Credit Card Fraud Detection

Loncón Joaquín Danielo

March 23, 2022

Introduction

The objective of this project is to build a **Machine Learning Model** that predicts if a **Credit card transaction was fraudulent or not**. To achieve this goal some classification models will be trained and then grouped into one to make a final model (**Ensemble**).

The dataset contains transactions made by credit cards that occurred in the period of two days in September 2013 by European cardholders, due to confidentiality issues, the original features and more background information about the data can not be provided. The variables are only numerical and are the result of a PCA transformation except for "Time" and "Amount".

This data is collected by the Machine Learning Group at the Université Libre de Bruxelles (MLG-ULB) which is a research unit of the Computer Science Department of the Faculty of Sciences, co-headed by Prof. Gianluca Bontempi and Prof. Tom Lenaerts. This data collection occurred during a research collaboration of Worldline and MLG-ULB on big data mining and fraud detection.

Prepare

Libraries required

```
if(!require(readr)) install.packages("readr", repos = "http://cran.us.r-project.org")
if(!require(plyr)) install.packages("plyr", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(rpart)) install.packages("rpart", repos = "http://cran.us.r-project.org")
if(!require(mboost)) install.packages("mboost", repos = "http://cran.us.r-project.org")
if(!require(MASS)) install.packages("MASS", repos = "http://cran.us.r-project.org")
if(!require(pamr)) install.packages("pamr", repos = "http://cran.us.r-project.org")
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")
if(!require(naivebayes)) install.packages("naivebayes", repos = "http://cran.us.r-project.org")

library(readr)
library(plyr)
library(caret)
library(rpart)
library(mboost)
library(MASS)
library(pamr)
library(dplyr)
library(naivebayes)
```

Obtain the data

Load the data

```
Credit_Card_Fraud_Detection <- read.csv("../input/creditcardfraud/creditcard.csv")
```

Pre-Process

Variable Format Changes

Make the variable of interest (Fraud) a factor and re-level the variable for later interpretation in the models, this makes that the fraud level can be interpreted as $Y = 1$.

```
names(Credit_Card_Fraud_Detection)[names(Credit_Card_Fraud_Detection)=="Class"] <- "Fraud"

Credit_Card_Fraud_Detection$Fraud <- factor(Credit_Card_Fraud_Detection$Fraud, labels = c("Normal", "Fra
Credit_Card_Fraud_Detection$Fraud <- relevel(Credit_Card_Fraud_Detection$Fraud, "Fraud")
```

Missing values

No value is missing in the dataframe, we can see this with the following simple code.

```
any(is.na(Credit_Card_Fraud_Detection))
```

```
## [1] FALSE
```

Data partition for validation

Create a partition of the data to later test the final model.

```
set.seed(1, sample.kind = "Rounding") # just for make the code reproducible

validation_index <- createDataPartition(y = Credit_Card_Fraud_Detection$Fraud, times = 1, p = 0.2, list

validation <- Credit_Card_Fraud_Detection[validation_index,]
df <- Credit_Card_Fraud_Detection[-validation_index,]
```

Train & Test sets

To train the models, test, and optimize, generate an index, with 80% of the data for training the model and 20% for testing (without using the validation set, which is going to be used at the end).

```
train_index <- createDataPartition(y = df$Fraud, times = 1, p = 0.8, list = FALSE)

train_set <- df[train_index,]
test_set <- df[-train_index,]
```

Analysis

Structure & simple statistics

Let's start the analysis by looking at some summary statistics. Only a few variables are shown for space reasons and because we don't now much about the meaning of the continuous variables in the data.

```
str(df[,c(1,2,3,30,31)])
```

```
## tibble [227,845 x 5] (S3: tbl_df/tbl/data.frame)
##  $ Time  : num [1:227845] 0 0 1 2 2 4 7 7 9 10 ...
##  $ V1    : num [1:227845] -1.36 1.192 -0.966 -1.158 -0.426 ...
##  $ V2    : num [1:227845] -0.0728 0.2662 -0.1852 0.8777 0.9605 ...
##  $ Amount: num [1:227845] 149.62 2.69 123.5 69.99 3.67 ...
##  $ Fraud : Factor w/ 2 levels "Fraud","Normal": 2 2 2 2 2 2 2 2 2 2 ...
```

```
summary(df[,c(1,2,3,30,31)])
```

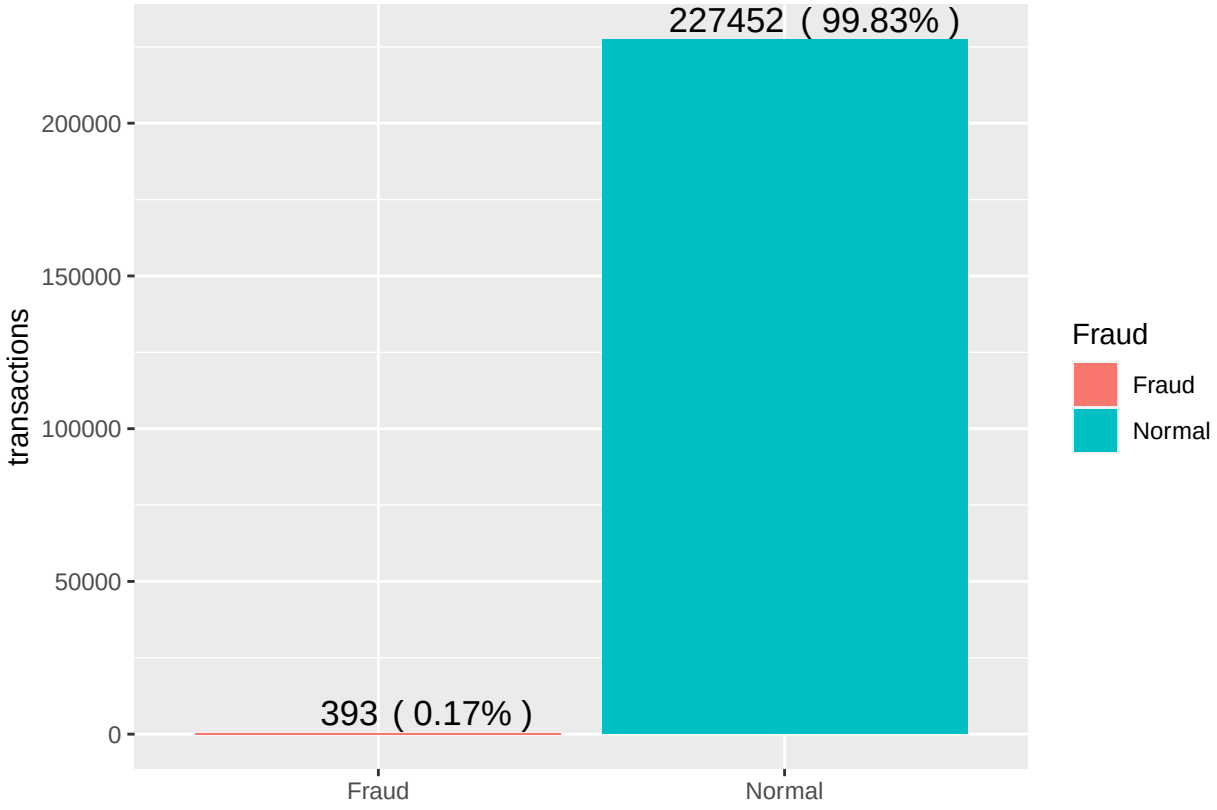
```
##       Time              V1                  V2                Amount        
##  Min.   :     0   Min.   :-56.40751   Min.   :-72.71573   Min.   :    0.00  
##  1st Qu.: 54230   1st Qu.: -0.92170   1st Qu.: -0.59880   1st Qu.:    5.50  
##  Median : 84695   Median :  0.01592   Median :  0.06618   Median :   22.00  
##  Mean   : 94820   Mean   : -0.00192   Mean   :  0.00013   Mean   :   88.62  
##  3rd Qu.:139316   3rd Qu.:  1.31450   3rd Qu.:  0.80412   3rd Qu.:   77.45  
##  Max.   :172792   Max.   :  2.45493   Max.   : 22.05773   Max.   :18910.00  
##     Fraud       
##  Fraud :   393  
##  Normal:227452  
##                 
##                 
##                 
##
```

Visual Analysis

Fraud Distribution

This graph shows the high prevalence in the data, only 0.17% of the transactions in the data were fraudulent and the other 99.83% do not.

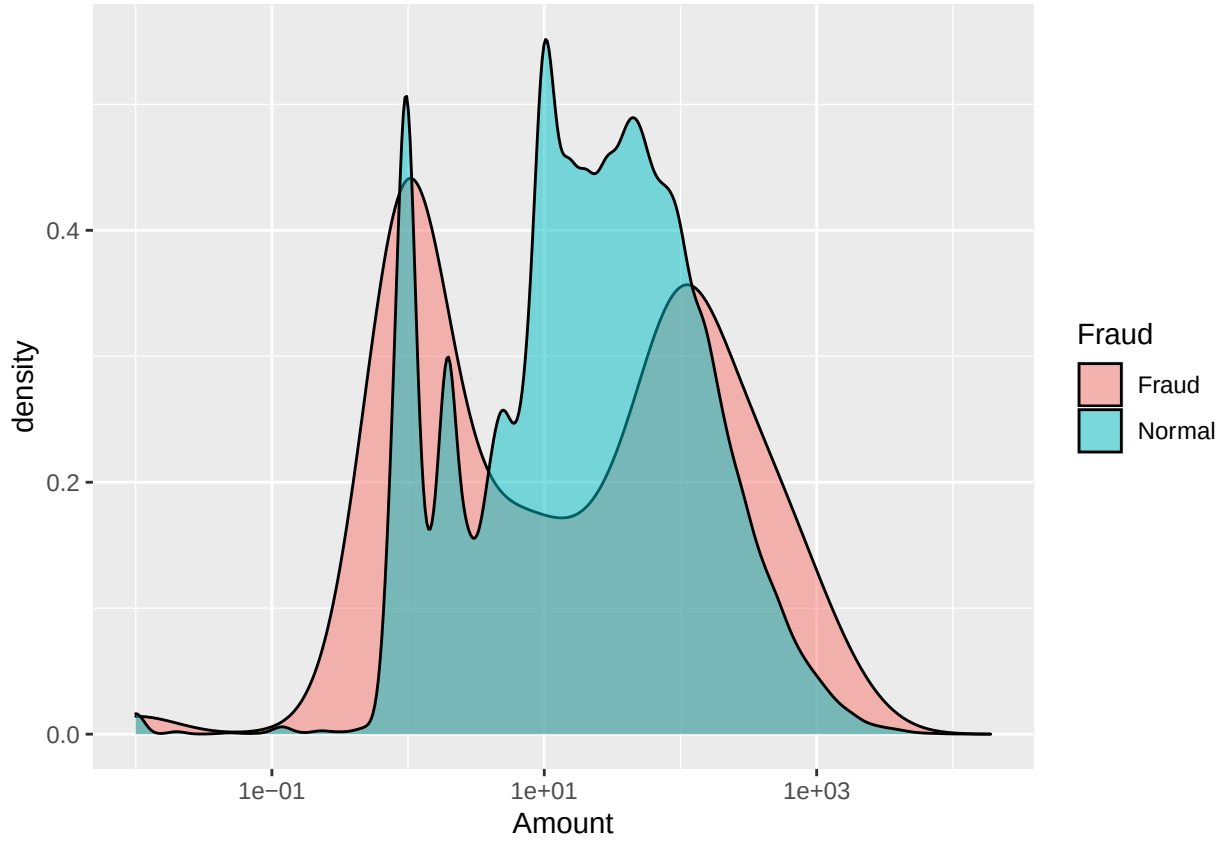
```
df%>%
    group_by(Fraud)%>%
    summarise(n = n())%>%
    mutate(percentage = n/sum(n)*100)%>%
    ggplot(aes(Fraud, n, fill=Fraud))+
    geom_bar(stat = "identity")+
    geom_text(aes(label=n), vjust=-0.3, hjust = 1, size=4.5)+
    geom_text(aes(label=paste0("( ",round(percentage,2),"% )")), vjust=-0.3, hjust = -0.1, size=4.5)+
    ylab("transactions")+
    xlab("")
```



Fraud Distribution by Amount (scaled)

See the relation between the amount of the transaction and if it is fraudulent or not.

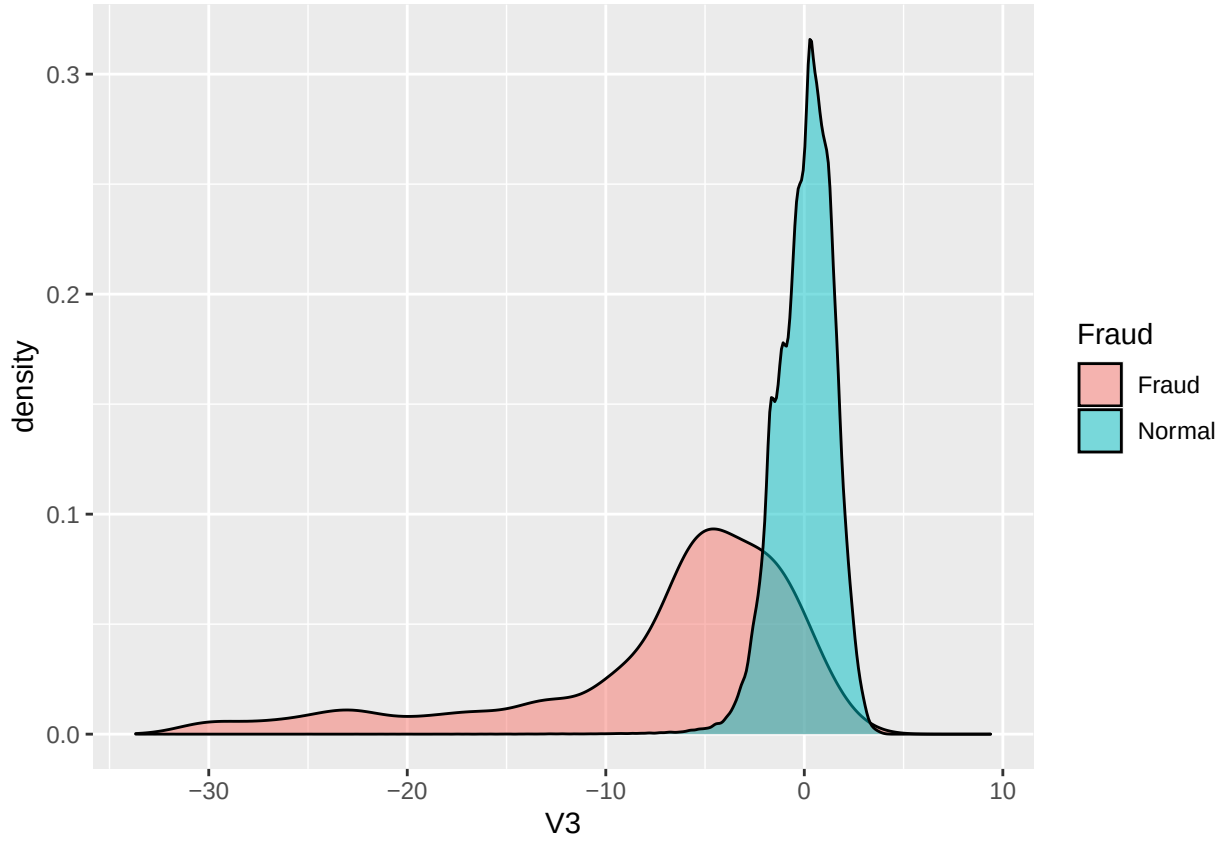
```
df%>%
    group_by(Fraud)%>%
    ggplot()+
    geom_density(aes(Amount, fill = Fraud), alpha = .5)+
    scale_x_log10()
```



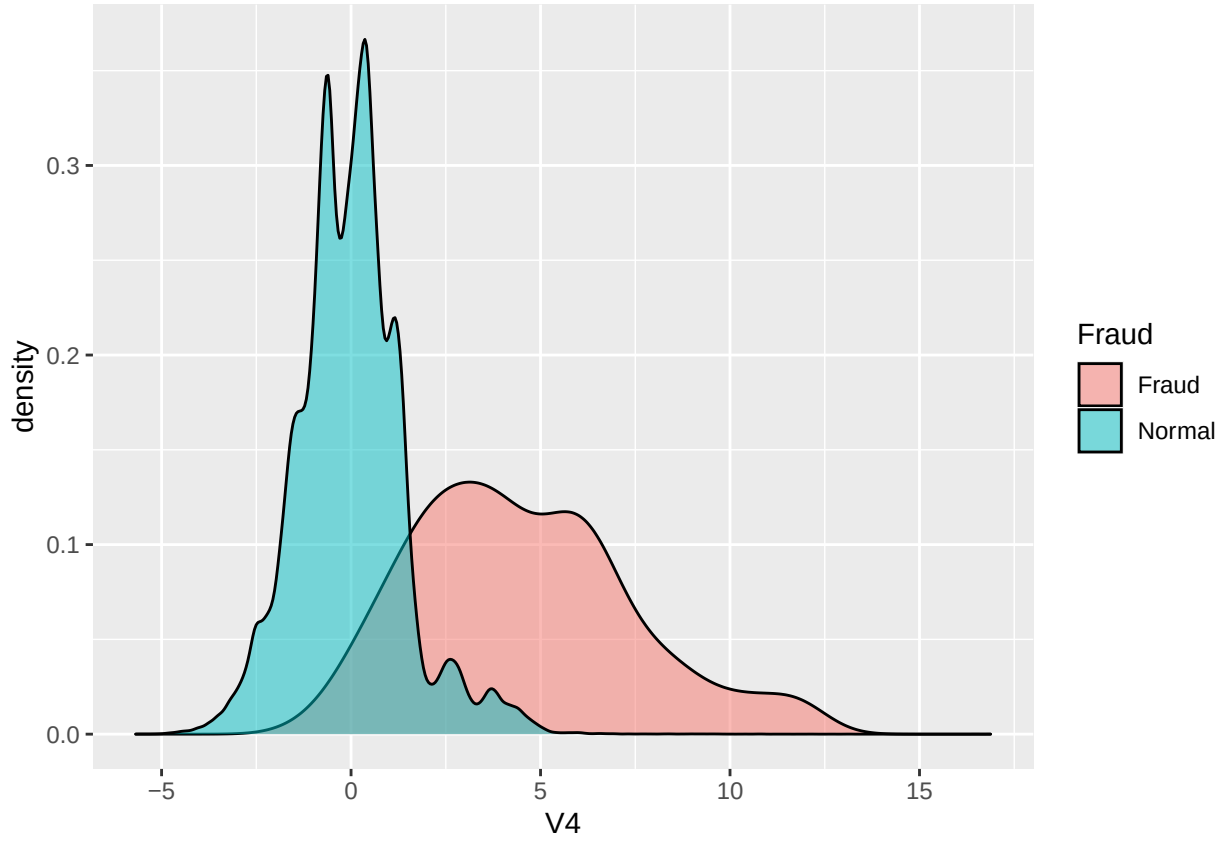
Fraud Distribution by other variables

Here we can see that for some variables we can easily discriminate the fraudulent transactions

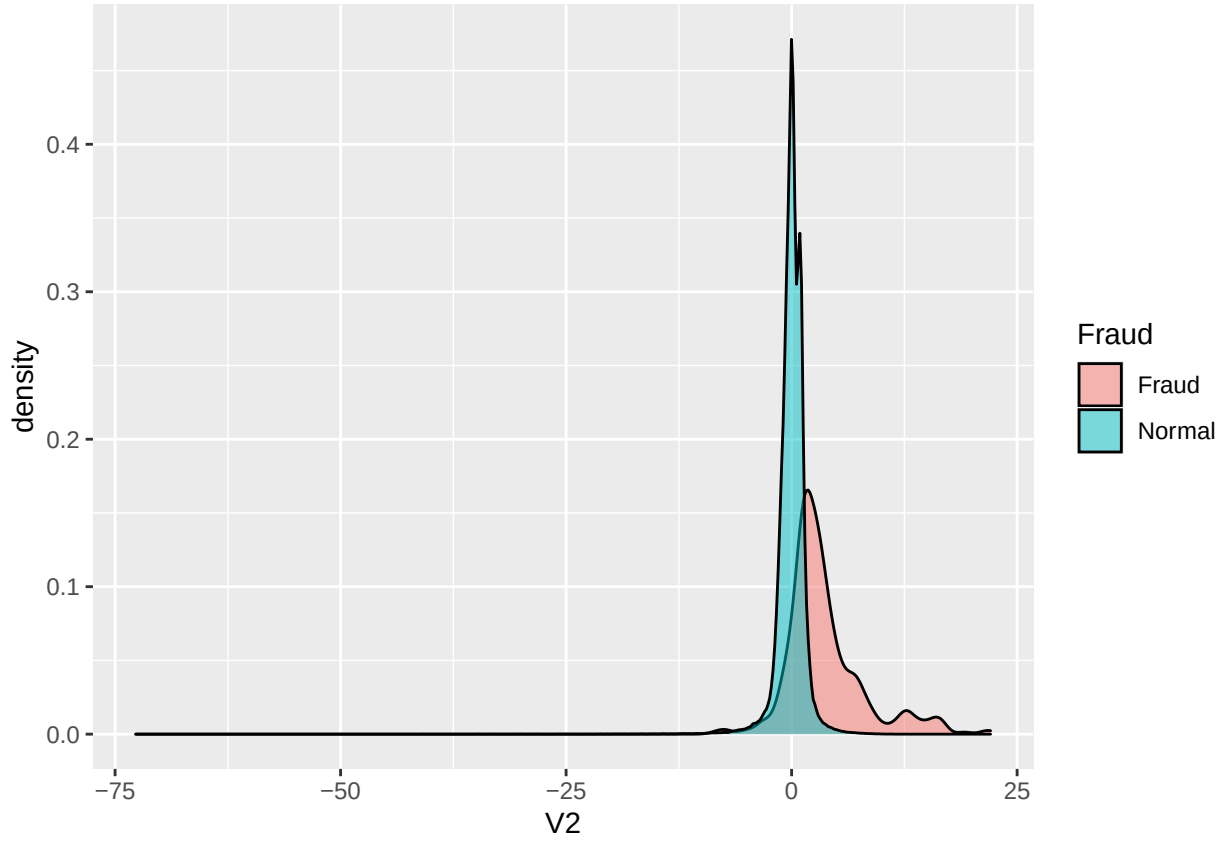
```
df%>%
    group_by(Fraud)%>%
    ggplot()+
    geom_density(aes(V3, fill = Fraud), alpha = .5)
```




```
df%>%
    group_by(Fraud)%>%
    ggplot()+
    geom_density(aes(V4, fill = Fraud), alpha = .5)
```

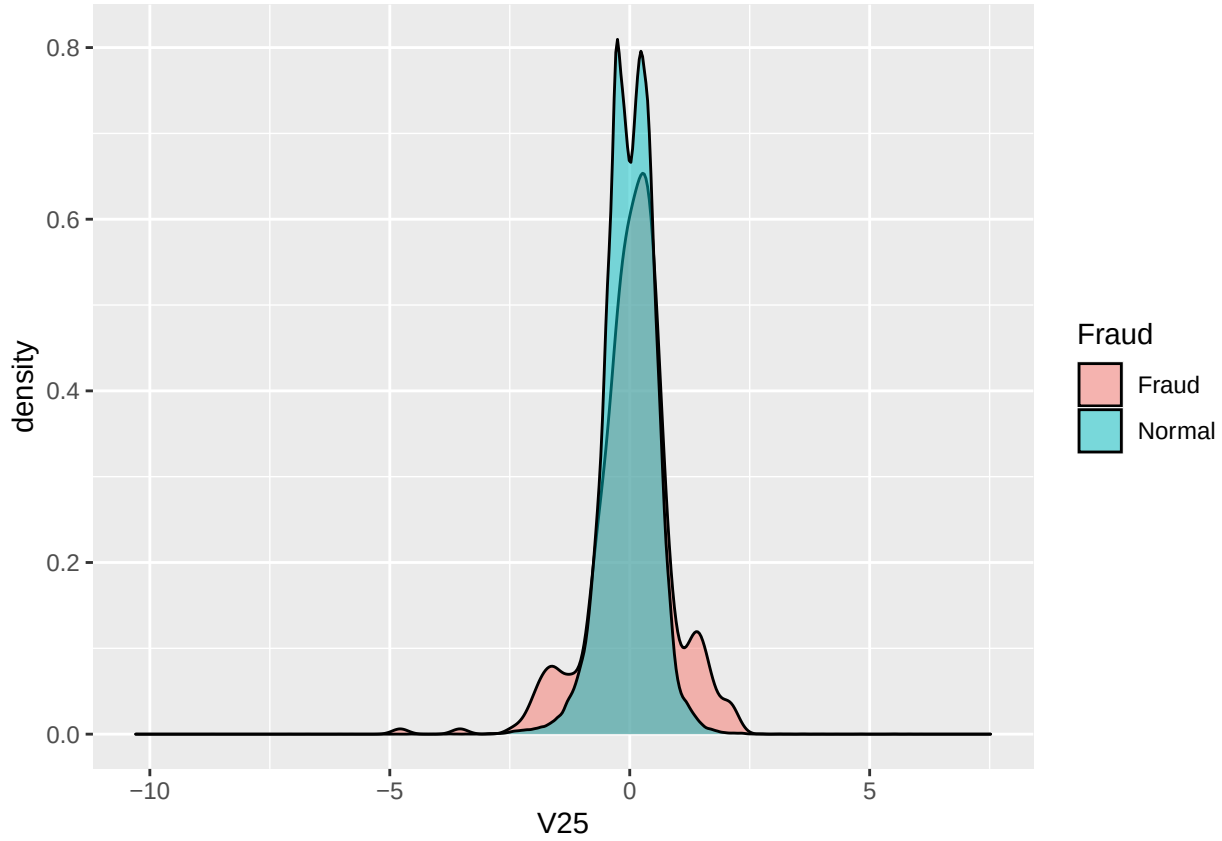



```
df%>%
    group_by(Fraud)%>%
    ggplot()+
    geom_density(aes(V2, fill = Fraud), alpha = .5)
```



But for others this would be a little more difficult.

```
df%>%
    group_by(Fraud)%>%
    ggplot()+
    geom_density(aes(V25, fill = Fraud), alpha = .5)
```



Model Building

Prevalence Problem

As shown above, there is a high prevalence of one class in the fraud variable. This brings big problems when creating a model, as seen below a classification tree is trained and its performance on the test set is displayed.

```
fit_bad_model <- train(Fraud~.,
                       data = train_set,
                       method = "rpart",
                       trControl = trainControl(method = "cv", number = 5))

hat_bad_model <- predict(fit_bad_model, newdata = test_set)

confusionMatrix(data = hat_bad_model, reference = test_set$Fraud)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fraud Normal
##     Fraud     58     12
##     Normal    20  45478
##
```

```
##                Accuracy : 0.9993
##                  95% CI : (0.999, 0.9995)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 2.917e-09
##
##                   Kappa : 0.7834
##
##  Mcnemar's Test P-Value : 0.2159
##
##             Sensitivity : 0.743590
##             Specificity : 0.999736
##          Pos Pred Value : 0.828571
##          Neg Pred Value : 0.999560
##              Prevalence : 0.001712
##          Detection Rate : 0.001273
##    Detection Prevalence : 0.001536
##       Balanced Accuracy : 0.871663
##
##        'Positive' Class : Fraud
##
```

But, note that even with this "Bad model" we got a relative high **Recall** (Sensitivity), good **Accuracy** and an important **Precision** (Pos Pred Value), note that this last variable it's also really important in this problem because we may want to end up with high Sensitivity to prevent fraudulent transactions, but, if we predict fraudulent to much, the users won't be able to use their cards because they'll probably be blocked, which would cause problems since many cards that make normal transactions would be unusable. But at the end, there is a trade-off between recall and precision, so we must make a decision.

Up-Sampling

To deal with the problem of prevalence we can do a technique called up-sampling. this method increases the size of the minority class by sampling with replacement so that the classes in the data will have the same size. Note that by doing this, the observations will balance out, causing there to be no more prevalence.

```
trainup <- upSample(y=train_set$Fraud,
                    x=train_set[,-ncol(train_set)],
                    yname = "Fraud")
table(trainup$Fraud)
```

```
##
##  Fraud Normal
## 181962 181962
```

Machine Learning Models

Now with the balanced data we are going to train a set of different models for classification and observe their performances on the test set. Predictors will be centered and scaled by the preProcess option. In addition, we are going to use a 5-fold cross-validation, this is randomly split the observations of our train set into 5 non-overlapping sets, and do the calculation of the Mean Squared Error (MSE) for each of these sets having into account the different parameters of the model that lead us to that MSE, then the model selects the optimal parameters, the ones that minimize the MSE.

Classification Tree

```
fit_rpart <- train(Fraud ~ .,
                   data = trainup,
                   method = "rpart",
                   trControl = trainControl(method = "cv", number = 5),
                   preProcess = c("center", "scale"))

hat_rpart <- predict(fit_rpart, newdata = test_set)

confusionMatrix(data = hat_rpart, reference = test_set$Fraud)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fraud Normal
##     Fraud     70   1703
##     Normal     8  43787
##
##                Accuracy : 0.9625
##                  95% CI : (0.9607, 0.9642)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0726
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.897436
##             Specificity : 0.962563
##          Pos Pred Value : 0.039481
##          Neg Pred Value : 0.999817
##              Prevalence : 0.001712
##          Detection Rate : 0.001536
##    Detection Prevalence : 0.038909
##       Balanced Accuracy : 0.930000
##
##        'Positive' Class : Fraud
##
```

Generalized Linear Model

```
fit_glm <- train(Fraud ~ .,
                   data = trainup,
                   method = "glm",
                   trControl = trainControl(method = "cv", number = 5),
                   preProcess = c("center", "scale"))

hat_glm <- predict(fit_glm, newdata = test_set)

confusionMatrix(data = hat_glm, reference = test_set$Fraud)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fraud Normal
##     Fraud     72   1030
##     Normal     6  44460
##
##                Accuracy : 0.9773
##                  95% CI : (0.9759, 0.9786)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1192
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.923077
##             Specificity : 0.977358
##          Pos Pred Value : 0.065336
##          Neg Pred Value : 0.999865
##              Prevalence : 0.001712
##          Detection Rate : 0.001580
##    Detection Prevalence : 0.024184
##       Balanced Accuracy : 0.950217
##
##        'Positive' Class : Fraud
##
```

Nearest Shrunken Centroids

```
fit_pam <- train(Fraud ~ .,
                 data = trainup,
                 method = "pam",
                 trControl = trainControl(method = "cv", number = 5),
                 preProcess = c("center", "scale"))

hat_pam <- predict(fit_pam, newdata = test_set)

confusionMatrix(data = hat_pam, reference = test_set$Fraud)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fraud Normal
##     Fraud     60     30
##     Normal    18  45460
##
##                Accuracy : 0.9989
##                  95% CI : (0.9986, 0.9992)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 0.0001748
##
```

```
##                  Kappa : 0.7138
##
##  Mcnemar's Test P-Value : 0.1123512
##
##             Sensitivity : 0.769231
##             Specificity : 0.999341
##          Pos Pred Value : 0.666667
##          Neg Pred Value : 0.999604
##              Prevalence : 0.001712
##          Detection Rate : 0.001317
##    Detection Prevalence : 0.001975
##       Balanced Accuracy : 0.884286
##
##        'Positive' Class : Fraud
##
```

Naive Bayes

```
fit_nb <- train(Fraud ~ .,
                data = trainup,
                method = "naive_bayes",
                trControl = trainControl(method = "cv", number = 5),
                preProcess = c("center", "scale"))

hat_nb <- predict(fit_nb, newdata = test_set)

confusionMatrix(data = hat_nb, reference = test_set$Fraud)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fraud Normal
##     Fraud     65   1180
##     Normal    13  44310
##
##                Accuracy : 0.9738
##                  95% CI : (0.9723, 0.9753)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0953
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.833333
##             Specificity : 0.974060
##          Pos Pred Value : 0.052209
##          Neg Pred Value : 0.999707
##              Prevalence : 0.001712
##          Detection Rate : 0.001426
##    Detection Prevalence : 0.027322
##       Balanced Accuracy : 0.903697
```

```
##
##        'Positive' Class : Fraud
##
```

Generalized linear model by likelihood based boosting

```
fit_glmboost <- train(Fraud ~ .,
                data = trainup,
                method = "glmboost",
                trControl = trainControl(method = "cv", number = 5),
                preProcess = c("center", "scale"))

hat_glmboost <- predict(fit_glmboost, newdata = test_set)

confusionMatrix(data = hat_glmboost, reference = test_set$Fraud)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fraud Normal
##     Fraud     66    427
##     Normal    12  45063
##
##                Accuracy : 0.9904
##                  95% CI : (0.9894, 0.9912)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2289
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.846154
##             Specificity : 0.990613
##          Pos Pred Value : 0.133874
##          Neg Pred Value : 0.999734
##              Prevalence : 0.001712
##          Detection Rate : 0.001448
##    Detection Prevalence : 0.010819
##       Balanced Accuracy : 0.918384
##
##        'Positive' Class : Fraud
##
```

Linear Discriminant Analysis

```
fit_lda <- train(Fraud ~ .,
                      data = trainup,
                      method = "lda",
                      trControl = trainControl(method = "cv", number = 5),
```

```
                     preProcess = c("center", "scale"))

hat_lda <- predict(fit_lda, newdata = test_set)

confusionMatrix(data = hat_lda, reference = test_set$Fraud)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fraud Normal
##     Fraud     62    691
##     Normal    16  44799
##
##                Accuracy : 0.9845
##                  95% CI : (0.9833, 0.9856)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1466
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.794872
##             Specificity : 0.984810
##          Pos Pred Value : 0.082337
##          Neg Pred Value : 0.999643
##              Prevalence : 0.001712
##          Detection Rate : 0.001361
##    Detection Prevalence : 0.016525
##       Balanced Accuracy : 0.889841
##
##        'Positive' Class : Fraud
##
```

Ensemble

An ensemble is the concept of combining the results of different algorithms, i.e. group different machine learning algorithms into one, this often greatly improve the final results. To achieve this goal we are going to take the vector of conditional probabilities given to each class by every particular algorithm, and then sum all the conditional probabilities given by the algorithms and take an average, so the result would be an average conditional probability for each observation.

Conditional Probabilities

First save the conditional probabilities with the option `type = "prob"` in the `predict()` function and then create the ensemble.

```
p_rpart <- predict(fit_rpart, newdata = test_set, type = "prob")
p_glm <- predict(fit_glm, newdata = test_set, type = "prob")
p_pam <- predict(fit_pam, newdata = test_set, type = "prob")
```

```r
p_nb <- predict(fit_nb, newdata = test_set, type = "prob")
p_glmboost <- predict(fit_glmboost, newdata = test_set, type = "prob")
p_lda <- predict(fit_lda, newdata = test_set, type = "prob")

p <- (p_rpart + p_glm + p_pam + p_nb + p_glmboost + p_lda )/6

head(p)
```

```
##       Fraud    Normal
## 1 0.1451888 0.8548112
## 2 0.1425308 0.8574692
## 3 0.1373561 0.8626439
## 4 0.1214434 0.8785566
## 5 0.1195205 0.8804795
## 6 0.1250242 0.8749758
```

Ensemble Results

Now test the performance of the ensemble.

```r
y_pred <- factor(apply(p, 1, which.max))

y_pred <- factor(y_pred, labels = c("Fraud", "Normal"))

confusionMatrix(y_pred, test_set$Fraud)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fraud Normal
##     Fraud     68    548
##     Normal    10  44942
##
##                Accuracy : 0.9878
##                  95% CI : (0.9867, 0.9887)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1935
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.871795
##             Specificity : 0.987953
##          Pos Pred Value : 0.110390
##          Neg Pred Value : 0.999778
##              Prevalence : 0.001712
##          Detection Rate : 0.001492
##    Detection Prevalence : 0.013518
##       Balanced Accuracy : 0.929874
##
##        'Positive' Class : Fraud
##
```

Final Model

Up-Sampling

We use the up-sampling technique again but in the `df` dataset.

```
df_trainup <- upSample(y=df$Fraud,
                       x=df[,-ncol(df)],
                       yname = "Fraud")

table(df_trainup$Fraud)
```

```
##
##  Fraud Normal
## 227452 227452
```

Machine Learning Models

And now train the models with all the `df_trainup` set

Classification Tree

```
fit_rpart_final <- train(Fraud ~ .,
                  data = df_trainup,
                  method = "rpart",
                  trControl = trainControl(method = "cv", number = 5),
                  preProcess = c("center", "scale"))

p_rpart_final <- predict(fit_rpart_final, newdata = validation, type = "prob")
```

Generalized Linear Model

```
fit_glm_final <- train(Fraud ~ .,
                 data = df_trainup,
                 method = "glm",
                 trControl = trainControl(method = "cv", number = 5),
                 preProcess = c("center", "scale"))

p_glm_final <- predict(fit_glm_final, newdata = validation, type = "prob")
```

Nearest Shrunken Centroids

```
fit_pam_final <- train(Fraud ~ .,
                 data = df_trainup,
                 method = "pam",
                 trControl = trainControl(method = "cv", number = 5),
```

```
                preProcess = c("center", "scale"))

p_pam_final <- predict(fit_pam_final, newdata = validation, type = "prob")
```

Naive Bayes

```
fit_nb_final <- train(Fraud ~ .,
                data = df_trainup,
                method = "naive_bayes",
                trControl = trainControl(method = "cv", number = 5),
                preProcess = c("center", "scale"))

p_nb_final <- predict(fit_nb_final, newdata = validation, type = "prob")
```

Generalized linear model by likelihood based boosting

```
fit_glmboost_final <- train(Fraud ~ .,
                      data = df_trainup,
                      method = "glmboost",
                      trControl = trainControl(method = "cv", number = 5),
                      preProcess = c("center", "scale"))

p_glmboost_final <- predict(fit_glmboost_final, newdata = validation, type = "prob")
```

Linear Discriminant Analysis

```
fit_lda_final <- train(Fraud ~ .,
                data = df_trainup,
                method = "lda",
                trControl = trainControl(method = "cv", number = 5),
                preProcess = c("center", "scale"))

p_lda_final <- predict(fit_lda_final, newdata = validation, type = "prob")
```

Final Ensemble

And now make the final Ensemble.

```
p_final <- (p_rpart_final + p_glm_final + p_pam_final + p_nb_final + p_glmboost_final + p_lda_final)/6

y_final_pred <- factor(apply(p_final, 1, which.max))

y_final_pred <- factor(y_final_pred, labels = c("Fraud", "Normal"))
```

Results

Final Model

The results of our Ensemble, the **Final Model**, are as follows:

```
confusionMatrix(y_final_pred, validation$Fraud)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Fraud Normal
##     Fraud     85    620
##     Normal    14  56243
##
##                Accuracy : 0.9889
##                  95% CI : (0.988, 0.9897)
##     No Information Rate : 0.9983
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.209
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.858586
##             Specificity : 0.989097
##          Pos Pred Value : 0.120567
##          Neg Pred Value : 0.999751
##              Prevalence : 0.001738
##          Detection Rate : 0.001492
##    Detection Prevalence : 0.012377
##       Balanced Accuracy : 0.923841
##
##        'Positive' Class : Fraud
##
```

Let's interpret these results.

The **Accuracy** is the ratio of correct predictions to total predictions made, the accuracy of our model was 0.9888698, this means that 98.89% of the time we predict classes well, but due to the prevalence of one class in the data we may want to look at other metrics like the **Balanced Accuracy**, this metric is the average between the sensitivity and specificity.

The **Sensitivity** (also called **Recall**) is the number of correct positive predictions, i.e. we predict $\widehat{Y} = 1$ and indeed $Y = 1$, divided by the total number of positives, all the times that $Y = 1$. In our case, the sensitivity represents the number of times we correctly predict a fraud, divided by the total number of fraud cases. So we can correctly predict a fraud 85.86% of the times.

The **Specificity** is the number of correct negative predictions, divided by the total number of negatives, i.e. $\widehat{Y} = 0$ and indeed $Y = 0$ divided by all the times that $Y = 0$. In our case, the sensitivity represents the number of times we correctly predict a normal transaction, divided by the total number of normal transactions. Therefore, we are able to correctly predict a normal transaction 98.91% of the times.

If we remember, the **Balanced Accuracy** was the average of this two last metrics ($\frac{Sensitivity + Specificity}{2}$), and we may prefer to use it instead of Accuracy due to the imbalance, because note that if we just predict

normal transactions we can achieve a high accuracy due to prevalence. However, if we use Balanced Accuracy we weight our predictive ability for both classes equivalently and is more representative of our predictive power. In our case, we have correct predictions 92.38% of the times by weighting both classes equally (taking into account the prevalence).

Last, but not least, the "Pos Pred Value", the **Precision**, is the number of correct positive predictions, i.e. we predict $\widehat{Y} = 1$ and indeed $Y = 1$, divided by the total number of positive predictions, all the times that we predict $\widehat{Y} = 1$. It is important to say that there is a trade-off between Recall and Precision for a given model, if we want to improve one, the other will get worse. In our case the 12.06% of the times that we predicted fraud, it was indeed fraud. If we want to think of this in terms of the normal transactions we can look at "Neg Pred Value".

Conclusion

In summary, we see that making an ensemble of our models improves the final estimation but we cannot do much with the trade-off between Recall and Precision.

Depending of what we want we can make the following decision, if we want to end up with a high Recall, i.e. have a high detection rate of the fraudulent transactions, we will inevitably end predicting fraudulent transactions that were normal, this approach will make that the costumers don't lose money because of crimes, but a lot more of costumers will we unable to use their cards since probably be locked do to fraudulent prediction, and this could also be bad, e.g. generate loses for a card company or affect the consumption of the people.

A second option is have a high precision at the cost of recall, it should be noted that we can also save a lot of losses by not predicting cards with normal transactions as fraudulent, so perhaps we could be willing to miss a fraudulent transaction detection and not block for example 1000 cards. In addition there may be different legal mechanisms to recover the loss of the fraudulent transaction.

The point I want to get to is that, at the end of the day one wants to detect fraudulent transactions because they generate economic discomfort in people, but giving up everything in order to try to predict fraudulent transactions may not be the best idea since it also has its economic counterparts, a middle point in this trade-off, as we can expect is achieved by our ensemble, may be a good option.