataset
set.seed(7)
#our SMOTed dataset and model using DMwR package
require(DMwR)
test_smote <- SMOTE(Class~., data = test, perc.over = 200, k = 5, perc.under = 200)
set.seed(6)
#putting randomness.
split <- sample(1:nrow(test_smote), nrow(test_smote))
#random train set.
test_smote <- test_smote[split,]
```

Now we have our dataset transformed after SMOTE as:

```
> table(test_smote$Class)

  0   1
380 285
> table(train_smot$Class)

   0    1
1588 1191
```

Now as we have our dataset balanced, we can go ahead and evaluate the different algorithms.

EVALUATING DIFFERENT ALGORITHMS

Now we have the dataset ready to try out with different algorithms and measuring the accuracy using ROC Region Under Curve.

RANDOM FOREST

1) We will do calibration of the Random forest algorithm, starting from a cut off of 0 to 0.5 in a step of 0.1. We will use a function to do this and get the threshold with the highest AUC.

```r
getThresholdRF <- function(train,test){

  c <- c()
  f <- c()
  j <- 1

  library(randomForest)
  library(pROC)
  for(i in seq(0, 0.5 , 0.01)){
    set.seed(7)
    fit <- randomForest(Class~., data = train)
    pre <- predict(fit, test, type = "prob")[,2]
    pre <- as.numeric(pre > i)
    auc <- roc(test$Class, pre)
    c[j] <- i
    f[j] <- as.numeric(auc$auc)
    j <- j + 1
  }


  df <- data.frame(c = c, f = f)
  p <- df$c[which.max(df$f)]
  return(p)
}
```

Threshold	AUC
0.00	0.5342105
0.01	0.6552632
0.02	0.7364035
0.03	0.7771930
0.04	0.8162281
0.05	0.8390351
0.06	0.8570175
0.07	0.8701754
0.08	0.8824561
0.09	0.8956140
0.10	0.9030702
0.11	0.9026316
0.12	0.9092105
0.13	0.9140351
0.14	0.9122807

2) As we can see we got the threshold of 0.34 with the highest AUC

3) Applying algorithm again using the evaluated threshold of 0.34

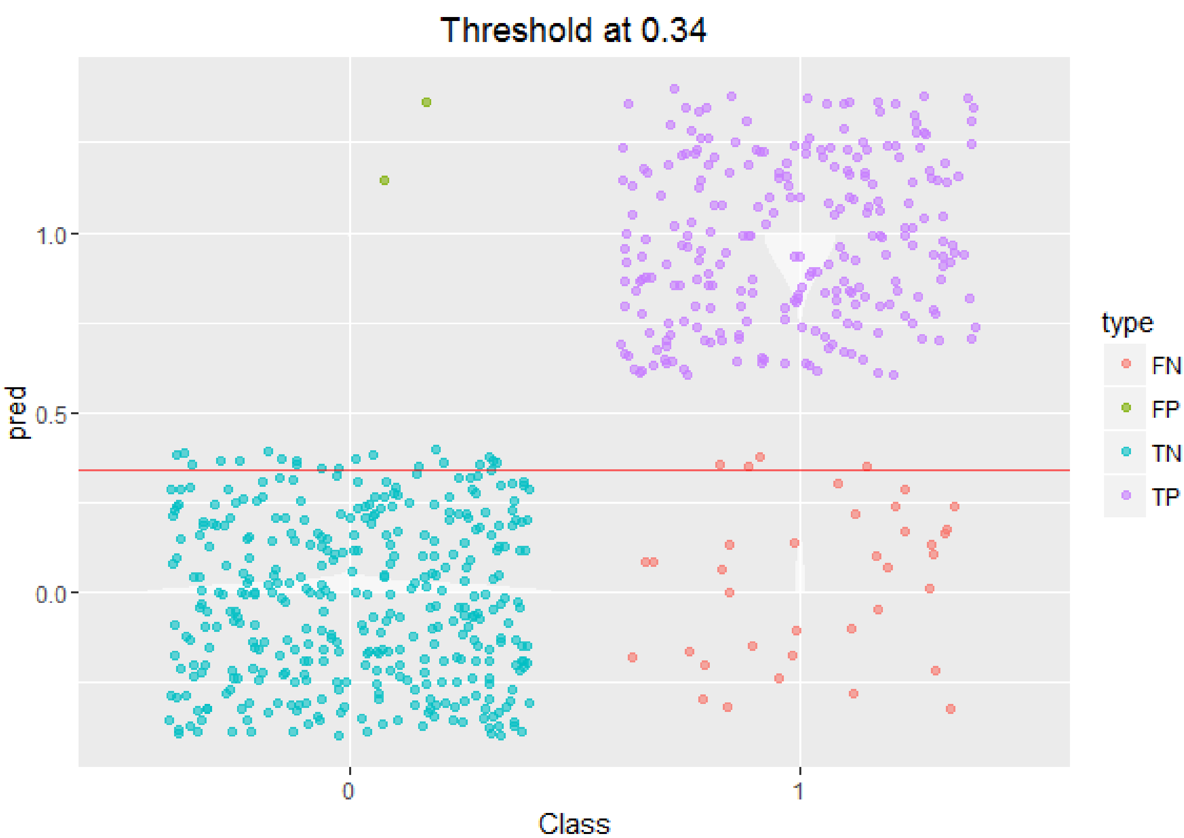
```r
##### RANDOM FOREST
set.seed(47)
source("ThresholdCalibration.R")
threshold <- getThresholdRF(train_smot,test_smote)

fit <- randomForest(formula, data = train_smot)
pre <- predict(fit, test_smote, type = "prob")[,2]
pre <- as.numeric(pre > threshold)
```

4) Now we will evaluate the confusion matrix, we can see that 4 false predicted records are there and difference of sensitivity and specificity is around 0.06.

```
## Random Forest

THRESHOLD: 0.34
Confusion Matrix and Statistics

Reference
Prediction    0   1
0 376    4
1  30 255

Accuracy : 0.9489
95% CI : (0.9293, 0.9643)
No Information Rate : 0.6105
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.8944
Mcnemar's Test P-Value : 1.807e-05

Sensitivity : 0.9261
Specificity : 0.9846
Pos Pred Value : 0.9895
Neg Pred Value : 0.8947
Prevalence : 0.6105
Detection Rate : 0.5654
Detection Prevalence : 0.5714
Balanced Accuracy : 0.9553

'Positive' Class : 0
```

5) We will plot the distribution of the confusion matrix using the below function and see the result

```
### Function for checking the distribution with threshold
plot_pred_type_distribution <- function(df, threshold) {
  v <- rep(NA, nrow(df))
  v <- ifelse(df$pred >= threshold & df$Class == 1, "TP", v)
  v <- ifelse(df$pred >= threshold & df$Class == 0, "FP", v)
  v <- ifelse(df$pred < threshold & df$Class == 1, "FN", v)
  v <- ifelse(df$pred < threshold & df$Class == 0, "TN", v)

  df$pred_type <- v

  ggplot(data=df, aes(x=Class, y=pred)) +
    geom_violin(fill=rgb(1,1,1,alpha=0.6), color=NA) +
    geom_jitter(aes(color=pred_type), alpha=0.6) +
    geom_hline(yintercept=threshold, color="red", alpha=0.6) +
    scale_color_discrete(name = "type") +
    labs(title=sprintf("Threshold at %.2f", threshold))
}

library(ggplot2)
test_pred <- test_smote
test_pred$pred <- pre
plot_pred_type_distribution(test_pred, threshold)
```



LOGISTIC REGRESSION

1) We will perform the similar calibration that we performed earlier with random forest. Starting from a cut off from 0 to 0.5 in a step of 0.1.

```
getThresholdLogistic <- function(train,test){

  c <- c()
  f <- c()
  j <- 1


  library(pROC)
  for(i in seq(0, 0.5 , 0.01)){
    set.seed(49)
    model2 <- glm(Class~.,data = train,
                  family = binomial(link = "logit"))
    results <- predict(model2,test[,-c(31)],
                       type = 'response')
    pre <- as.numeric(results > i)
    auc <- roc(test$Class, pre)
    c[j] <- i
    f[j] <- as.numeric(auc$auc)
    j <- j + 1
  }


  df <- data.frame(c = c, f = f)
  p <- df$c[which.max(df$f)]
  return(p)
}
```

2) We will apply algorithm again based on our evaluated treshold.

```
##### Logistic Regression

set.seed(7)
test_smote <- SMOTE(Class~., data = test, perc.over = 200, k = 5, perc.under = 200)
set.seed(6)
#putting randomness.
split <- sample(1:nrow(test_smote), nrow(test_smote))
#random train set.
test_smote <- test_smote[split,]
```

3) Now we will evaluate the confusion matrix, we can see that 7 false predicted records are there and difference of sensitivity and specificity is around 0.04, which is lesser than the random forest.

```
## Logistic Regression
Threshold = 0.25
Confusion Matrix and Statistics

Reference
Prediction   0   1
0 373   7
1  25 260

Accuracy : 0.9519
95% CI : (0.9327, 0.9669)
No Information Rate : 0.5985
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.901
Mcnemar's Test P-Value : 0.002654

Sensitivity : 0.9372
Specificity : 0.9738
Pos Pred Value : 0.9816
Neg Pred Value : 0.9123
Prevalence : 0.5985
Detection Rate : 0.5609
Detection Prevalence : 0.5714
Balanced Accuracy : 0.9555

'Positive' Class : 0
```

NEURAL NETWORK

1) We will perform the similar calibration that we performed earlier with random forest. Starting from a cut off from 0 to 0.5 in a step of 0.1.

```
##### Neural Network Classification

set.seed(7)
test_smote <- SMOTE(Class~., data = test, perc.over = 200, k = 5, perc.under = 200)
set.seed(6)
#putting randomness.
split <- sample(1:nrow(test_smote), nrow(test_smote))
#random train set.
test_smote <- test_smote[split,]


set.seed(561)
source("ThresholdCalibration.R")
threshold <- getThresholdNN(train_smot,test_smote)

library(neuralnet)
model4 <- neuralnet(Class~.,data = train_smot, hidden=17,linear.output=FALSE)
pr.nn <- neuralnet::compute(nn,test_smote[,-31])

pre <- as.numeric(pr.nn$net.result > threshold)

caret::confusionMatrix(test_smote$Class, factor(pre))

pr.nn <- neuralnet::compute(nn,test[,-31])
pre <- as.numeric(pr.nn$net.result > threshold)
caret::confusionMatrix(test$Class, factor(pre))
```

2) We will apply algorithm again based on our evaluated treshold.

```r
getThresholdNN <- function(train,test){

  c <- c()
  f <- c()
  j <- 1

  library(neuralnet)
  n <- names(train)
  fo <- as.formula(paste("Class ~",
                         paste(n[!n %in% c("Class")],
                               collapse = " + ")))
  library(pROC)
  for(i in seq(0, 0.5 , 0.01)){
    set.seed(999)
    start.time <- Sys.time()
    print(start.time)
    nn <- neuralnet(fo,
                    data=train,
                    linear.output = F,
                    hidden = 17,
                    threshold = 0.1)
    end.time <- Sys.time()
    time.taken <- end.time - start.time
    print(time.taken)
    pr.nn <- neuralnet::compute(nn,test[,-31])
    pre <- as.numeric(pr.nn$net.result  > i)
    auc <- roc(test$Class, pre)
    c[j] <- i
    f[j] <- as.numeric(auc$auc)
    j <- j + 1
  }
  df <- data.frame(c = c, f = f)
  p <- df$c[which.max(df$f)]
  return(p)
}
```

3) Now we will evaluate the confusion matrix, we can see that only 2 false predicted records are there but difference of sensitivity and specificity is now around 0.08.

```
## Neural Network
Threshold = 0.08

Confusion Matrix and Statistics

Reference
Prediction   0   1
0 378   2
1  36 249

Accuracy : 0.9428571
95% CI : (0.9224075, 0.9592487)
No Information Rate : 0.6225564
P-Value [Acc > NIR] : < 0.000000000000000022204

Kappa : 0.8815672
Mcnemar's Test P-Value : 0.00000008636119

Sensitivity : 0.9130435
Specificity : 0.9920319
Pos Pred Value : 0.9947368
Neg Pred Value : 0.8736842
Prevalence : 0.6225564
Detection Rate : 0.5684211
Detection Prevalence : 0.5714286
Balanced Accuracy : 0.9525377

'Positive' Class : 0
```

SUPPORT VECTOR MACHINE

1) We will perform the similar calibration that we performed earlier with random forest. Starting from a cut off from 0 to 0.5 in a step of 0.1.

```
### SVM
library(e1071)
model5 <- svm(Class ~.,data = train_smot,type="C-classification")


pred <- predict(model5,test_smote)
system.time(pred <- predict(model5,test_smote))
caret::confusionMatrix(test_smote$Class, factor(pred))
```

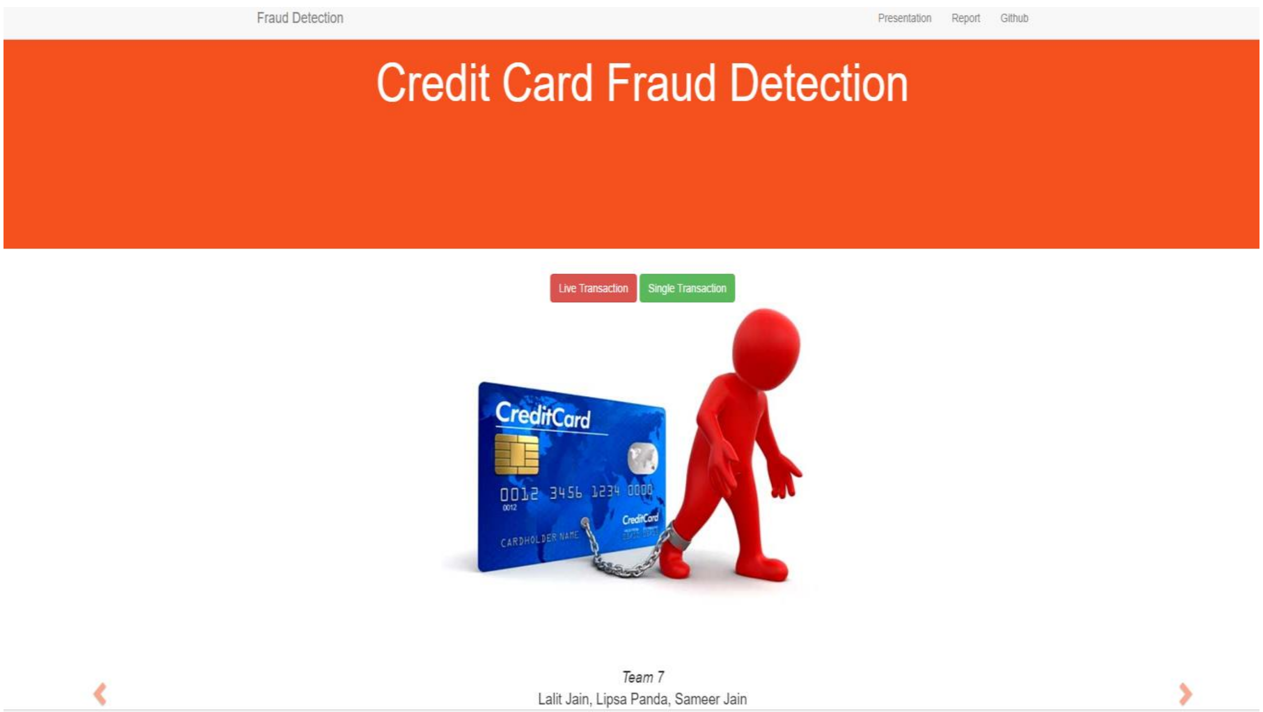
2) Now we will evaluate the confusion matrix, we can see that 5 false predicted records are there and difference of sensitivity and specificity is around 0.13 now.

```
## SVM

Confusion Matrix and Statistics

Reference
Prediction   0   1
0 375   5
1  71 214

Accuracy : 0.8857
95% CI : (0.8591, 0.9089)
No Information Rate : 0.6707
P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.7597
Mcnemar's Test P-Value : 8.918e-14

Sensitivity : 0.8408
Specificity : 0.9772
Pos Pred Value : 0.9868
Neg Pred Value : 0.7509
Prevalence : 0.6707
Detection Rate : 0.5639
Detection Prevalence : 0.5714
Balanced Accuracy : 0.9090

'Positive' Class : 0
```

DEMO – FRAUD DETECTION (WEB SITE)

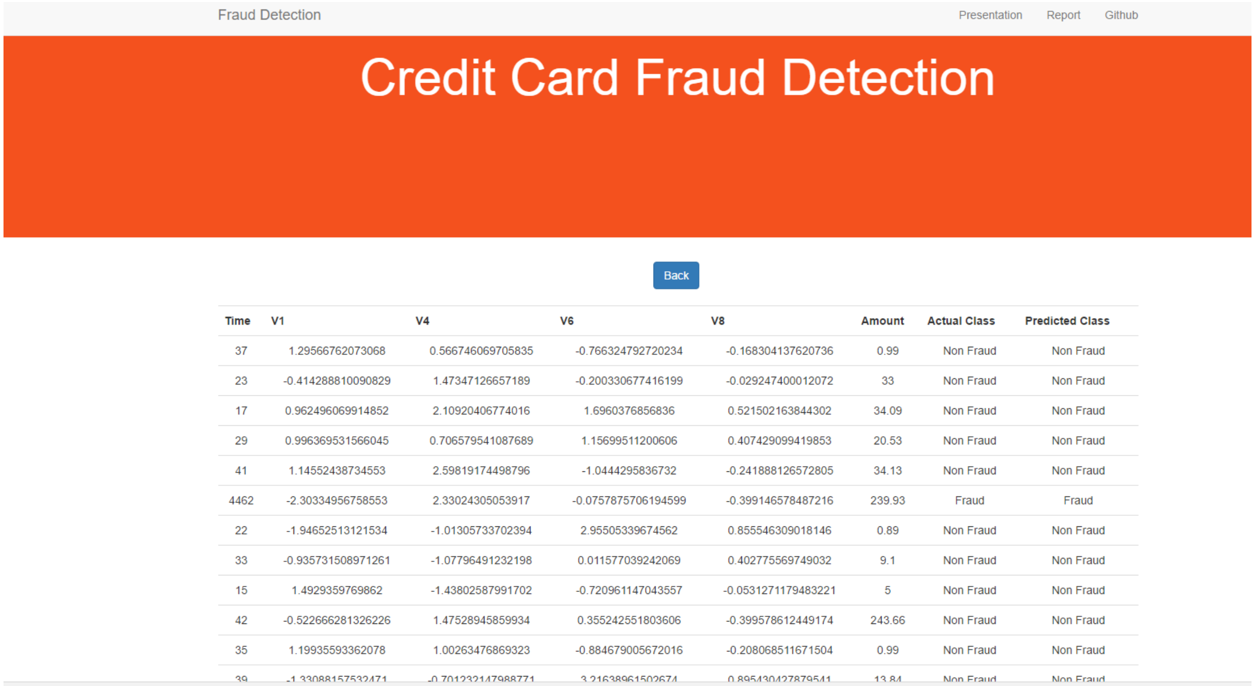
The website has two buttons on the Home Page :

- Live Transaction
- Single Transaction



LIVE TRANSACTION

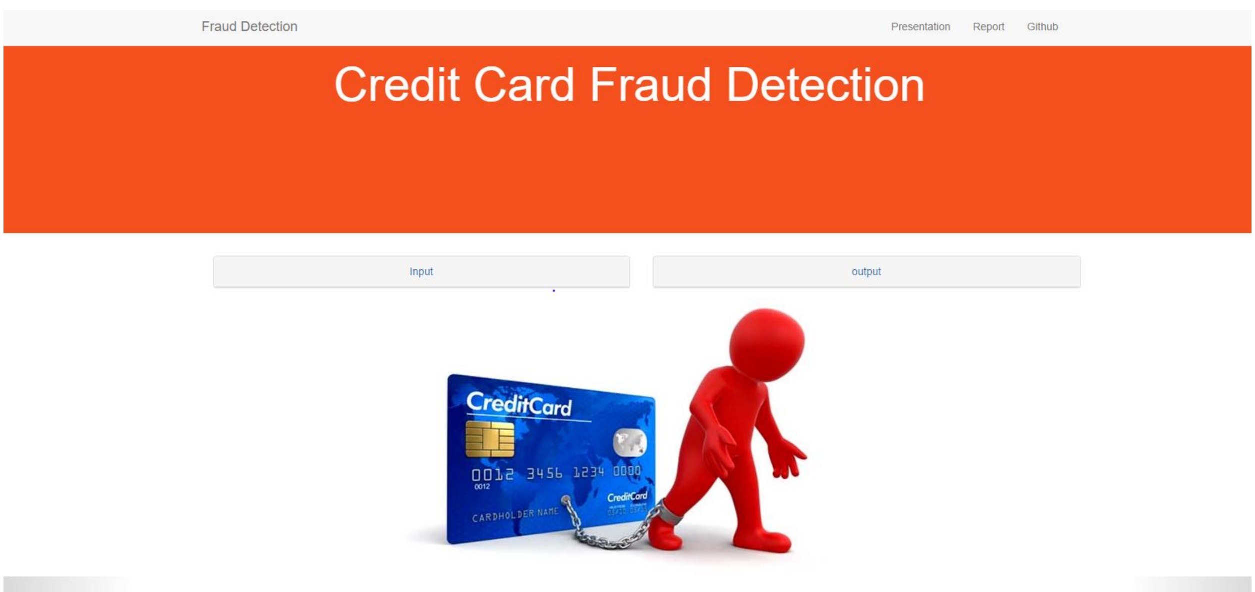
We are fetching the data that is trained under SMOTE and predicting whether the transaction is fraud or not.

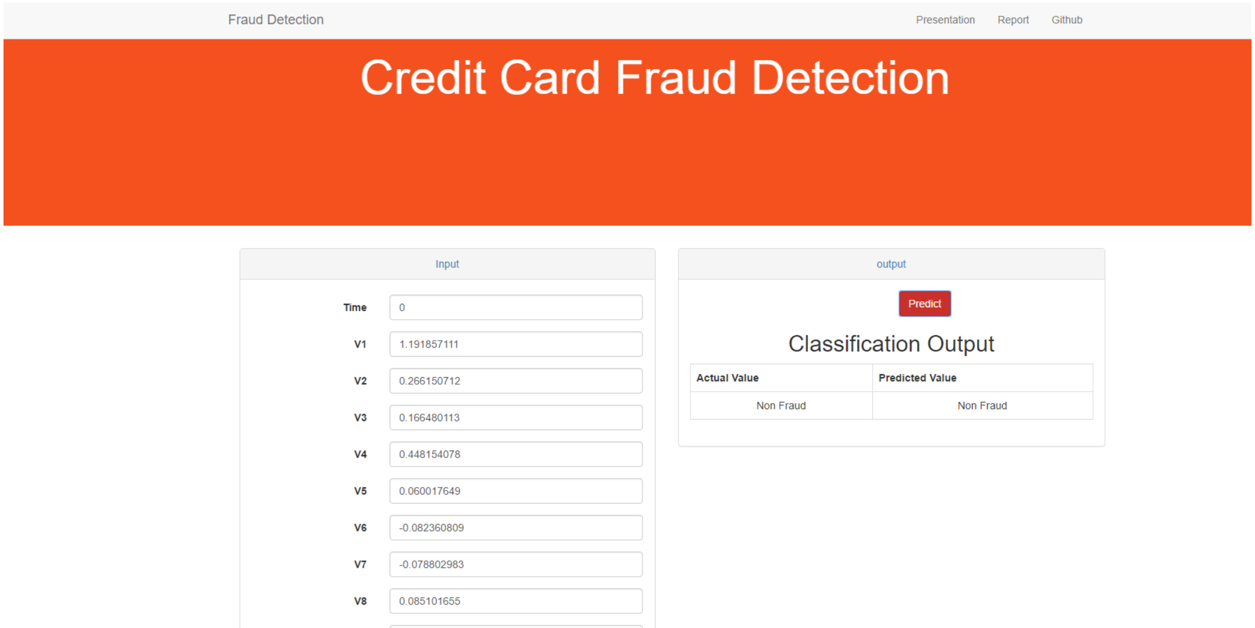


Time	V1	V4	V6	V8	Amount	Actual Class	Predicted Class
37	1.29566762073068	0.566746069705835	-0.766324792720234	-0.168304137620736	0.99	Non Fraud	Non Fraud
23	-0.414288810090829	1.47347126657189	-0.200330677416199	-0.029247400012072	33	Non Fraud	Non Fraud
17	0.962496069914852	2.10920406774016	1.6960376856836	0.521502163844302	34.09	Non Fraud	Non Fraud
29	0.996369531566045	0.706579541087689	1.15699511200606	0.407429099419853	20.53	Non Fraud	Non Fraud
41	1.14552438734553	2.59819174498796	-1.0444295836732	-0.241888126572805	34.13	Non Fraud	Non Fraud
4462	-2.30334956758553	2.33024305053917	-0.0757875706194599	-0.399146578487216	239.93	Fraud	Fraud
22	-1.94652513121534	-1.01305733702394	2.95505339674562	0.855546309018146	0.89	Non Fraud	Non Fraud
33	-0.935731508971261	-1.07796491232198	0.011577039242069	0.402775569749032	9.1	Non Fraud	Non Fraud
15	1.4929359769862	-1.43802587991702	-0.720961147043557	-0.0531271179483221	5	Non Fraud	Non Fraud
42	-0.522666281326226	1.47528945859934	0.355242551803606	-0.399578612449174	243.66	Non Fraud	Non Fraud
35	1.19935593362078	1.00263476869323	-0.884679005672016	-0.208068511671504	0.99	Non Fraud	Non Fraud
39	-1.33088157532471	-0.701232147988771	3.21638961502674	0.895430427879541	13.84	Non Fraud	Non Fraud

SAMPLE DATA

Here we are modelling the data using the one class SVM (Azure Machine Learning Algorithm) for predicting a single input transaction.




Fraud Detection
Presentation
Report
Github
Credit Card Fraud Detection
Input
Time
0
V1
1.191857111
V2
0.266150712
V3
0.166480113
V4
0.448154078
V5
0.060017649
V6
-0.082360809
V7
-0.078802983
V8
0.085101655
output
Predict
Classification Output
Actual Value
Predicted Value
Non Fraud
Non Fraud