

HarvardX: PH125.9x Data Science

Choose your own Project

Subrahmanyam Aryasomayajula

October 9, 2022

Introduction

Credit Card Fraud detection project is done as a part of Choose your Own (CYO) Data Science Project , HarvardX: PH125.9x. In the project we have obtained creditcard.csv data from kaggle dataset . The dataset has been hosted in github along with other project files in order to facilitate easy execution of the code .The data has been inspected and machine learning models have been applied to evaluate them for the accuracy of predicting the faults in the credit card data set .We are trying to identify the machine learning model which gives the best accuracy out of the models we applied .This course also refers to and draws in on the knowledge developed as a part of “Data Science professional certificate program” by Harvardx.

Overview

The aim of this project is to evaluate K-Nearest Neighbors , Naive Bayes classifying algorithms, Random Forest algorithm and determine which algorithm results in highest accuracy in detecting the fraud in credit card transactions. Models have been trained and tested on the the Credit Card Fraud Detection data set to discover the highest accuracy model between the three models. The credit card transactions data set obtained from the kaggle has been randomly sampled to obtain a subset of the data . The data has been further divided into training and test data sets to build a training model and test the data . Confusion matrix has been generated in testing phase to obtain the metrics to evaluate the accuracy of the models.

Executive Summmary

The goal of the project is to apply each of the three models K-Nearest Neighbours , Naive Bayes classifying algorithms, Random Forest on the Credit Card Fraud Detection dataset and calculate the confusion matrix from which we can determine the accuracy of each of the three models .

confusion matrix measures the following four values:

```
tab <- matrix(c('True Negative (TN)', 'False Negative (FN)', 'False Positive (FP)', 'True Positive (TP)'),  
  colnames(tab) <- c('Actually Negative(0)', 'Actually Positive (1)')  
  rownames(tab) <- c('Predicted Negative (0)', 'Predicted Positive (1)')  
  tab <- as.table(tab)  
  tab
```

```
##                Actually Negative(0)  Actually Positive (1)  
## Predicted Negative (0) True Negative (TN)  False Negative (FN)  
## Predicted Positive (1) False Positive (FP)  True Positive (TP)
```

Total number of True Negative (TN) are the transaction detected as not fraudulent but in reality are also not fraudulent. Total number of False Negative (FN) are the transactions detected as not fraudulent but in reality are fraudulent . Total number of False Positive (FP) are the transaction detected as fraudulent but are not fraudulent in reality . Total number of True Positives (TP) are the transaction detected as fraudulent and really are fraudulent .

```
if(!require(caret)) install.packages("caret")
if(!require(class)) install.packages("class")
if(!require(e1071)) install.packages("e1071")
if(!require(ROCR)) install.packages("ROCR")
if(!require(dplyr)) install.packages("dplyr")

if(!require(ggplot2)) install.packages("ggplot2")
if(!require(readr)) install.packages("readr")
if(!require(pROC)) install.packages("pROC")
if(!require(randomForest)) install.packages("randomForest")
if(!require(corrplot)) install.packages("corrplot")
if(!require(data.table)) install.packages("data.table")
if(!require(rpart)) install.packages("rpart")
if(!require(stringr)) install.packages("stringr")
if(!require(tidyverse)) install.packages("tidyverse")

library(caret)
library(class)
library(e1071)
library(ROCR)
library(dplyr)

library(ggplot2)
library(readr)
library(pROC)
library(randomForest)
library(corrplot)

library(data.table)
library(rpart)
library(stringr)
library(rmarkdown)
library(tinytex)
library(knitr)

tinytex::install_tinytex(force = TRUE)

dir = getwd()

download.file("https://raw.githubusercontent.com/SubraArya/creditcardfraud/main/creditcard.csv", "./creditcard.csv")

filename = paste(dir , "/creditcard.csv", sep = "")

filename = str_replace_all(filename , "/", "\\")
credit <- read.csv(filename)
```

#Methods Data Exploration and analysis

#Basic structure of the record such as column names , types and sample records examined.

```
str(credit)
```

```
## 'data.frame': 284807 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 : int 0 0 0 0 0 0 0 0 0 0 ...
```

#Dimensions of the data frame i.e total number of rows and columns .

```
dim(credit)
```

```
## [1] 284807 31
```

factor transformation was on variable class .:

```
credit$Class <- factor(credit$Class)
```

Summary stats on every columns on the dataframe .

```
summary(credit)
```

```
##      Time      V1      V2      V3
## Min.   :    0  Min.  :-56.40751  Min.  :-72.71573  Min.  :-48.3256
## 1st Qu.: 54202  1st Qu.: -0.92037  1st Qu.: -0.59855  1st Qu.: -0.8904
## Median : 84692  Median :  0.01811  Median :  0.06549  Median :  0.1799
## Mean   : 94814  Mean   :  0.00000  Mean   :  0.00000  Mean   :  0.0000
## 3rd Qu.:139321  3rd Qu.:  1.31564  3rd Qu.:  0.80372  3rd Qu.:  1.0272
## Max.   :172792  Max.   :  2.45493  Max.   : 22.05773  Max.   :  9.3826
##      V4      V5      V6      V7
## Min.  :-5.68317  Min.  :-113.74331  Min.  :-26.1605  Min.  :-43.5572
## 1st Qu.: -0.84864  1st Qu.: -0.69160  1st Qu.: -0.7683  1st Qu.: -0.5541
## Median : -0.01985  Median : -0.05434  Median : -0.2742  Median :  0.0401
## Mean   : 0.00000  Mean   :  0.00000  Mean   :  0.0000  Mean   :  0.0000
## 3rd Qu.: 0.74334  3rd Qu.:  0.61193  3rd Qu.:  0.3986  3rd Qu.:  0.5704
## Max.   :16.87534  Max.   : 34.80167  Max.   : 73.3016  Max.   :120.5895
##      V8      V9      V10     V11
## Min.  :-73.21672  Min.  :-13.43407  Min.  :-24.58826  Min.  :-4.79747
## 1st Qu.: -0.20863  1st Qu.: -0.64310  1st Qu.: -0.53543  1st Qu.: -0.76249
## Median :  0.02236  Median : -0.05143  Median : -0.09292  Median : -0.03276
## Mean   :  0.00000  Mean   :  0.00000  Mean   :  0.00000  Mean   :  0.00000
## 3rd Qu.:  0.32735  3rd Qu.:  0.59714  3rd Qu.:  0.45392  3rd Qu.:  0.73959
## Max.   : 20.00721  Max.   : 15.59500  Max.   : 23.74514  Max.   :12.01891
##      V12     V13     V14     V15
## Min.  :-18.6837  Min.  :-5.79188  Min.  :-19.2143  Min.  :-4.49894
## 1st Qu.: -0.4056  1st Qu.: -0.64854  1st Qu.: -0.4256  1st Qu.: -0.58288
## Median :  0.1400  Median : -0.01357  Median :  0.0506  Median :  0.04807
## Mean   :  0.0000  Mean   :  0.00000  Mean   :  0.0000  Mean   :  0.00000
## 3rd Qu.:  0.6182  3rd Qu.:  0.66251  3rd Qu.:  0.4931  3rd Qu.:  0.64882
## Max.   :  7.8484  Max.   :  7.12688  Max.   : 10.5268  Max.   :  8.87774
##      V16     V17     V18
## Min.  :-14.12985  Min.  :-25.16280  Min.  :-9.498746
## 1st Qu.: -0.46804  1st Qu.: -0.48375  1st Qu.: -0.498850
## Median :  0.06641  Median : -0.06568  Median : -0.003636
## Mean   :  0.00000  Mean   :  0.00000  Mean   :  0.000000
## 3rd Qu.:  0.52330  3rd Qu.:  0.39968  3rd Qu.:  0.500807
## Max.   : 17.31511  Max.   :  9.25353  Max.   :  5.041069
##      V19     V20     V21
## Min.  :-7.213527  Min.  :-54.49772  Min.  :-34.83038
## 1st Qu.: -0.456299  1st Qu.: -0.21172  1st Qu.: -0.22839
## Median :  0.003735  Median : -0.06248  Median : -0.02945
## Mean   :  0.000000  Mean   :  0.00000  Mean   :  0.00000
## 3rd Qu.:  0.458949  3rd Qu.:  0.13304  3rd Qu.:  0.18638
## Max.   :  5.591971  Max.   : 39.42090  Max.   : 27.20284
##      V22     V23     V24
## Min.  :-10.933144  Min.  :-44.80774  Min.  :-2.83663
## 1st Qu.: -0.542350  1st Qu.: -0.16185  1st Qu.: -0.35459
## Median :  0.006782  Median : -0.01119  Median :  0.04098
## Mean   :  0.000000  Mean   :  0.00000  Mean   :  0.00000
## 3rd Qu.:  0.528554  3rd Qu.:  0.14764  3rd Qu.:  0.43953
```

```
## Max. : 10.503090 Max. : 22.52841 Max. : 4.58455
## V25 V26 V27
## Min. :-10.29540 Min. :-2.60455 Min. :-22.565679
## 1st Qu.: -0.31715 1st Qu.: -0.32698 1st Qu.: -0.070840
## Median : 0.01659 Median : -0.05214 Median : 0.001342
## Mean : 0.00000 Mean : 0.00000 Mean : 0.000000
## 3rd Qu.: 0.35072 3rd Qu.: 0.24095 3rd Qu.: 0.091045
## Max. : 7.51959 Max. : 3.51735 Max. : 31.612198
## V28 Amount Class
## Min. :-15.43008 Min. : 0.00 0:284315
## 1st Qu.: -0.05296 1st Qu.: 5.60 1: 492
## Median : 0.01124 Median : 22.00
## Mean : 0.00000 Mean : 88.35
## 3rd Qu.: 0.07828 3rd Qu.: 77.17
## Max. : 33.84781 Max. : 25691.16
```

Top 5 records

```
head(credit)
```

```
## Time V1 V2 V3 V4 V5 V6
## 1 0 -1.3598071 -0.07278117 2.5363467 1.3781552 -0.33832077 0.46238778
## 2 0 1.1918571 0.26615071 0.1664801 0.4481541 0.06001765 -0.08236081
## 3 1 -1.3583541 -1.34016307 1.7732093 0.3797796 -0.50319813 1.80049938
## 4 1 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888 1.24720317
## 5 2 -1.1582331 0.87773676 1.5487178 0.4030339 -0.40719338 0.09592146
## 6 2 -0.4259659 0.96052304 1.1411093 -0.1682521 0.42098688 -0.02972755
## V7 V8 V9 V10 V11 V12
## 1 0.23959855 0.09869790 0.3637870 0.09079417 -0.5515995 -0.61780086
## 2 -0.07880298 0.08510165 -0.2554251 -0.16697441 1.6127267 1.06523531
## 3 0.79146096 0.24767579 -1.5146543 0.20764287 0.6245015 0.06608369
## 4 0.23760894 0.37743587 -1.3870241 -0.05495192 -0.2264873 0.17822823
## 5 0.59294075 -0.27053268 0.8177393 0.75307443 -0.8228429 0.53819555
## 6 0.47620095 0.26031433 -0.5686714 -0.37140720 1.3412620 0.35989384
## V13 V14 V15 V16 V17 V18
## 1 -0.9913898 -0.3111694 1.4681770 -0.4704005 0.20797124 0.02579058
## 2 0.4890950 -0.1437723 0.6355581 0.4639170 -0.11480466 -0.18336127
## 3 0.7172927 -0.1659459 2.3458649 -2.8900832 1.10996938 -0.12135931
## 4 0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.68409279 1.96577500
## 5 1.3458516 -1.1196698 0.1751211 -0.4514492 -0.23703324 -0.03819479
## 6 -0.3580907 -0.1371337 0.5176168 0.4017259 -0.05813282 0.06865315
## V19 V20 V21 V22 V23 V24
## 1 0.40399296 0.25141210 -0.018306778 0.277837576 -0.11047391 0.06692808
## 2 -0.14578304 -0.06908314 -0.225775248 -0.638671953 0.10128802 -0.33984648
## 3 -2.26185709 0.52497973 0.247998153 0.771679402 0.90941226 -0.68928096
## 4 -1.23262197 -0.20803778 -0.108300452 0.005273597 -0.19032052 -1.17557533
## 5 0.80348692 0.40854236 -0.009430697 0.798278495 -0.13745808 0.14126698
## 6 -0.03319379 0.08496767 -0.208253515 -0.559824796 -0.02639767 -0.37142658
## V25 V26 V27 V28 Amount Class
## 1 0.1285394 -0.1891148 0.133558377 -0.02105305 149.62 0
## 2 0.1671704 0.1258945 -0.008983099 0.01472417 2.69 0
```

```
## 3 -0.3276418 -0.1390966 -0.055352794 -0.05975184 378.66 0
## 4 0.6473760 -0.2219288 0.062722849 0.06145763 123.50 0
## 5 -0.2060096 0.5022922 0.219422230 0.21515315 69.99 0
## 6 -0.2327938 0.1059148 0.253844225 0.08108026 3.67 0
```

Methods Insights into Total number of Fraud and Non-Fraud Rows in data

```
rowsTotal <- nrow(credit)
fraudRowsTotal <- nrow(credit[credit$Class == 1,])
nonFraudRowsTotal <- rowsTotal - fraudRowsTotal

fraudRowsTotal
```

```
## [1] 492
```

```
nonFraudRowsTotal
```

```
## [1] 284315
```

```
rowsTotal
```

```
## [1] 284807
```

- There is high skewness in the data. The number of fraudulent transactions are very less as compared to non fraudulent (good) transactions, comprising of only 492 frauds out of 284807 transactions (0.1727486% of the data set). The skewness in the data is expected as the number of fraudulent transactions are generally less compared to good transactions.
- The data set consists of numerical values of 28 PCA transformed features, V1 - V28 , time , amount and class fields. Further, no metadata about the original features is provided. class fields with value 0 are non fraudulent (good) transactions and class fields with value 1 are fraudulent transactions.

Methods , Dataset preparation

The data set is large to execute in reasonable time on a 16b ram , Intel 7th generation laptop . Hence 20% of the data was randomly sampled. Training and test data sets were generated from randomized data. Training data Set is 75% of sample Test data set is 25% of sample

set seed for random sampling and take 20% sample data approximately.

```
set.seed(2000)

samp <- sample(1:nrow(credit), round(0.2*nrow(credit)))

credit <- credit[samp, ]
```

Total number of records in the sample 56961

```
nrow(credit)
```

```
## [1] 56961
```

Partition the sample data into training and test data sets. Training data Set 75% of sample and Test data set 25% of sample.

```
index <- createDataPartition(credit$Class, p = 0.75, list = F)
```

```
train <- credit[index, ]
```

```
nrow(train)
```

```
## [1] 42722
```

Total number of records in Training data Set in the sample is 42722 which is 75% of sample .

```
test <- credit[-index, ]  
nrow(test)
```

```
## [1] 14239
```

Total number of records in Test data Set in the sample is 14239 which is 25% of sample .

Methods

WE have applied K-Nearest Neighbors ,Naive Bayes , Random Forest algorithms on credit card dataset . Models have been trained and tested on the the Credit Card Fraud Detection data set to discover the highest accuracy model between the three models.

K-Nearest Neighbors

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. It can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another. For us to determine which data points are closest to a given query point, the distance between the query point and the other data points need to be calculated. These distance metrics help to form decision boundaries, which partitions query points into different regions. Euclidean distance, Manhattan distance, Minkowski distance ,Hamming distance are various ways of measuring the distance.

The k value in the k-NN algorithm defines how many neighbors will be checked to determine the classification of a specific query point. For example, if k=1, the instance will be assigned to the same class as its single nearest neighbor. Defining k can be a balancing act as different values can lead to overfitting or underfitting. Lower values of k can have high variance, but low bias, and larger values of k may lead to high bias and lower variance. The choice of k will largely depend on the input data as data with more outliers or noise will likely perform better with higher values of k. odd number for k is preferred to avoid ties in classification, and cross-validation techniques can help us choose the optimal k for our dataset.

In our credit card fraud detection data set ,as all the variables were of class either “numeric” or “integer” , The knn classification with the number of neighbours was set to 5 as a default.

```
knn1 <- knn(train = train[,-31], test = test[,-31], cl = train$Class, k = 5)

cmknn <- confusionMatrix(knn1, test$Class, positive = "1")
cmknn
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##           0 14215     24
##           1      0      0
##
##           Accuracy : 0.9983
##           95% CI : (0.9975, 0.9989)
##       No Information Rate : 0.9983
##       P-Value [Acc > NIR] : 0.554
##
##           Kappa : 0
##
##  McNemar's Test P-Value : 2.668e-06
##
##           Sensitivity : 0.000000
##           Specificity : 1.000000
##       Pos Pred Value :      NaN
##       Neg Pred Value : 0.998314
##           Prevalence : 0.001686
##       Detection Rate : 0.000000
##       Detection Prevalence : 0.000000
##       Balanced Accuracy : 0.500000
##
##       'Positive' Class : 1
##
```

Naive Bayes

Naive Bayes is a simple technique for constructing classifier ie models that assign class labels to problem instances, it is represented as vectors of feature values, where the class labels are drawn from a finite set. It is a family of algorithms training such classifiers , algorithms are based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

The Naive Bayes classification algorithm is a probabilistic classifier. It is based on probability models that incorporate strong independence assumptions. The independence assumptions often do not have an impact on reality that is why they are considered as naive.

We can derive probability models by using Bayes' theorem . Depending on the nature of the probability model, we can train the Naive Bayes algorithm in a supervised learning setting.

A Naive Bayes model consists of a large cube that includes the following dimensions: Input field value for discrete fields, or input field value range for continuous fields. Continuous fields are divided into discrete bins by the Naive Bayes algorithm.

Target field value ,means that a Naive Bayes model records how often a target field value appears together with a value of an input field.

Naive Bayes model based analysis of the data set to obtain the confusion matrix for fault detection.

The model was to adjust for the possibility of experiencing posterior class probability of “0” by “laplace = 1”.

```
bayes <- naiveBayes(Class~., data = train, laplace = 1)
bayes$sapriori
```

```
## Y
##      0      1
## 42647    75
```

```
pred <- predict(bayes, test)
cmnb <- confusionMatrix(pred, test$Class, positive = "1")
cmnb
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction      0      1
##              0 13948      7
##              1   267     17
##
##              Accuracy : 0.9808
##              95% CI : (0.9784, 0.9829)
##      No Information Rate : 0.9983
##      P-Value [Acc > NIR] : 1
##
##              Kappa : 0.1076
##
##  Mcnemar's Test P-Value : <2e-16
##
##              Sensitivity : 0.708333
##              Specificity : 0.981217
##              Pos Pred Value : 0.059859
##              Neg Pred Value : 0.999498
##              Prevalence : 0.001686
##              Detection Rate : 0.001194
##      Detection Prevalence : 0.019945
##              Balanced Accuracy : 0.844775
##
##              'Positive' Class : 1
##
```

Build the Model with the Random Forest with decision trees set to 40.

Random forest consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction. The fundamental idea behind random forests is that a large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models

. The low correlation between models is the key .uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason for this positive effect is that the trees protect each other from their individual errors . While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction.

For our credit card data set we have used number of decision trees to 40.

```
random_model_train <- randomForest(Class ~ ., data = train, ntree = 40)

random_pred_test <- predict(random_model_train, test)

cmrf <- confusionMatrix(random_pred_test, test$Class, positive = "1")
cmrf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction      0      1
##           0 14215    11
##           1      0    13
##
##           Accuracy : 0.9992
##           95% CI : (0.9986, 0.9996)
##       No Information Rate : 0.9983
##       P-Value [Acc > NIR] : 0.002508
##
##           Kappa : 0.7024
##
##  Mcnemar's Test P-Value : 0.002569
##
##           Sensitivity : 0.541667
##           Specificity : 1.000000
##       Pos Pred Value : 1.000000
##       Neg Pred Value : 0.999227
##           Prevalence : 0.001686
##       Detection Rate : 0.000913
##       Detection Prevalence : 0.000913
##       Balanced Accuracy : 0.770833
##
##       'Positive' Class : 1
##
```

Results

K-Nearest Neighbors

In Model the number of nearest neighbors was set to 5. 99.8314488% Accuracy in prediction was achieved as per the above confusion matrix . Out of a total 14239 test cases , Total number of True Negative (TN) are 14215 the transaction detected as not fraudulent but in reality are also not fraudulent. Total number of False Negative (FN) are 24 the transactions detected as not fraudulent but in reality are fraudulent . Total number of False Positive (FP) are 0 the transaction detected as fraudulent but are not fraudulent in reality

. Total number of True Positives (TP) are 0 the transaction detected as fraudulent and really are fraudulent .

Although 99.8314488% accuracy was obtained with the specified index of $k = 5$, there are still some shortcomings as seen from “confusionMatrix” output .The model has not predicted any True Postive cases as illustrated by the class = 1 , prediction is 0 .

Naive Baiyes

98.0757076% Accuracy in prediction was achieved as per the above confusion matrix . Out of a total 14239 test cases ,

Total number of True Negative (TN) are 13948 the transaction detected as not fraudulent but in reality are also not fraudulent. Total number of False Negative (FN) are 7 the transactions detected as not fraudulent but in reality are fraudulent . Total number of False Positive (FP) are 267 the transaction detected as fradulent but are not fraudulent in reality . Total number of True Positives (TP) are 17 the transaction detected as fradulent and really are fraudulent .

Naive Bayes is under performing for this set .

Naive Bayes is an under performing algorithm in comparison with knn as we can see the accuracy is 98.0757076% with Naive Bayes Vs 99.8314488% for K Nearest neighbors .

Random forest

99.9227474% Accuracy in prediction was achieved as per the above confusion matrix . Out of a total 14239 test cases ,

Total number of True Negative (TN) are 14215 the transaction detected as not fraudulent but in reality are also not fraudulent. Total number of False Negative (FN) are 11 the transactions detected as not fraudulent but in reality are fraudulent . Total number of False Positive (FP) are 0 the transaction detected as fraudulent but are not fraudulent in reality . Total number of True Positives (TP) are 13 the transaction detected as fraudulent and really are fraudulent .

Over all Random forest is performing the best with 99.9227474% accuracy compared to Naive Bayes which is an under performing algorithm in comparison with knn . We can see the accuracy is 98.0757076% with Naive Bayes and then 99.8314488% for K Nearest neighbors.

Conclusion

In conclusion , with regards to creditcard data set, Random forest is better performing than K-Nearest Neighbors algorithm and Naive Bayes in the extraction of the most accurate predictions in terms of whether a credit card will be detected fraud or not.

Random forest has detected 13 True positives , in comparison to 17 from Naive Bayes , 0 from K nearest neighbors . Although we see Naive Bayes has detected 17 True positives , we should also consider false negatives and false positives to evaluate the accuracy of the algorithm . Random Forest having 0 False positives and 11 false negatives, vs Naive Bayes having 267 False positives and 7 False negatives , and K-Nearest Neighbors having 0 false positives and 24 false negatives .This is the reason for overall accuracy of the Random forest the best with 99.93 accuracy compared to Naive Bayes which is an under performing algorithm with 98.08 and then 99.83 for K Nearest neighbors.

However the performance of the Random forest on high volume datasets can be significantly slow as the number of decision Trees increase and needs further investigation.