# Anomaly Detection: KNN, XGBoost, Random Forest

Lillian Jarrett, Samuel Brown, Stephen Kullman

DS 6030 | Fall 2023

## Contents



## Contents

### 1. Introduction to Anomaly Detection

## Anomaly Detection

Anomaly detection is a significant problem.

- In December 2022, Nilson published a report estimating that credit card fraud will cost US consumers 166 billion dollars over the next 10 years, affecting every age group in every state.
- The Fair Credit Billing Act limits consumer liability in credit card fraud cases to just 50 dollars, provided the theft is reported and the correct steps are taken.
  - Credit card companies have a huge interest in quickly and accurately detecting anomalies. Understanding what an anomaly could actually mean, such as a credit card transaction amount significantly above expectations, is crucial.

#### Anomaly detection

The process of identifying data points that fall outside what is considered normal. It also can be called outlier detection or changepoint detection. These are all closely related.

- Anomalies don't have to be bad; there could be opportunities to investigate, like a salesperson or product selling more than predicted, leading to insights into improving the business.
- Anomalies don't always indicate bad intent; they could be caused by bad sensor data or a network error. However, sometimes, anomalies are related to fraud or crime.
- Anomaly detection has a long history rooted in traditional statistics, but the past several decades have seen companies using machine learning techniques to automate and improve these processes.
- In the case of credit card fraud, anomaly detection could be implemented with push notification can allow the credit card user to confirm or deny the fraudulent

**Supervised? Unsupervised?**

- Some datasets have labels, like credit card data that we can label fraudulent or normal. For example, a push notification on your phone about a suspicious charge allows you to label it normal or tell the credit card company it's fraudulent. This labeled data can be used to improve fraud detection using machine learning algorithms. We will focus on this type of learning in our examples.
- Most of the time, it'll involve using unlabeled data where unsupervised learning will be applied.

**Types of algorithms used**

- Density Based
  - Determines when an outlier differs from a larger, denser normal data set, using algorithms like K-nearest neighbor. This is what we will go over in these notes.
  - https://scikit-learn.org/stable/modules/outlier_detection.html

**Density Thresholds:** - Anomaly detection can be based on a density threshold

- How to evaluate it?
  - Reachability distance
  - Given the predictions of fraud and no fraud, maybe the best model is the one that reduces the cost to the business and/or customer. For example, the cost of not identifying a fraudulent transaction could be very high, while misidentifying a strange buy as fraudulent would cost nothing more than a phone notification.
  - From a model perspective, accuracy would only be useful in a balanced dataset, since the original dataset, we can just guess non fraudulent every time and get 99.98% accuracy
- Cluster Based
  - Shows when data points differ from a data cluster using K-means clustering techniques.
- Bayesian
  - Probabilistic modeling of events happening given a set of observations and recording when significant deviations occur.
- Neural Networks
  - Train a prediction time series model and flag deviations.

**Types of anomalies**

- Point anomalies
  - Far outside of the normal data range.
- Contextual anomalies
  - Deviations within specific contexts (like increased spending around Christmas time being taken into account).
- Collective anomalies
  - Groups of data points that collectively deviate significantly from the overall distribution of a dataset.
  - The difficulty of defining a hard and fast rule for what is abnormal (as it changes as fraudsters learn).
  - Algorithms need to be efficient enough to handle the scale of data and work fast enough to provide timely alerts to anomalies.
  - It takes a long time to generate enough data to provide a useful baseline for what is considered normal.
  - The boundary between normal and abnormal behavior is often not precise.
  - The definition of abnormal or normal may frequently change, as malicious adversaries constantly adapt themselves. Therefore, the threshold based on the moving average may not always apply.

## 2. Example Dataset: Credit Card Fraud Detection

https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud/data

```r
#load data
data <- read.csv("/Users/lilyjarrett/Desktop/MSDS/ds6030/Project/creditcard.csv")
```

- The Kaggle dataset predictors are already scaled, so we must transform the time and amount predictors.

```r
# set ggplot2 theme
ggplot2::theme_set(ggplot2::theme_bw())
# load packages
library(tidyverse)
library(dplyr)
library(ggplot2)
library(scales)
library(ROSE)
library(caTools)
library(caret)
library(PRROC)
library(randomForest)
library(pROC)
library(xgboost)
```

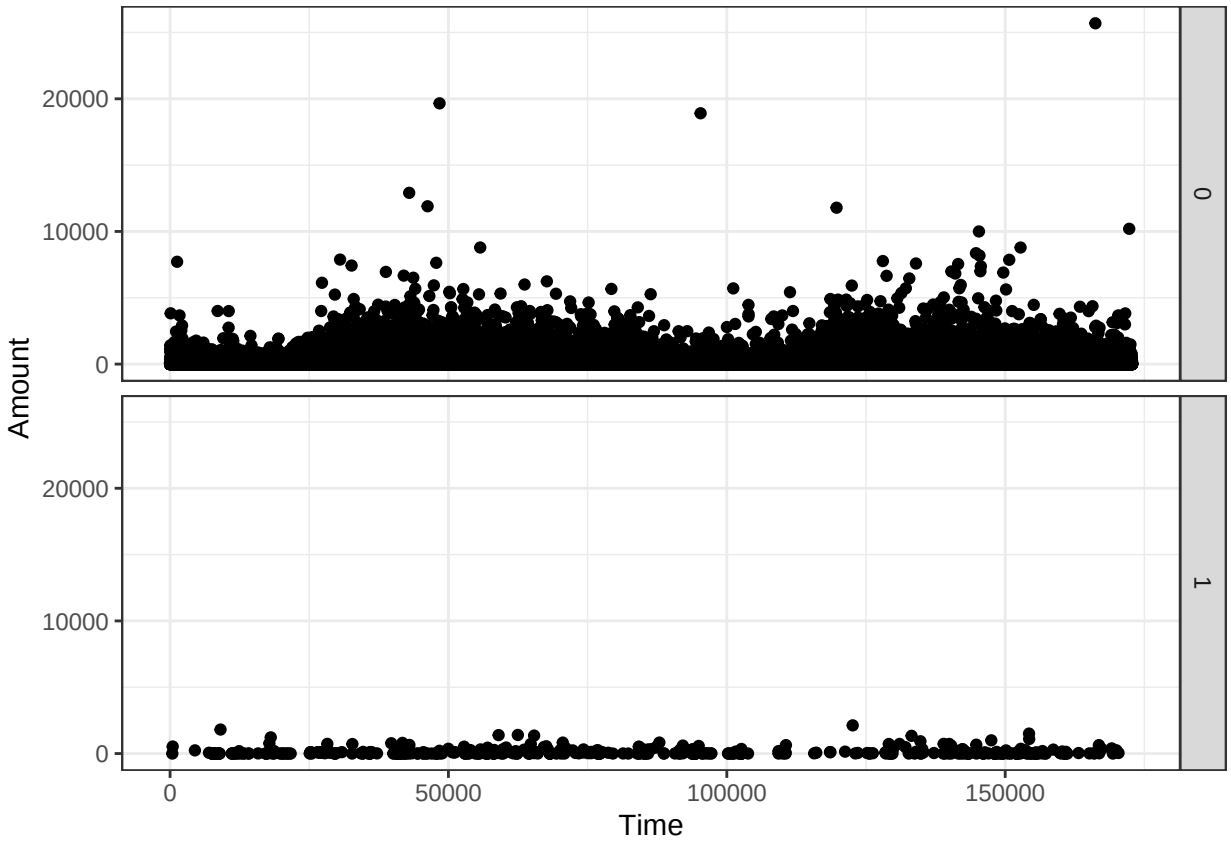
This data is unbalanced; we'll deal with this later

```r
data %>%
  group_by(Class) %>%
  tally()
```

```
## # A tibble: 2 x 2
##   Class      n
##   <int>  <int>
## 1     0 284315
## 2     1    492
```

Scatterplot: Amount v. Time

```r
ggplot(data = data, aes(x=Time,y=Amount))+
  geom_point()+
  facet_grid(rows = vars(Class))
```




Average Amount by Class:

```
data %>%
  group_by(Class) %>%
  summarize(avg_amt = mean(Amount))
```

```
## # A tibble: 2 x 2
##   Class avg_amt
##   <int>   <dbl>
## 1     0    88.3
## 2     1   122.
```

Correlation map:

```
# round(cor(data),2)
```

```
# Class as factor
data$Class <- as.factor(data$Class)
str(data$Class)
```

```
##  Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

```
# scale data
scaled <- data
scaled$Time <- scale(scaled$Time)
scaled$Amount <- scale(scaled$Amount)
```

Split into test/train

```
set.seed(1738)
ind = sample.split(scaled, SplitRatio = 0.8) # split ratio is 0.8.
```

```
train = scaled[ind, ]
test =  scaled[!ind, ]
```

Dealing with unbalanced data using undersampling approach

```
nrow_fraud <- nrow(train[train$Class == 1, ])

undersample_frac = 0.2

# find the desired sample size when applying the undersampling technique.
undersample_size <- nrow_fraud/undersample_frac

# undersample the dataset
undersample <- ovun.sample(Class ~ ., data = train, method = "under",
                           N = undersample_size, seed = 1738)

undersample_df <- undersample$data

print("Class Frequency before and after using undersampling techniques || 0: Not Fraud & 1: Fraud")
```

```
## [1] "Class Frequency before and after using undersampling techniques || 0: Not Fraud & 1: Fraud"
```

```
cbind(Before = table(train$Class), After = table(undersample_df$Class))
```

```
##   Before After
## 0 220125  1488
## 1    372   372
```

**How to deal with unbalanced data?**

- Undersample:
  - Decrease the size of the majority class
- Oversample:
  - Generates copies of minority class to balance
  - SMOTE
  - Since KNN does better with a lot of data, maybe we use oversampling as well

**What are tuning parameters (e.g. k, distance metric, missing data)**

- It's non-parametric, the only tuning parameter would be k, or the kind of distance used (manhattan, euclidian)

Now we will try three different supervised methods of anomaly detections:

## 3. KNN

- In order to make a prediction for an observation X = x, the training observations that are closest to x are identified. Then x is assigned to the class to which the majority of these observations belong. KNN is a completely non-parametric approach: no assumptions are made about the shape of the decision boundary.
- Because it's non-parametric, KNN is way better for decision boundaries that are non-linear
- KNN requires a lot of data relative to the number of predictors.
- Other classification techniques like QDA can be used if the sample size is smaller
- K needs to be odd, to avoid ties
- Drawbacks include the computational load, finding the neighbors and storing the entire training set.
  - Reducing storage requirements, the idea is to isolate the subset of the training set that is near the decision boundary.

```
knnModel <- train(
             Class ~ .,
             data = undersample_df,
                method = "knn",
             trControl = trainControl(method = "cv"),
             tuneGrid = data.frame(k = c(3,5,7,9,11,13))
            )
```

We find the optimal k = 11

```
knnModel$bestTune
```

```
##    k
## 5 11
```

```
best_model<- knn3(
                  Class ~ .,
                  data = undersample_df,
                  k = knnModel$bestTune$k
                 )
```

Making Predictions

```
predictions <- predict(best_model, test, type='class')
# Calculate confusion matrix
```

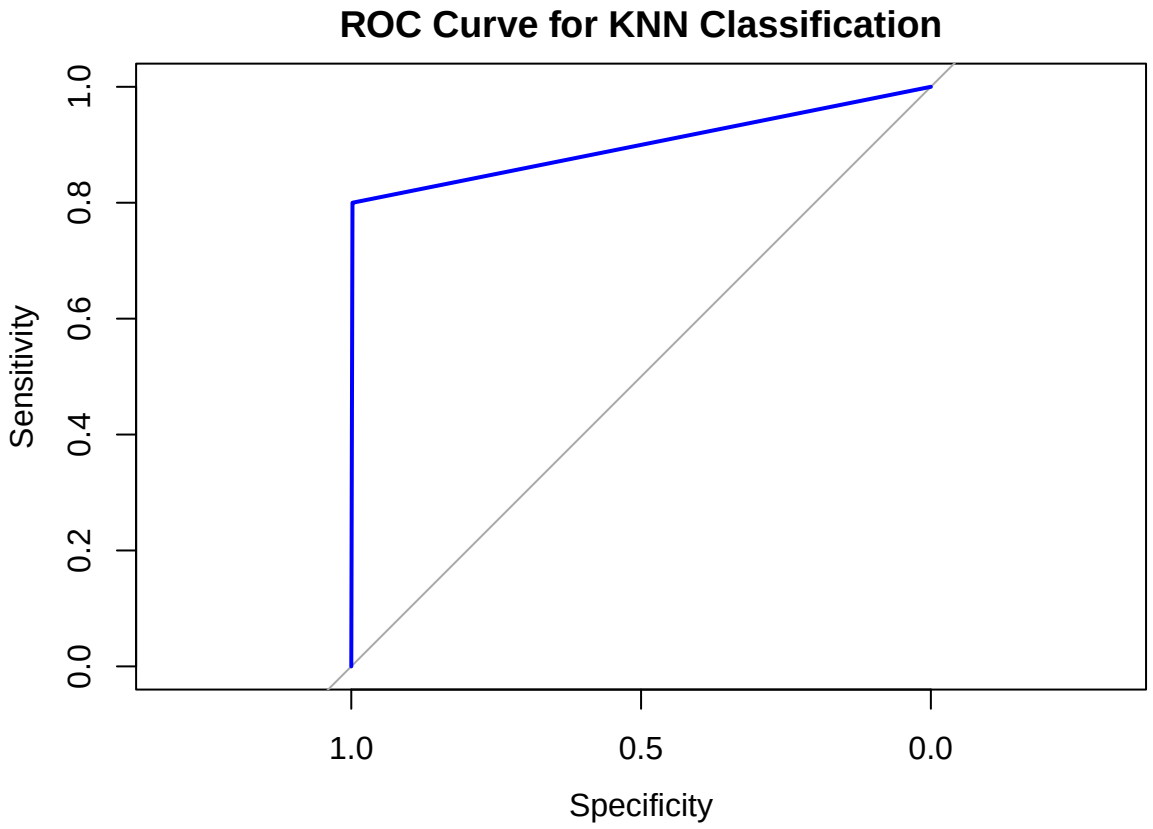
```
preds_numeric <- as.numeric(predictions)
```

```
roc_obj <- roc(test$Class, preds_numeric)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

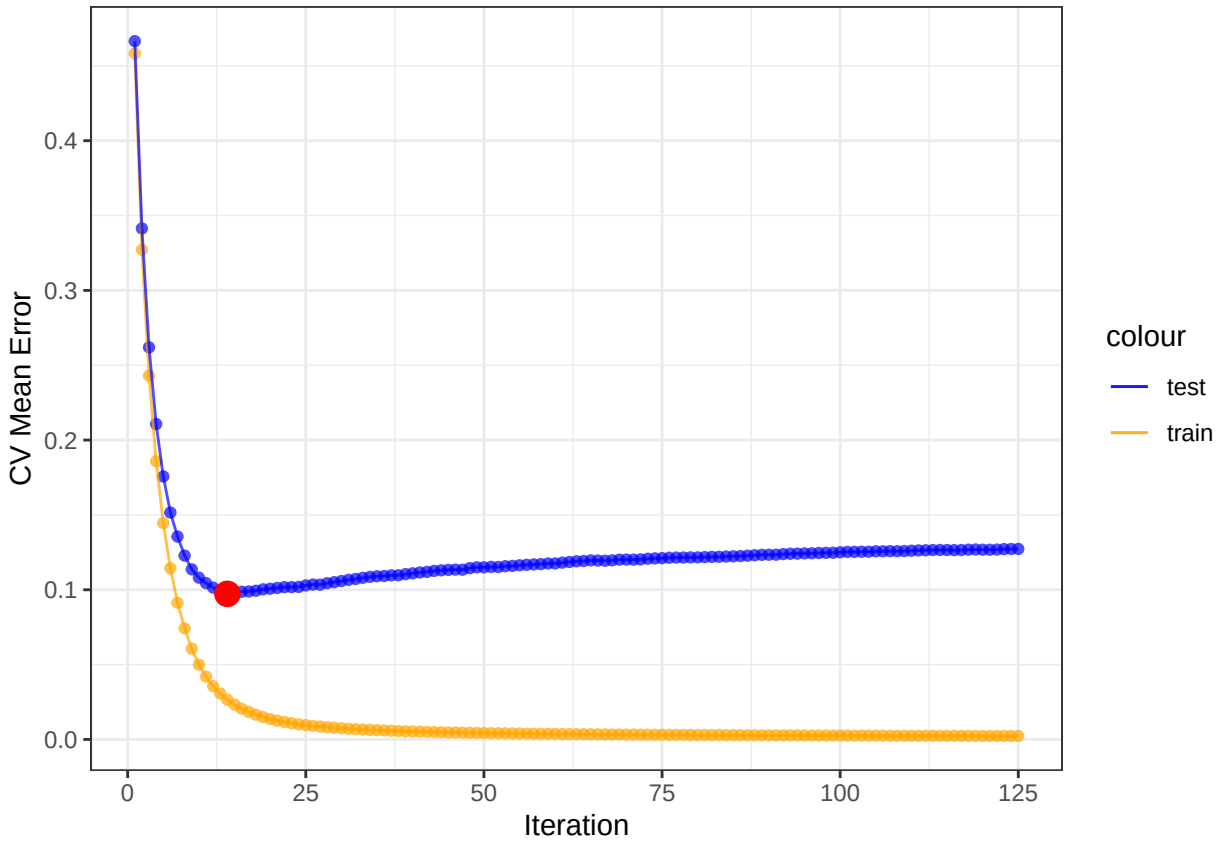
```
plot(roc_obj, main="ROC Curve for KNN Classification", col="blue")
```




## 4. XGBoost

Something like an XGBoost can also be used to classify anomalies. Here we use the undersampled training data to perform Cross-Validation with an XgBoost and find the optimal number of rounds:

```
set.seed(200)
# Get data ready to XGBoost
train_mat <- undersample_df %>%
  select(-Class) %>%
  data.matrix() %>%
  xgb.DMatrix(label = undersample_df %>% select(Class) %>% as.matrix())
# Run CV
train_boost <- xgb.cv(data = train_mat, objective = "binary:logistic",
                      nrounds = 125, nfold = 7, verbose = 0,
                      metrics = list('error', 'logloss'))
# Graph the performance
train_boost$evaluation_log %>%
  ggplot(aes(x = iter)) +
  geom_point(aes(y = train_logloss_mean), color = 'orange', alpha = .7) +
  geom_point(aes(y = test_logloss_mean), color = 'blue', alpha = .7) +
  geom_line(aes(y = train_logloss_mean, color = 'train'), alpha = .7) +
  geom_line(aes(y = test_logloss_mean, color = 'test'), alpha = .7) +
  geom_point(data = train_boost$evaluation_log %>% slice_min(test_logloss_mean) %>%
               slice_min(iter),
             aes(y = test_logloss_mean),
             color = 'red', size = 4) +
  labs(y = 'CV Mean Error', x = 'Iteration') +
  scale_color_manual(values = c('train' = 'orange', 'test' = 'blue'))
```

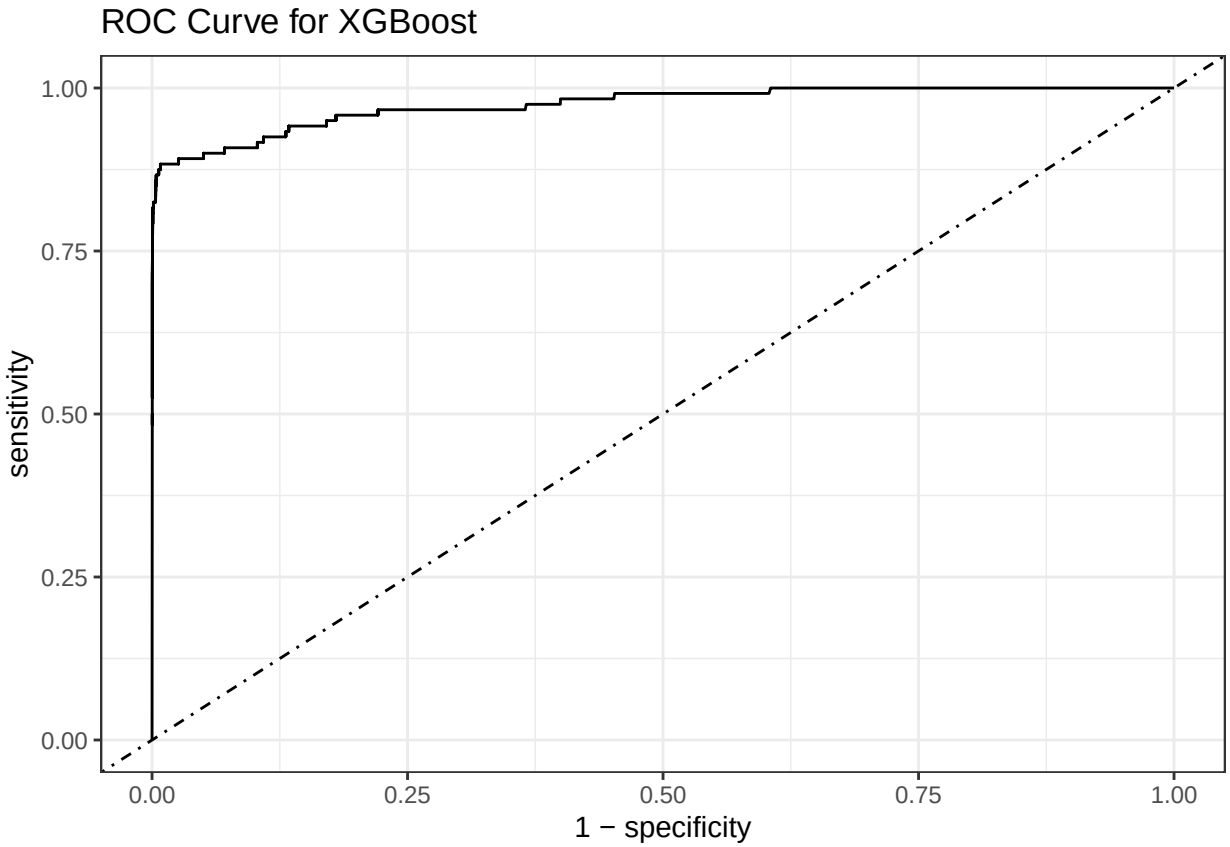



And then we use this optimal number of rounds to make the final XgBoost on the training data and assess its accuracy on the test data:

```
optimal_rounds <- train_boost$evaluation_log %>%
  slice_min(test_logloss_mean) %>%
  pull(iter) %>%
  min()

final_boost <- xgboost(data = train_mat, objective = "binary:logistic",
                       nrounds = optimal_rounds, verbose = FALSE)

# Show XGBoost results on test data
test_results_boost <- predict(final_boost,
        test %>%
          select(-Class) %>%
          data.matrix() %>%
          xgb.DMatrix()) %>%
  mutate(.data = test, preds = .) %>%
  mutate(pred_class = ifelse(preds >= .15, 1, 0)) %>% # The threshold is set to .15
                                                      # to reflect the chance at which
                                                      # you might want to get a push
                                                      # notification
  select(Class, pred_class, preds)

confusionMatrix(as.factor(test_results_boost$pred_class),
                test_results_boost$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 62151    13
##          1  2039   107
##
##                Accuracy : 0.9681
##                  95% CI : (0.9667, 0.9694)
##     No Information Rate : 0.9981
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0912
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.96823
##             Specificity : 0.89167
##          Pos Pred Value : 0.99979
##          Neg Pred Value : 0.04986
##              Prevalence : 0.99813
##          Detection Rate : 0.96643
##    Detection Prevalence : 0.96663
##       Balanced Accuracy : 0.92995
##
##        'Positive' Class : 0
##
```

```
test_results_boost %>%
  mutate(truth = factor(Class, levels = c(1, 0))) %>%
  rename(est = preds) %>%
  yardstick::roc_curve(data = ., truth = truth, est) %>%
  ggplot() +
  geom_line(aes(x = 1 - specificity, y = sensitivity)) +
  labs(title = "ROC Curve for XGBoost") +
  geom_abline(lty = 4)
```




By accuracy, specificity, and sensitivity, an XGBoost can do quite well in detecting anomalies!

## 5. Random Forest

Create random forest model

```
undersample_copy2 = undersample_df  # create copy of the undersampled data

# create random forest model
rf_model <- randomForest(Class ~ . , data = undersample_copy2)

test$predicted <- predict(rf_model, test)

confusionMatrix(test$Class, test$predicted)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 63984   206
##          1    17   103
##
##                Accuracy : 0.9965
##                  95% CI : (0.996, 0.997)
##     No Information Rate : 0.9952
##     P-Value [Acc > NIR] : 1.527e-07
##
##                   Kappa : 0.4788
```

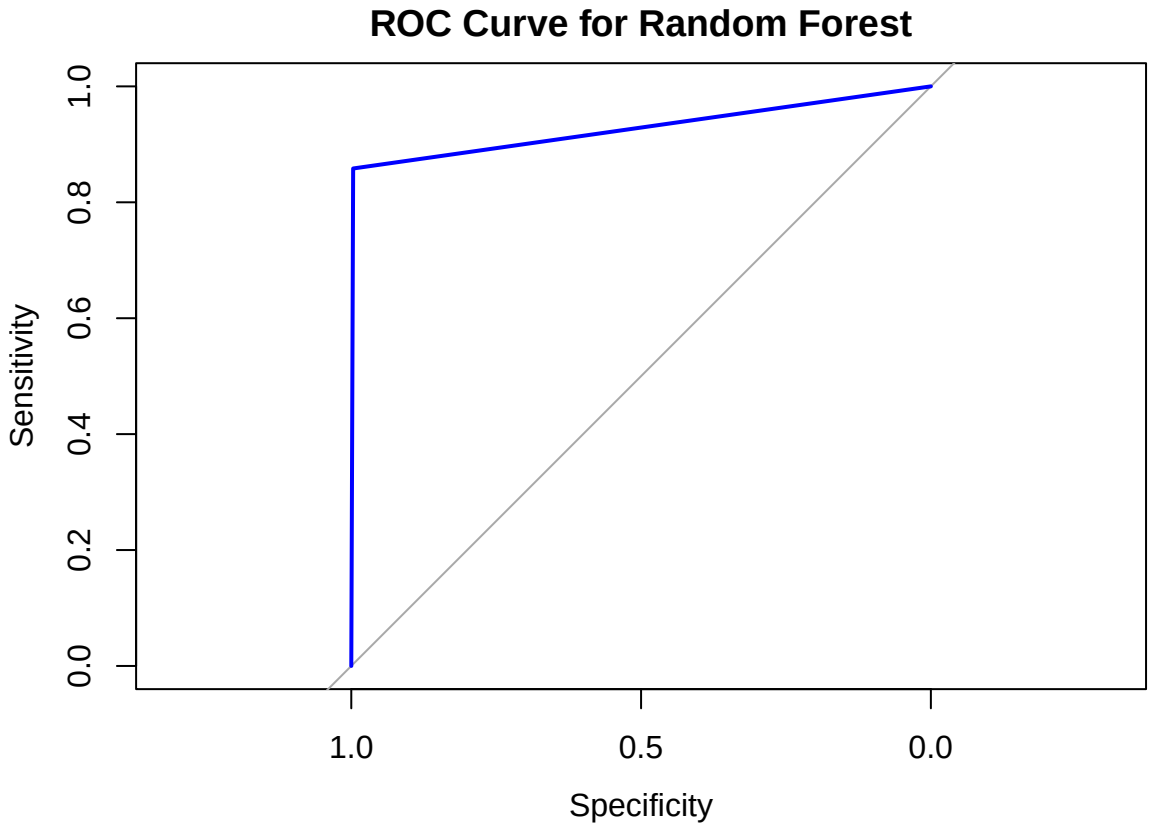
```
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9997
##             Specificity : 0.3333
##          Pos Pred Value : 0.9968
##          Neg Pred Value : 0.8583
##              Prevalence : 0.9952
##          Detection Rate : 0.9949
##    Detection Prevalence : 0.9981
##       Balanced Accuracy : 0.6665
##
##        'Positive' Class : 0
##
```

```
roc_curve <- roc(test$Class, as.numeric(test$predicted))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
plot(roc_curve, main="ROC Curve for Random Forest", col="blue")
```




Look at precision, recall, F1 score

```
confusionMatrix(test$Class, test$predicted)$byClass["Precision"] # precision
```

```
## Precision
## 0.9967908
```

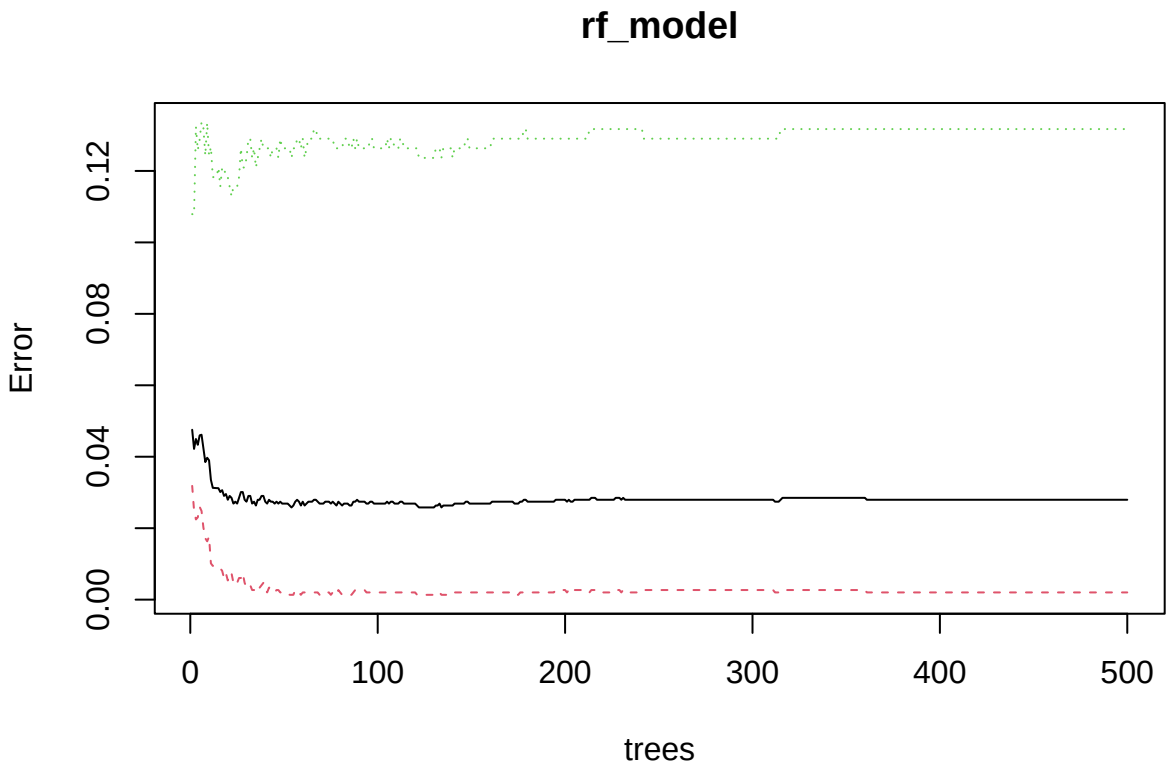
```r
confusionMatrix(test$Class, test$predicted)$byClass["Recall"] # recall
```

```
##    Recall
## 0.9997344
```

```r
confusionMatrix(test$Class, test$predicted)$byClass["F1"] #F1 score
```

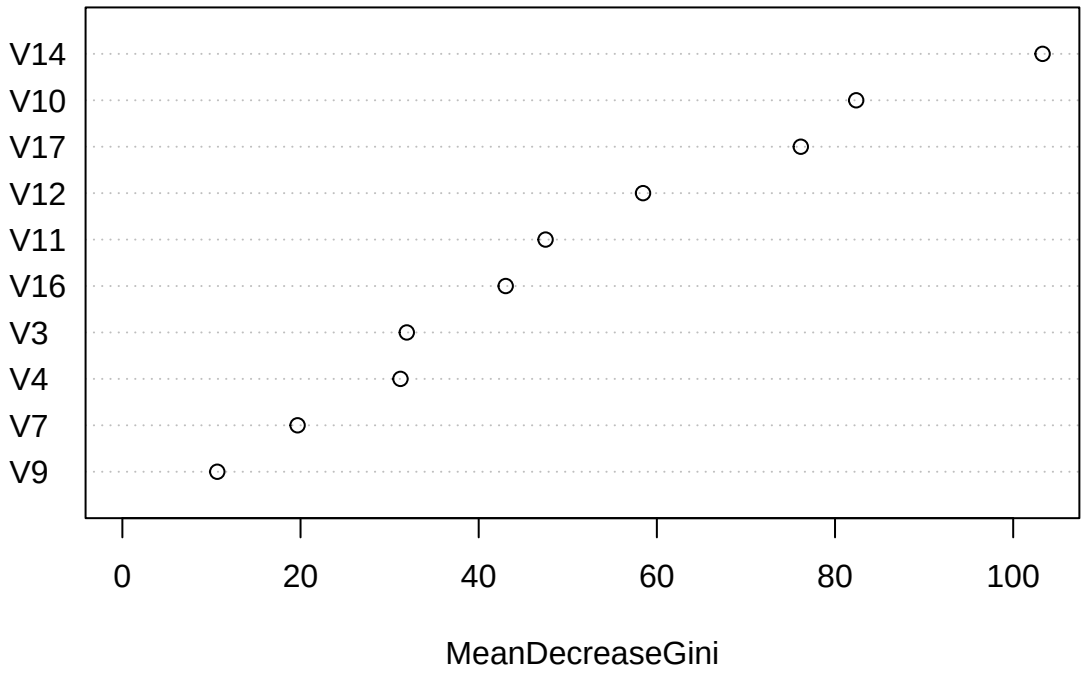
```
##        F1
## 0.9982604
```

```r
# plot the random forest model
plot(rf_model)
```




Take a look at significant predictors

```r
varImpPlot(rf_model,
           sort = T,
            n.var=10,
            main="Important Predictors")
```




## 6. Resources

- Dataset: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
  - https://www.kaggle.com/code/linhanphm/imbalanced-dataset-fraud-detection
- Reference: https://www.kaggle.com/code/naveengowda16/anomaly-detection-credit-card-fraud-analysis/notebook
  - https://www.kaggle.com/code/kimchanyoung/simple-anomaly-detection-using-unsupervised-knn
- Sources: https://www.techtarget.com/searchenterpriseai/definition/anomaly-detection
  - https://www.bankrate.com/finance/credit-cards/credit-card-fraud-statistics/#fra