Credit Card Fraud Detection

Fernando José Velasco Borea

May 12th 2019

Contents

1 Introduction and Overview

An article conducted by Loss Prevention Magazine in 2018 showed that by 2020 the total monetary losses due to credit card fraud in the U.S. alone could excede the $10,000,000,000 mark (you can find the article here). With a constant growth on cardholders across the years, the concern about this type of fraud has also increased. On 2017 we saw an increment of 1.3 million credit card fraud victims, implying an increase of 8.4% when compared to the 2016 period (as reported by Javelin Strategy & Research). Taking this into account, I decided to conduct a supervised machine learning project with the goal of predicting potential fraudulent credit card transactions.

For this project, we will be using the data set provided by Machine Learning Group - ULB through Kaggle (you can find it through this link). The data set contains information about the time (relative to the frequency of the transactions when compared to the first one in the data set), amount, type of transaction (either fraudulent or non-fraudulent, represented by a 1 or a 0 respectively) and 28 numerical features resulting from a PCA Dimensionality Reduction to protect the users identity and sensitive information.

The project will be divided into 4 major sections, as follows:

1. Data Adquisition
2. Data Exploration and Wrangling
3. Modeling
4. Testing

Once we complete the sections mentioned above, we will create a *Conclusions* section with the insights we gathered throughout the project.

1.1 Side Notes

Although the data set used for this project is downloaded within the code, to improve the run time, it is recommended to clone the GitHub repository as it contains the `csv` file with the data set we used.

To enhance code readability when viewing the Rmd version of this report and/or when viewing the Credit Card Fraud Detection Script file to see only the coding part of the project, you can *fold* the all the sections from RStudio to then just *unfold* the section you are currently viewing, therefore, easing the interpretation of the code.

You can quickly do this from RStudio going to *Edit > Folding > Collapse All* or simply with the shortcut *ALT + O* on windows. If you want to exapnd all the sections again, you can use the shortcut *ALT + SHIFT + O* on windows or from *Edit > Folding > Expand All.*

The code contained in this report can be found on the Credit Card Fraud Detection Script file. It follows the same structure and order as the report, therefore, making it easier to reproduce the results while maintaining code readability.

To render the Rmd version of this report you will need to have a LaTeX installation. If you don't have it, you can find more details on how to install it here.

2 Data Adquisition

This section is going be mainly intended to download or read the data set (depending if you have the repository cloned into your local machine) that we will be using throughout the project.

First, we will start by loading the required libraries for our project, and then proceed to read our data either from our working directory if we cloned the repository, or from Git LFS if we have not. Note that because of formatting purposes, we will not show the output messages from the code below on the report.

Executing this code section might take some minutes depending on your internet connection.

```r
if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")

if(!require(RCurl)) install.packages("RCurl",
                                     repos = "http://cran.us.r-project.org")

if(!require(knitr)) install.packages("knitr",
                                     repos = "http://cran.us.r-project.org")

if(!require(caret)) install.packages("caret",
                                     repos = "http://cran.us.r-project.org")

if(!require(randomForest)) install.packages("randomForest",
                                     repos = "http://cran.us.r-project.org")

if(file.exists("creditcard.csv"))
{

  cc_dataset <- read_csv("creditcard.csv")

} else {

  URL_p1 <- "https://media.githubusercontent.com"
  URL_p2 <- "/media/FernandoBorea/Credit-Card-Fraud-Detection/master/creditcard.csv"
  datURL <- getURL(paste(URL_p1, URL_p2, sep = ""))

#We divided the entire URL in 2 string vectors and
#then used paste to maintain the report formatting

  cc_dataset <- read_csv(datURL)

}
```

2.1 Preliminary Data Exploration

Once the Data Adquisition process is finished, we will start performing some preliminary data exploration to make sure the data was downloaded and/or read correctly and to familiarize ourselves with the data set.

When calling the function `str()` to look for the data structure, it will result in quite a large and somewhat messy output within our report. We already know from the Kaggle Site where we got our data from, that we have several columns in our data set, therefore we are not going to include the output of the code below.

```r
str(cc_dataset)
```

As we did not show the output of the code above, we will use another approach to still show some information about the data set within this section, more specifically, we will just check the amount of rows and columns as well as the class of each column:

```r
data.frame(Columns = ncol(cc_dataset), Rows = nrow(cc_dataset)) %>%
  knitr::kable()
```

Columns	Rows
31	284807

```r
col_classes <- data.frame(Column = colnames(cc_dataset)[1:16],
                          Class = unname(apply(cc_dataset, 2, class))[1:16],
                          Column = c(colnames(cc_dataset)[17:ncol(cc_dataset)],""),
                          Class = c(unname(apply(cc_dataset, 2, class))[17:ncol(cc_dataset)],""))

colnames(col_classes) <- c("Column", "Class", "Column","Class")

col_classes %>% knitr::kable()
```

Column	Class	Column	Class
Time	numeric	V16	numeric
V1	numeric	V17	numeric
V2	numeric	V18	numeric
V3	numeric	V19	numeric
V4	numeric	V20	numeric
V5	numeric	V21	numeric
V6	numeric	V22	numeric
V7	numeric	V23	numeric
V8	numeric	V24	numeric
V9	numeric	V25	numeric
V10	numeric	V26	numeric
V11	numeric	V27	numeric
V12	numeric	V28	numeric
V13	numeric	Amount	numeric
V14	numeric	Class	numeric
V15	numeric		

3 Data Exploration and Wrangling

Now, as we finished the Data Adquisition phase, we will dive into our data set to gather useful insights and perform some data wrangling if necessary for the Modeling phase. As we saw from our Preliminary Data Exploration section, we are dealing with a slightly large amount of variables.

If we evaluate the chart containing the classes for each column on the previous section, we can notice that all of them contain numeric values, but the last one (the Class column) contains a binary value, either 1 or 0 for fraudulent or non-fraudulent transactions respectively. For this reason, we need to convert it from numeric to factor. We can do this using the following code:

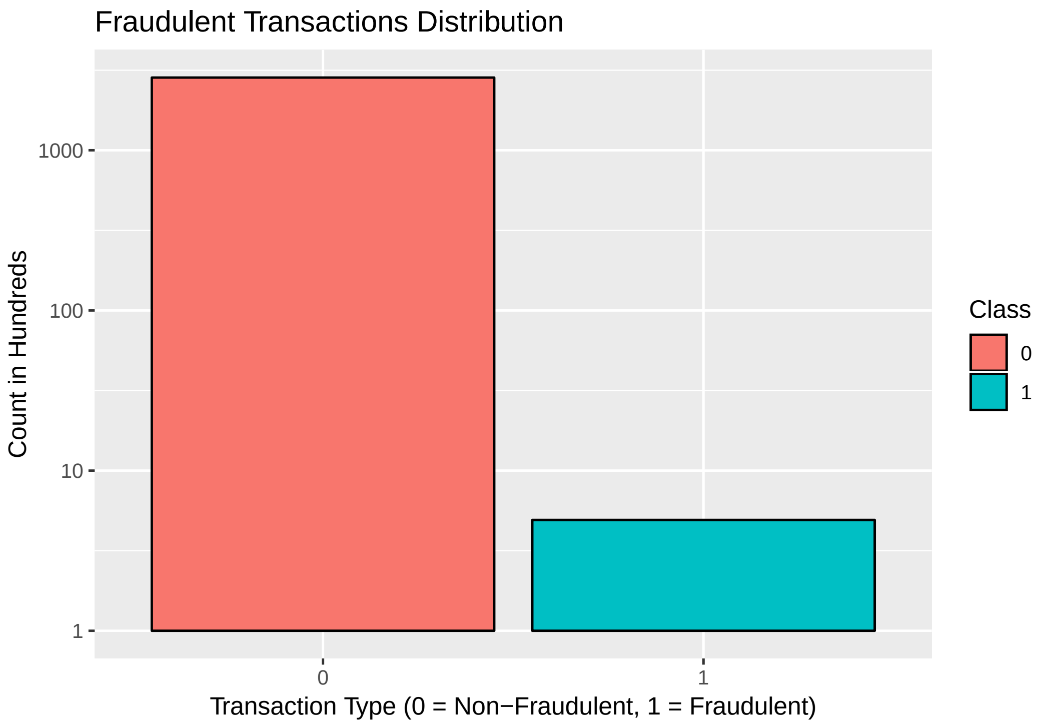
```
cc_dataset <- cc_dataset %>% mutate(Class = as.factor(cc_dataset$Class))

class(cc_dataset$Class)
```

```
## [1] "factor"
```

Next, we are going to explore the data with some plots. We will start by cheking out the distribution of the `Class` variable. We can do that with the follwing code:

```
cc_dataset %>%
  count(Class) %>%
  ggplot(aes(x = Class, y = n/100, fill = Class)) +
  geom_col(col = "Black") +
  scale_y_log10() +
  labs(title = "Fraudulent Transactions Distribution",
       x = "Transaction Type (0 = Non-Fraudulent, 1 = Fraudulent)",
       y = "Count in Hundreds")
```



We can inmediately see a huge difference on the data distribution. Because of this, we will now check how many non-fraudulent transactions and how many fraudulent transactions we have in our data.

```r
class_dist <- cc_dataset %>%
               group_by(Class) %>%
               summarize(Count = n())
class_dist %>% knitr::kable()
```

Class	Count
0	284315
1	492

The previous result tells us that we are facing a data set with a massively unbalanced distribution on the `Class` column. Because of this, we are going to take a new approach in our data exploration. In order to avoid creating 30 plots to check the distribution of each variable relative to the Class column, we are going to group each variable by the type of the observation to then compute the average on each group, then, we will visualize the distribution of the ones that have the biggest difference on the averages.

```r
#Because of formatting purposes, we will not print out this object

vars_grouped_avgs <- cc_dataset %>%
                      group_by(Class) %>%
                      summarize_all(list(mean))

#We will drop the class column as it is a factor and we cannot perform operations with it

vars_diff <- vars_grouped_avgs[,-1]

#Then, we will calculate the difference bewtween values and sort them

diff <- abs(vars_diff[1,] - vars_diff[2,])
diff <- sort(diff, decreasing = TRUE)
diff
```

```
##      Time  Amount       V3      V14      V17      V12      V10       V7
## 1 14091.4 33.9203 7.045452 6.983787 6.677371 6.270225 5.686707 5.578368
##         V1       V4     V16      V11       V2       V5       V9      V18
## 1 4.780206 4.549889 4.14711 3.806749 3.630049 3.156678 2.585589 2.250195
##         V6       V21       V19        V8       V20     V27      V13
## 1 1.400155 0.7148232 0.6818372 0.5716234 0.3729637 0.17087 0.109523
##         V24        V15        V28       V26        V25        V23
## 1 0.1053122 0.09308956 0.07579823 0.0517375 0.04152061 0.04037772
##          V22
## 1 0.01407319
```

Now, as we can see, the biggest differences were on the `Time` and `Amount` variables, but when we look at those entries on the `vars_grouped_avgs` data frame, we notice that we might have a very similar distribution as the difference between the averages on those cases is not that large when we take into account the magnitude of the values.

```
vars_grouped_avgs[c("Time", "Amount")]
```

```
## # A tibble: 2 x 2
##      Time Amount
##     <dbl>  <dbl>
## 1 94838.   88.3
## 2 80747.  122.
```

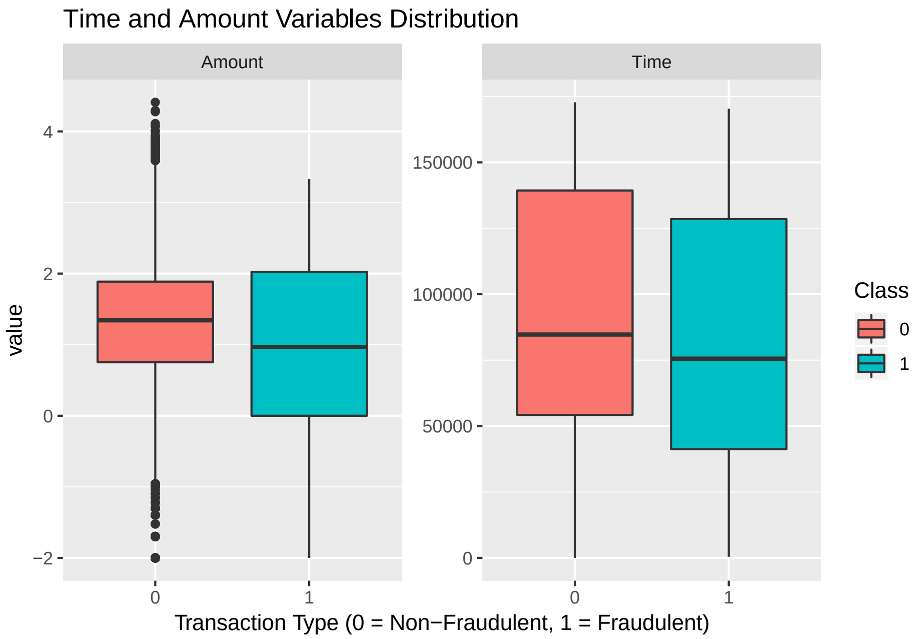
We can check if our hyphothesis is correct by plotting those variables and compare the actual distribution when grouped by the transaction type.

```
#We will transform Amount variable to log10 values and tidy up the data

time_amount_tidy <- cc_dataset[c("Time", "Amount", "Class")] %>%
  mutate(Amount = if_else(Amount != 0,log10(Amount), Amount)) %>%
  gather(-Class, key = "Variable", value = "Values")

#To then plot it

time_amount_tidy %>%
  ggplot(aes(x = Class, y = Values, fill = Class)) +
  facet_wrap(~Variable, scales = "free") +
  geom_boxplot() +
  labs(title = "Time and Amount Variables Distribution",
       x = "Transaction Type (0 = Non-Fraudulent, 1 = Fraudulent)",
       y = "value")
```



As we can see, our hyphothesis was correct as we see a very similar distribution on both variables when we plot them grouped by the transaction type. This fact makes them have less relevance as predictors, so we will drop them from our vector.

```
diff <- diff[-which(names(diff) %in% c("Time", "Amount"))]
```

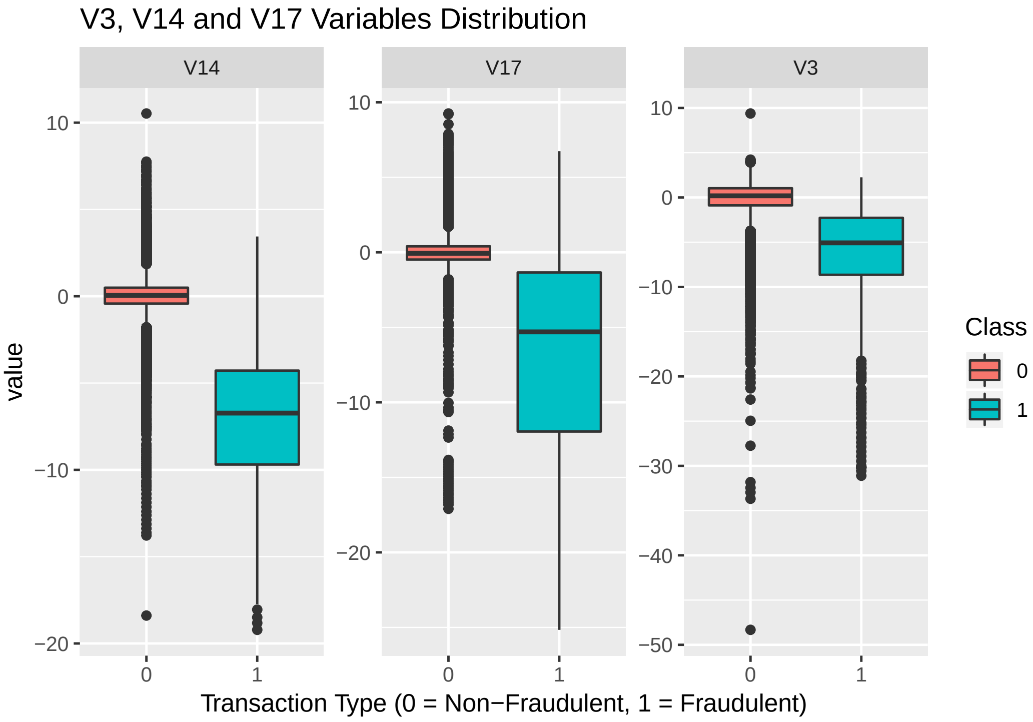
Once we have updated the `diff` vector, we will make a plot comparing the value distribution of the new top 3 variables we got from our calculation, once again, dividing the data into fraudulent and non-fraudulent transactions:

```
#We will tidy up the data first

tidy_data <- cc_dataset[c("V3", "V14", "V17", "Class")] %>%
  gather(-Class, key = "Variable", value = "Values")

#And then plot it

tidy_data %>% ggplot(aes(x = Class, y = Values, fill = Class)) +
  facet_wrap(~Variable, scales = 'free') +
  geom_boxplot() +
  labs(title = "V3, V14 and V17 Variables Distribution",
       x = "Transaction Type (0 = Non-Fraudulent, 1 = Fraudulent)",
       y = "value")
```



As we can tell from the previous plots we generated, the Interquartile Ranges of the data distribution when we group by transaction type are well separated from each other, meaning that we can in fact have some predictive power from those variables, validating our initial approach of calculating the variable averages difference when grouped by transaction type.

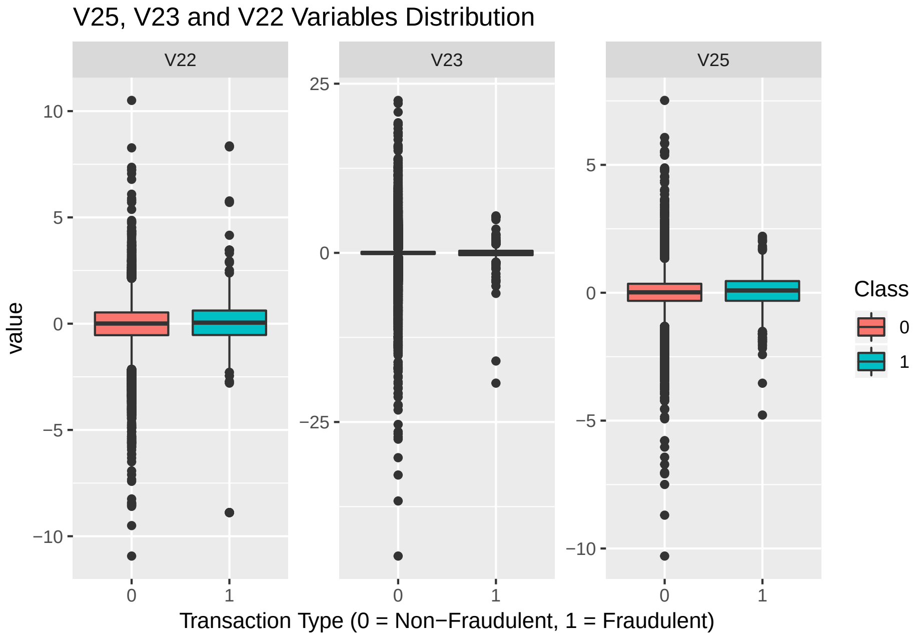
Now, another study we can perform with our data to corroborate that our approach is correct, is to make the same plots but this time with the last 3 variables from our `diff` vector and see if we in fact get a very similar distribution, meaning that those variables have less predictive power. We can perform this with the following code:

```
#We will again tidy up the data first

tidy_data <- cc_dataset[c("V25", "V23", "V22", "Class")] %>%
  gather(-Class, key = "Variable", value = "Values")

#And then plot it

tidy_data %>% ggplot(aes(x = Class, y = Values, fill = Class)) +
  facet_wrap(~Variable, scales = 'free') +
  geom_boxplot() +
  labs(title = "V25, V23 and V22 Variables Distribution",
       x = "Transaction Type (0 = Non-Fraudulent, 1 = Fraudulent)",
       y = "value")
```



As we can see, our approach seems to hold up, as the data on the last 3 variables on our vector have a very similar Interquartile Range, therefore, providing low predictive power. Lastly on this section, we will perform some aditional data wrangling prior starting our modeling phase, more specifically, we will create a training set as well as a test set. We can do this with the following code:

```
y <- cc_dataset$Class

#We will use 70% of the data for training and 30% for testing

train_index <-  createDataPartition(y, times = 1, p = 0.7, list = FALSE)

train_set <- cc_dataset %>% slice(train_index)
test_set <- cc_dataset %>% slice(-train_index)
```

4 Modeling

4.1 General Approach

As we noticed on the Data Exploration phase, we are dealing with a very unbalanced data distribution, so evaluating our model just based on accuracy is not viable because if we predict all transactions as non-fraudulent (which would cost billions in losses to the banking industry), we would get an accuracy of $\sim 99.83\%$. Taking this into account, we can also evaluate our model using the Sensitivity value (or True Positive Rate), as our final goal is to predict as much true fraudulent transactions as we can.

We will focus on achieving a $Sensitivity >= 0.95$ while maintaining also a $Specificity >= 0.95$ so we also avoid alerting about too many false positives. Despite the low viability of predicting all transactions as non-fraudulent, we can use this very simplistic approach as our baseline to evaluate the ongoing models.

To calculate all the metrics for this approach, we will create a Confusion Matrix where we predict all transactions as non-fraudulent. This gives us a potential walkaround to get more informatives results, as the Confusion Matrix also calculates a balanced accuracy that takes into account the data prevalence, so we can also use that value on our metrics. Taking this into account, we will also be focucing on ahcieving a $BalancedAccuracy >= 0.95$. We will store our results into a new chart.

```r
all_nonfraud <- rep(0, nrow(test_set)) %>% factor(levels = c("0", "1"))

cm_all_nonfraud <- confusionMatrix(data = all_nonfraud,
                                   reference = test_set$Class,
                                   positive = "1")

base_accuracy <- cm_all_nonfraud$overal["Accuracy"]
base_b_accuracy <- cm_all_nonfraud$byClass["Balanced Accuracy"]
base_sensitivity <- cm_all_nonfraud$byClass["Sensitivity"]
base_specificity <- cm_all_nonfraud$byClass["Specificity"]

metrics <- c("Model", "Accuracy", "Balanced Accuracy", "Sensitivity", "Specificity")

results <- data.frame(Model = "All Non-Fraudulent",
                      Accuracy = unname(base_accuracy),
                      "Balanced Accuracy" = unname(base_b_accuracy),
                      Sensitivity = unname(base_sensitivity),
                      Specificity = unname(base_specificity),
                      stringsAsFactors = FALSE)
colnames(results) <- metrics
results %>% knitr::kable()
```

Model	Accuracy	Balanced Accuracy	Sensitivity	Specificity
All Non-Fraudulent	0.9982795	0.5	0	1

As we can see, including the balanced accuracy in our metrics gives us a way more informative result when compared to just the accuracy alone. It takes into account the data prevalence, therefore, sorting out our highly unbalanced data. As expected, we got a quite bad value on that metric, still, it will serve as a baseline measure.

We will build initially 3 models based on all the variables of our data, then we will choose the best performing model among those and tune-up the variables we will be using as predictors as we clearly saw that some of them have little to no predictive power.

We will use these 3 models to decide which one we will use for the final model:

1. Generalized Linear Model
2. Decision Tree Model
3. kNN Model

4.2 Generalized Linear Model - All Variables

As explained on the General Approach section, we will now train a *GLM* model using all the available variables to get a sense of the model performance we would get if we decide to use a Generalized Linear Model as a final model, then, we will store our results.

Note that we will get this warning message: `fitted probabilities numerically 0 or 1 occurred`, this is due to the amount of predictors we are using, and as we are not going to consider in this case the Coefficients we get from our fitted model, we can disregard the warning.

```
fit_glm_allvar <- glm(Class ~ ., data = train_set, family = "binomial")
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
predict_glm_allvar <- predict(fit_glm_allvar, test_set, type = "response")

yhat_glm_allvar <- if_else(predict_glm_allvar > 0.5, 1, 0) %>% factor()

cm_glm_allvar <- confusionMatrix(data = yhat_glm_allvar, reference = test_set$Class,
                                 positive = "1")

glm_allvar_acc <- cm_glm_allvar$overall["Accuracy"]
glm_allvar_b_acc <- cm_glm_allvar$byClass["Balanced Accuracy"]
glm_allvar_st <- cm_glm_allvar$byClass["Sensitivity"]
glm_allvar_sp <- cm_glm_allvar$byClass["Specificity"]

glm_allvar_results <-  data.frame(Model = "GLM - All Variables",
                                  Accuracy = unname(glm_allvar_acc),
                                  "Balanced Accuracy" = unname(glm_allvar_b_acc),
                                  Sensitivity = unname(glm_allvar_st),
                                  Specificity = unname(glm_allvar_sp),
                                  stringsAsFactors = FALSE)
colnames(glm_allvar_results) <- metrics

results <- bind_rows(results, glm_allvar_results)
results %>% knitr::kable()
```

Model	Accuracy	Balanced Accuracy	Sensitivity	Specificity
All Non-Fraudulent	0.9982795	0.5000000	0.0000000	1.0000000
GLM - All Variables	0.9991807	0.7890687	0.5782313	0.9999062

5 Results

6 Conclusions