# Capstone_CreditcardFraudDetection

*Sesha*

*10/10/2019*

## Introduction and Overview

Creditcard Fraud Detection Project Dataset is approx. 25mb data holding various transactions happened over a period of time for various set of users. The main objective of this project is to generate algorithm to detect Fraudrulant transactions based on existing dataset. Though there are several methods to generate predictive machine learning algorithm, here in this model various methods apart from Logistic Regression are being used, here are few - SVM, Randomforest models to generate efficient algorithm.

The key metric used to measure would be accuracy calculation there by estimate efficiency of this algorithm.

Creditcard Fraud Detection Project is based on Creditcard transactions file, consists of 50k transactions data along with time elapsed between transactions on same card, amount involved in transaction,class (binary value , if fraud then 0 else 1) along with 28 variables used to hold client details, however they were masked due to privacy constraints.

## Methods and Analysis

This is regarding Creditcard Fraud Detection Project. In this Project , an algorithm has been developed to predict ratings for random sample of creditcard transactions.This algorithm provides efficiency of predicted ratings upto 99%. This algorithm has been analysed using different methods-

Dataset is split into testset and trainsets for analysis. Since there are 50k records in dataset we have split into 70% train set and 30% data for testset, for some of models below dataset has been sampled with 10000 records with 149 fraud records.

Since algorithm needs to be trained and number of fraudulent transactions are relatively low around 500 transactions in the given dataset, we split data as 70% for training and 30% for test set. Initially Logistic Regression has been used , resulted in accuracy of 99.8%. with Decission tree model - accuracy is 99.925% with SVM Model- 98.8% and finally, Random Forest model provides an 100% accuracy.

tinytex used for pdf output,installing tinytex

```
system("ls ../", intern=TRUE)
```

```
##  [1] "Caret_q2.R"                    "Caret_q3.R"
##  [3] "Caret_q4.R"                    "Creditcard"
##  [5] "CreditcardFDetection.Rmd"      "CreditcardFDetection.html"
##  [7] "CreditcardFDetection.pdf"      "Creditcard_RF.Rmd"
##  [9] "Creditcard_RF.html"            "Creditcard_RF_1.Rmd"
## [11] "Creditcard_RF_1.html"          "Custom Office Templates"
## [13] "DimRed_q2.R"                   "Dim_red_q3.R"
## [15] "Dimension_reduction_Q1.R"      "Ensemble_all_q1.R"
## [17] "Identify_Creditcardfraud"      "Knn.R"
## [19] "Linear_Reg_Machine_Learning.R" "ML_pj_final.R"
## [21] "MovieLens.R"                   "MovieLens_RMSE.R"
## [23] "MovieLens_html.Rmd"            "MovieLens_html.html"
## [25] "Movielens-Project"             "My Music"
```

```
## [27] "My Pictures"                    "My Videos"
## [29] "Personal"                       "Q6_matrices.R"
## [31] "QDA_LDA_Q1.R"                   "R"
## [33] "RandomForest_1.R"               "RandomForest_q5.R"
## [35] "RandomForest_q6.R"              "Random_forest_caret_!.R"
## [37] "Random_forest_caret_1.R"        "Random_forest_q3.R"
## [39] "Recommendn_q1.R"                "Recommendn_q2.R"
## [41] "Recommendn_q3.R"                "Recommendn_q4.R"
## [43] "Recommendn_q6.R"                "Regularization_q1.R"
## [45] "Rplot.pdf"                      "SQL Server Management Studio"
## [47] "Visual Studio 2015"             "Visual Studio 2017"
## [49] "bootstrap_q1.R"                 "bootstrap_q2.R"
## [51] "bootstrap_q3.R"                 "bootstrap_q4.R"
## [53] "clustering_q1.R"                "correct_knn_genes.R"
## [55] "correct_mclearning_reg.R"       "correct_mclearning_reg_q4.R"
## [57] "crossval_2.R"                   "crossval_q4.R"
## [59] "crossvali.R"                    "cum_per_var_dimred_q6.R"
## [61] "desktop.ini"                    "dim_red_q5.R"
## [63] "distance.R"                     "ed_0"
## [65] "ex.Rmd"                         "ex.html"
## [67] "input"                          "knn1.R"
## [69] "knn_q2.R"                       "lda_qda_q6.R"
## [71] "linear_reg_q6.R"                "linear_reg_q8.R"
## [73] "linear_reg_show_ans_q7.R"       "linearreg_check_q4_latest.R"
## [75] "linearreg_mc.R"                 "logistic_reg_exe.R"
## [77] "logistic_reg_latest.R"          "matrices.R"
## [79] "matrix_fact_q1.R"               "ml-10M100K"
## [81] "murdersexample.Rmd"             "murdersexample_files"
## [83] "p2_mlpj.Rmd"                    "p2_mlpj.pdf"
## [85] "p2_mlpj_cache"                  "p2_mlpj_files"
## [87] "p3_ml_pj.Rmd"                   "practice_1.R"
## [89] "qda_lda_q4.R"                   "r_dslabs_course.R"
## [91] "showanswer_knn.R"               "showanswer_rf_mtry.R"
## [93] "smoothing.R"                    "smoothing_test.R"
```

```
if(!require(tinytex)) install.packages("tinytex")
```

```
## Loading required package: tinytex
```

Dataset file needs to be downloaded and placed at project location Below code downloads file and reads into csv variable

```
wd <-getwd()
wd
```

```
## [1] "C:/Users/User/Documents/Creditcard"
```

```
read_url_csv <- function(url){
  tmpFile <- tempfile()
  download.file(url, destfile = tmpFile)
  url_csv <- readr::read_csv(tmpFile)
  return(url_csv)
```

```
}

onedrive_url <- "https://raw.githubusercontent.com/seshlab/Creditcard/master/creditcarddataset1.csv"
csv <- read_url_csv(onedrive_url)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double()
## )
```

```
## See spec(...) for full column specifications.
```

```
head(csv)
```

```
## # A tibble: 6 x 31
##    Time     V1      V2    V3     V4      V5      V6      V7      V8     V9
##   <dbl>  <dbl>   <dbl> <dbl>  <dbl>   <dbl>   <dbl>   <dbl>   <dbl>  <dbl>
## 1     0 -1.36  -0.0728 2.54   1.38  -0.338   0.462   0.240   0.0987  0.364
## 2     0  1.19   0.266  0.166  0.448  0.0600 -0.0824 -0.0788  0.0851 -0.255
## 3     1 -1.36  -1.34   1.77   0.380 -0.503   1.80    0.791   0.248  -1.51
## 4     1 -0.966 -0.185  1.79  -0.863 -0.0103  1.25    0.238   0.377  -1.39
## 5     2 -1.16   0.878  1.55   0.403 -0.407   0.0959  0.593  -0.271   0.818
## 6     2 -0.426  0.961  1.14  -0.168  0.421  -0.0297  0.476   0.260  -0.569
## # ... with 21 more variables: V10 <dbl>, V11 <dbl>, V12 <dbl>, V13 <dbl>,
## #   V14 <dbl>, V15 <dbl>, V16 <dbl>, V17 <dbl>, V18 <dbl>, V19 <dbl>,
## #   V20 <dbl>, V21 <dbl>, V22 <dbl>, V23 <dbl>, V24 <dbl>, V25 <dbl>,
## #   V26 <dbl>, V27 <dbl>, V28 <dbl>, Amount <dbl>, Class <dbl>
```

Below are required packages to load libraries

```
library(readr)
library(randomForest)
library(e1071)
library(rpart)
library(rpart.plot)
library(caTools)
library(caret)
```

Below code turns file into factor format

```
datacard <- csv
str(datacard)
```

```
## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 284617 obs. of  31 variables:
##  $ Time  : num  0 0 1 1 2 2 4 7 7 9 ...
##  $ V1    : num  -1.36 1.192 -1.358 -0.966 -1.158 ...
##  $ V2    : num  -0.0728 0.2662 -1.3402 -0.1852 0.8777 ...
##  $ V3    : num  2.536 0.166 1.773 1.793 1.549 ...
##  $ V4    : num  1.378 0.448 0.38 -0.863 0.403 ...
##  $ V5    : num  -0.3383 0.06 -0.5032 -0.0103 -0.4072 ...
##  $ V6    : num  0.4624 -0.0824 1.8005 1.2472 0.0959 ...
```

```
##  $ V7    : num  0.2396 -0.0788 0.7915 0.2376 0.5929 ...
##  $ V8    : num  0.0987 0.0851 0.2477 0.3774 -0.2705 ...
##  $ V9    : num  0.364 -0.255 -1.515 -1.387 0.818 ...
##  $ V10   : num  0.0908 -0.167 0.2076 -0.055 0.7531 ...
##  $ V11   : num  -0.552 1.613 0.625 -0.226 -0.823 ...
##  $ V12   : num  -0.6178 1.0652 0.0661 0.1782 0.5382 ...
##  $ V13   : num  -0.991 0.489 0.717 0.508 1.346 ...
##  $ V14   : num  -0.311 -0.144 -0.166 -0.288 -1.12 ...
##  $ V15   : num  1.468 0.636 2.346 -0.631 0.175 ...
##  $ V16   : num  -0.47 0.464 -2.89 -1.06 -0.451 ...
##  $ V17   : num  0.208 -0.115 1.11 -0.684 -0.237 ...
##  $ V18   : num  0.0258 -0.1834 -0.1214 1.9658 -0.0382 ...
##  $ V19   : num  0.404 -0.146 -2.262 -1.233 0.803 ...
##  $ V20   : num  0.2514 -0.0691 0.525 -0.208 0.4085 ...
##  $ V21   : num  -0.01831 -0.22578 0.248 -0.1083 -0.00943 ...
##  $ V22   : num  0.27784 -0.63867 0.77168 0.00527 0.79828 ...
##  $ V23   : num  -0.11 0.101 0.909 -0.19 -0.137 ...
##  $ V24   : num  0.0669 -0.3398 -0.6893 -1.1756 0.1413 ...
##  $ V25   : num  0.129 0.167 -0.328 0.647 -0.206 ...
##  $ V26   : num  -0.189 0.126 -0.139 -0.222 0.502 ...
##  $ V27   : num  0.13356 -0.00898 -0.05535 0.06272 0.21942 ...
##  $ V28   : num  -0.0211 0.0147 -0.0598 0.0615 0.2152 ...
##  $ Amount: num  149.62 2.69 378.66 123.5 69.99 ...
##  $ Class : num  0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   Time = col_double(),
##   ..   V1 = col_double(),
##   ..   V2 = col_double(),
##   ..   V3 = col_double(),
##   ..   V4 = col_double(),
##   ..   V5 = col_double(),
##   ..   V6 = col_double(),
##   ..   V7 = col_double(),
##   ..   V8 = col_double(),
##   ..   V9 = col_double(),
##   ..   V10 = col_double(),
##   ..   V11 = col_double(),
##   ..   V12 = col_double(),
##   ..   V13 = col_double(),
##   ..   V14 = col_double(),
##   ..   V15 = col_double(),
##   ..   V16 = col_double(),
##   ..   V17 = col_double(),
##   ..   V18 = col_double(),
##   ..   V19 = col_double(),
##   ..   V20 = col_double(),
##   ..   V21 = col_double(),
##   ..   V22 = col_double(),
##   ..   V23 = col_double(),
##   ..   V24 = col_double(),
##   ..   V25 = col_double(),
##   ..   V26 = col_double(),
##   ..   V27 = col_double(),
```

```
##   ..   V28 = col_double(),
##   ..   Amount = col_double(),
##   ..   Class = col_double()
##   .. )
```

```
#count of rows in datacard

nrow(datacard)
```

```
## [1] 284617
```

```
#sample data from datacard
head(datacard)
```

```
## # A tibble: 6 x 31
##    Time     V1      V2    V3     V4      V5      V6      V7      V8     V9
##   <dbl>  <dbl>   <dbl> <dbl>  <dbl>   <dbl>   <dbl>   <dbl>   <dbl>  <dbl>
## 1     0 -1.36  -0.0728 2.54   1.38  -0.338   0.462   0.240   0.0987  0.364
## 2     0  1.19   0.266  0.166  0.448  0.0600 -0.0824 -0.0788  0.0851 -0.255
## 3     1 -1.36  -1.34   1.77   0.380 -0.503   1.80    0.791   0.248  -1.51
## 4     1 -0.966 -0.185  1.79  -0.863 -0.0103  1.25    0.238   0.377  -1.39
## 5     2 -1.16   0.878  1.55   0.403 -0.407   0.0959  0.593  -0.271   0.818
## 6     2 -0.426  0.961  1.14  -0.168  0.421  -0.0297  0.476   0.260  -0.569
## # ... with 21 more variables: V10 <dbl>, V11 <dbl>, V12 <dbl>, V13 <dbl>,
## #   V14 <dbl>, V15 <dbl>, V16 <dbl>, V17 <dbl>, V18 <dbl>, V19 <dbl>,
## #   V20 <dbl>, V21 <dbl>, V22 <dbl>, V23 <dbl>, V24 <dbl>, V25 <dbl>,
## #   V26 <dbl>, V27 <dbl>, V28 <dbl>, Amount <dbl>, Class <dbl>
```

```
# convert class variable to factor
datacard$Class <- factor(datacard$Class)
```

Predictive Modelling since this algorithm relatively less fraudulent data, datasplit at 70:30 works well inorder to acheive well balanced data both fraud and non fraudulent transactions. split data 70:30

```
set.seed(1)
splitdatacard <- sample.split(datacard$Class, SplitRatio = 0.7)
traindata <- subset(datacard, splitdatacard == T)
testdata <- subset(datacard, splitdatacard == F)

# check output Class distributiion
table(testdata$Class)
```

```
##
##     0     1
## 15025    45
```

baseline accuracy –> 99.826785 %

##logistic regression

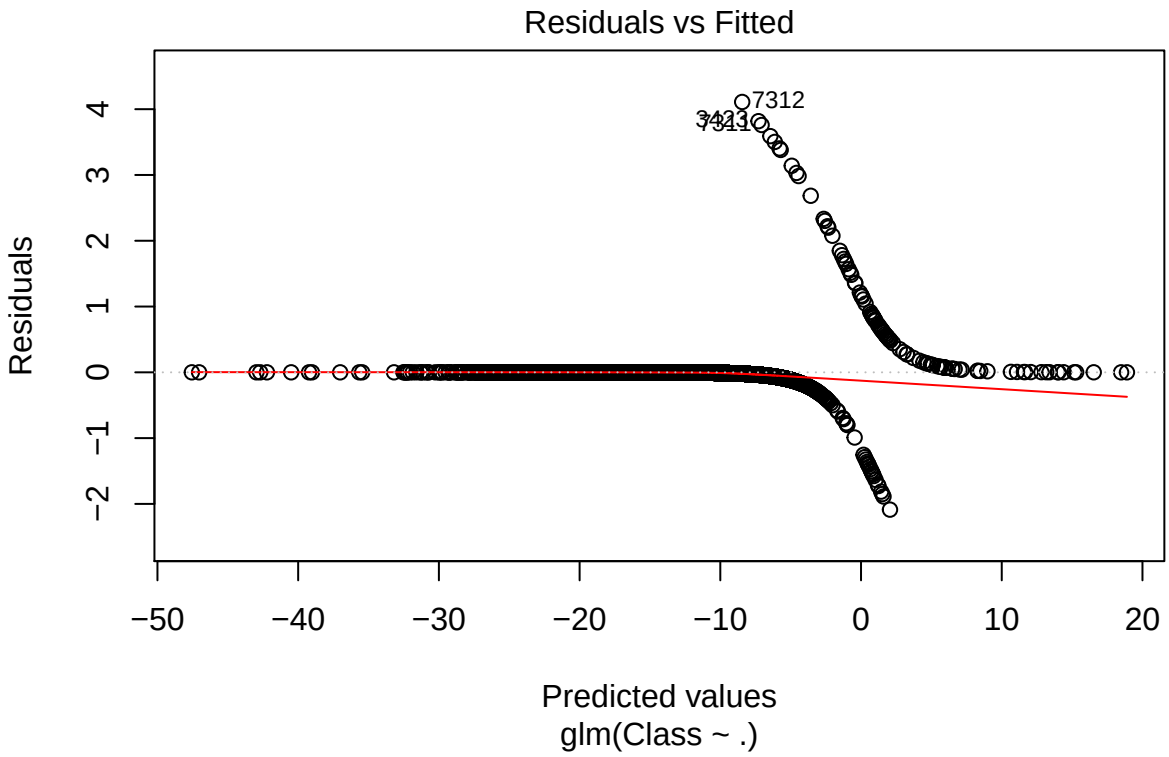
```
glm.model <- glm(Class ~ ., data = traindata, family = "binomial")
glm.predict <- predict(glm.model, testdata, type = "response")
table(testdata$Class, glm.predict > 0.5)
```

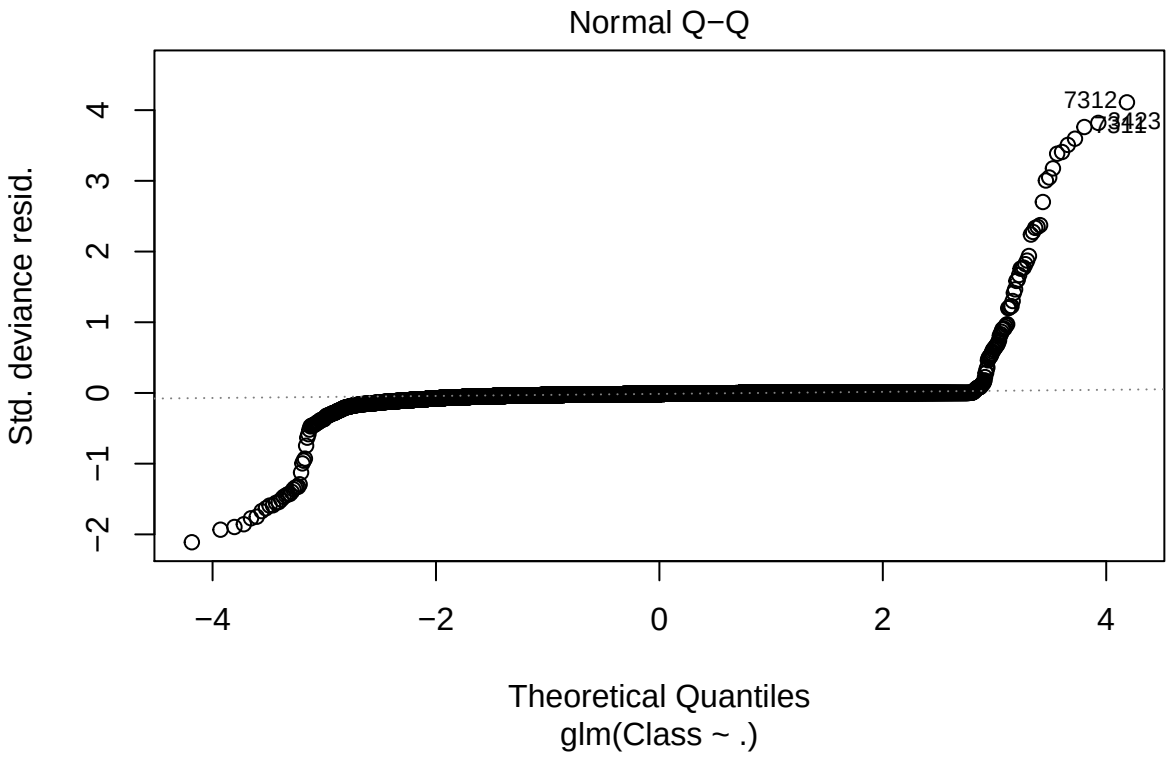
```
##
##     FALSE  TRUE
##   0 15011    14
##   1    13    32
```

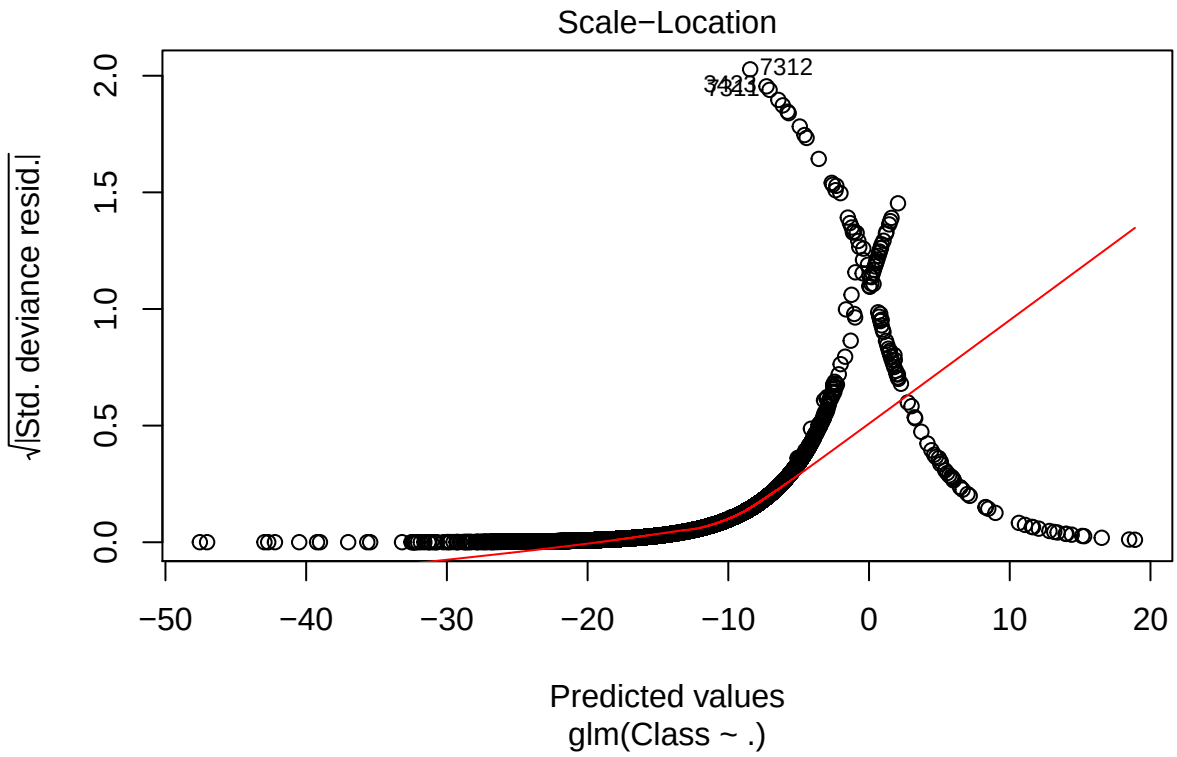
```
fitted.results.cat <- ifelse(glm.predict > 0.5,"1","0")

fitted.results.cat<-as.factor(fitted.results.cat)
confusionMatrix(testdata$Class,fitted.results.cat)
```

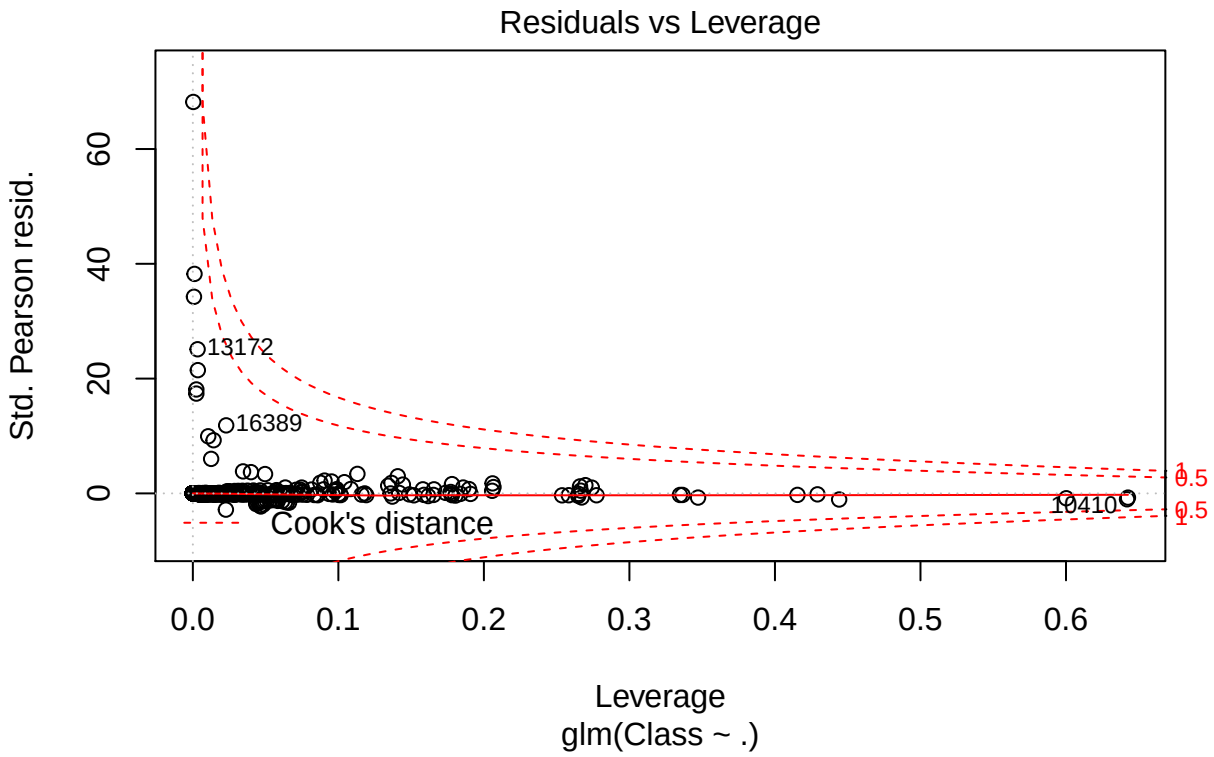
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 15011    14
##          1    13    32
##
##                Accuracy : 0.9982
##                  95% CI : (0.9974, 0.9988)
##     No Information Rate : 0.9969
##     P-Value [Acc > NIR] : 0.001708
##
##                   Kappa : 0.7024
##
##  Mcnemar's Test P-Value : 1.000000
##
##             Sensitivity : 0.9991
##             Specificity : 0.6957
##          Pos Pred Value : 0.9991
##          Neg Pred Value : 0.7111
##              Prevalence : 0.9969
##          Detection Rate : 0.9961
##    Detection Prevalence : 0.9970
##       Balanced Accuracy : 0.8474
##
##        'Positive' Class : 0
##
```

```
plot(glm.model)
```


Residuals vs Fitted
7312
3423
7311
Residuals
4
3
2
1
0
−1
−2
−50
−40
−30
−20
−10
0
10
20
Predicted values
glm(Class ~ .)


Normal Q−Q
Std. deviance resid.
Theoretical Quantiles
glm(Class ~ .)
7312
7423
7311
−2
−1
0
1
2
3
4
−4
−2
0
2
4

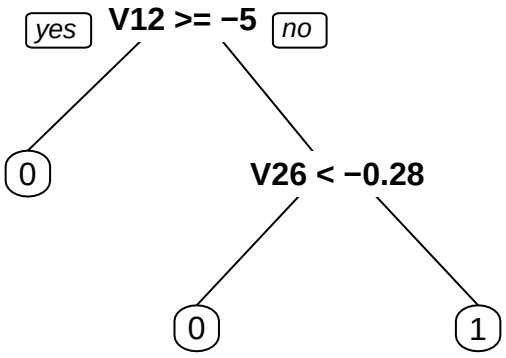

Scale−Location
7312
342
7311
√|Std. deviance resid.|
0.0
0.5
1.0
1.5
2.0
−50
−40
−30
−20
−10
0
10
20
Predicted values
glm(Class ~ .)




##99.82 % accuracy using logistic regression model.

##Decision tree model

```
tree.model <- rpart(Class ~ ., data = traindata, method = "class", minbucket = 20)
prp(tree.model)
```




```
tree.predict <- predict(tree.model, testdata, type = "class")
confusionMatrix(testdata$Class, tree.predict)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 15022     3
##          1    14    31
##
##                Accuracy : 0.9989
##                  95% CI : (0.9982, 0.9993)
##     No Information Rate : 0.9977
##     P-Value [Acc > NIR] : 0.0009861
##
##                   Kappa : 0.7843
##
##  Mcnemar's Test P-Value : 0.0152934
##
##             Sensitivity : 0.9991
##             Specificity : 0.9118
##          Pos Pred Value : 0.9998
##          Neg Pred Value : 0.6889
##              Prevalence : 0.9977
##          Detection Rate : 0.9968
```

```
##    Detection Prevalence : 0.9970
##       Balanced Accuracy : 0.9554
##
##        'Positive' Class : 0
##
```

```
#plot(tree.model)
```

##99.89 % accuracy (best) using decision tree. ## Noticed 0.07 % rise in accuracy so far let us analyze using only part of data that is 10000 records along with fraudulent transactions(149 records) this will result in well balanced data with 10149 records

Now we only keep 10000 rows of data with class = 0 Below code provides counts of non fraudulent and count of fraudulents records

```
data_nofraud <- subset(datacard, datacard$Class == 0)
data_fraud <- subset(datacard, datacard$Class == 1)
nrow(data_nofraud)
```

```
## [1] 50083
```

```
nrow(data_fraud)
```

```
## [1] 149
```

```
data_nofraud <- data_nofraud[1:10000, ]
nrow(data_nofraud)
```

```
## [1] 10000
```

```
data_all <- rbind(data_nofraud, data_fraud)
nrow(data_all)
```

```
## [1] 10149
```

##split data 70:30 , and find accuracy for different models

```
set.seed(1)
data_split <- sample.split(data_all$Class, SplitRatio = 0.7)
traindata <- subset(data_all, data_split == T)
testdata <- subset(data_all, data_split == F)

table(testdata$Class)
```

```
##
##    0    1
## 3000   45
```

baseline accuracy –> 95.298602 %

## logistic regression

```
glm.model <- glm(Class ~ ., data = traindata, family = "binomial", control = list(maxit = 50))
glm.predict <- predict(glm.model, testdata, type = "response")
table(testdata$Class, glm.predict > 0.5)
```

```
##
##     FALSE TRUE
##   0  2996    4
##   1     2   43
```

```
fitted.results.cat <- ifelse(glm.predict > 0.5,"1","0")

fitted.results.cat<-as.factor(fitted.results.cat)
confusionMatrix(testdata$Class,fitted.results.cat)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2996    4
##          1    2   43
##
##                Accuracy : 0.998
##                  95% CI : (0.9957, 0.9993)
##     No Information Rate : 0.9846
##     P-Value [Acc > NIR] : 5.008e-14
##
##                   Kappa : 0.9338
##
##  Mcnemar's Test P-Value : 0.6831
##
##             Sensitivity : 0.9993
##             Specificity : 0.9149
##          Pos Pred Value : 0.9987
##          Neg Pred Value : 0.9556
##              Prevalence : 0.9846
##          Detection Rate : 0.9839
##    Detection Prevalence : 0.9852
##       Balanced Accuracy : 0.9571
##
##        'Positive' Class : 0
##
```

**99.8 % accuracy using a 10149 records**

## SVM model

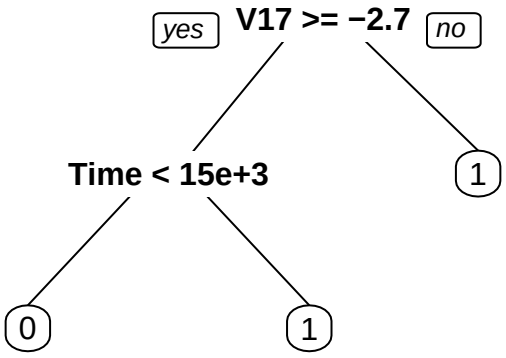
```
svm.model <- svm(Class ~ ., data = traindata, kernel = "radial", cost = 1, gamma = 0.1)
svm.predict <- predict(svm.model, testdata)
confusionMatrix(testdata$Class, svm.predict)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3000    0
##          1   24   21
##
##                Accuracy : 0.9921
##                  95% CI : (0.9883, 0.9949)
##     No Information Rate : 0.9931
##     P-Value [Acc > NIR] : 0.7828
##
##                   Kappa : 0.6329
##
##  Mcnemar's Test P-Value : 2.668e-06
##
##             Sensitivity : 0.9921
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.4667
##              Prevalence : 0.9931
##          Detection Rate : 0.9852
##    Detection Prevalence : 0.9852
##       Balanced Accuracy : 0.9960
##
##        'Positive' Class : 0
##
```

**99.21 % accuracy , not much improvement with svm**

## Decision Tree Model

```
tree.model <- rpart(Class ~ ., data = traindata, method = "class", minbucket = 20)
prp(tree.model)
```




```
tree.predict <- predict(tree.model, testdata, type = "class")
confusionMatrix(testdata$Class, tree.predict)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2994    6
##          1    4   41
##
##                Accuracy : 0.9967
##                  95% CI : (0.994, 0.9984)
##     No Information Rate : 0.9846
##     P-Value [Acc > NIR] : 5.635e-11
##
##                   Kappa : 0.8896
##
##  Mcnemar's Test P-Value : 0.7518
##
##             Sensitivity : 0.9987
##             Specificity : 0.8723
##          Pos Pred Value : 0.9980
##          Neg Pred Value : 0.9111
##              Prevalence : 0.9846
##          Detection Rate : 0.9833
```

```
##    Detection Prevalence : 0.9852
##       Balanced Accuracy : 0.9355
##
##        'Positive' Class : 0
##
```

```
#plot(tree.model)
```

**99.67 % accuracy !!**

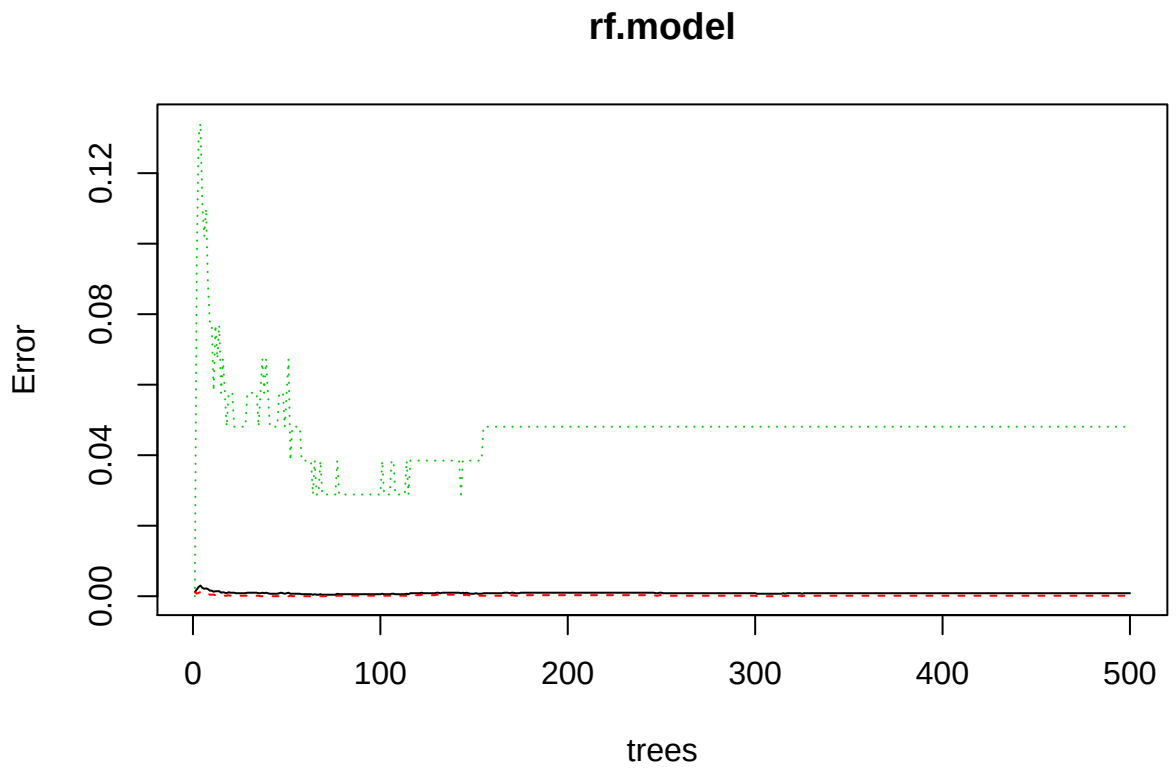
## Let's try random forest as well..

## random forest model

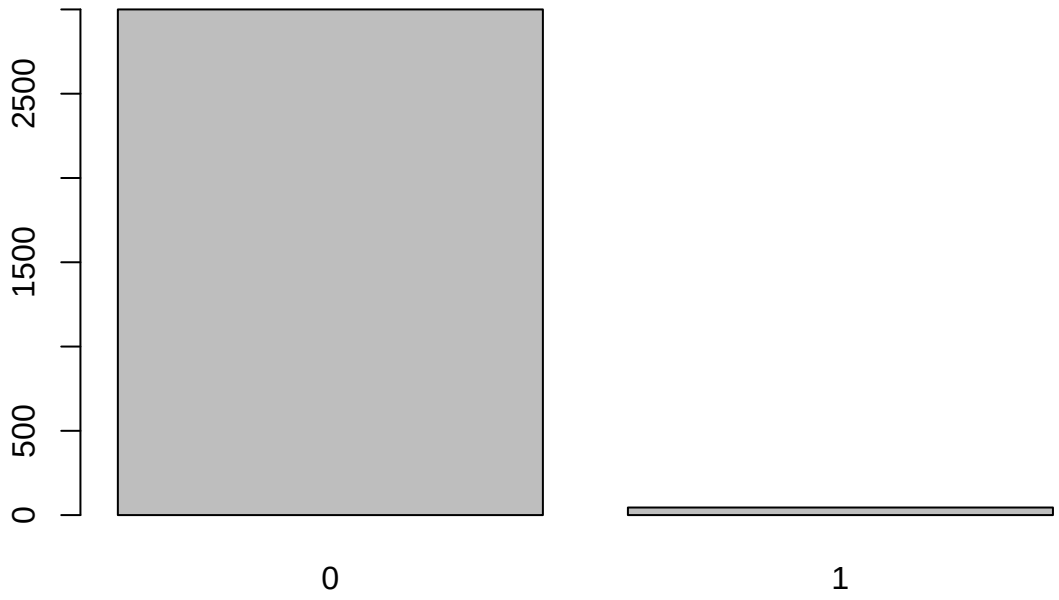
```
set.seed(10)
rf.model <- randomForest(Class ~ ., data = traindata,
                         ntree = 500, nodesize = 20)

rf.predict <- predict(rf.model, testdata)
confusionMatrix(testdata$Class, rf.predict)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3000    0
##          1    0   45
##
##                Accuracy : 1
##                  95% CI : (0.9988, 1)
##     No Information Rate : 0.9852
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.9852
##          Detection Rate : 0.9852
##    Detection Prevalence : 0.9852
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

```
#minimal error noticed below
plot(rf.model)
```




```
#used to understand the volume of fraudulent transactions(value=1)
plot(rf.predict)
```




**Random forest gives 100 % accuracy performed well compared to other models**

## Results

As per analysis using above models , noticed that using logistic regression it didnt fetch expected results due to comparatively less fraudulent data. However noticed better accuracy with good sample of well balanced data making sure that test and training sets are balanced well with mix of non fraudulent and fraudulent data.Accuracy is above 98% for most of models however random forest has provided high accuracy before and after 10k split of data.

## Conclusion

Brief Summary: This algorithm utilises 50000 before 10k split and 10149 transactions after split from 0.2 million records in actual dataset. This algorithm provides an extreme efficiency by using Randomforest model and decission tree model(before 10k split) among other models logistic regression, SVM model etc.

Limitations and Future work: However there are few limitations to this algorithm in terms of balanced and over sampling. This data can be further balanced such that fraudulant transactions ratio to non fraudulent transactions is maintained in balance. As we are not completely using entire dataset here , different data samples and well balanced datasets can be studied for further analysis with a close study on different fraud trends like elapsed times between transactions at different rates.

However due to datasize limitations actual dataset was trimmed to work for this model. Incase if there are no size restrictions we can train model more effectively and under stand other bias variables if the data

variables didnt have any privacy restrictions.

we can include a new variables like “time elapsed” or location risk or bulk amount transaction flags that will provide possible occurance of fraud on flagged accounts or suspected accounts can be analysed further.

Above variables will be further enhancements to this model if those values were made available in dataset with out privacy restrictions.