Fraudulent or not?

Sari Vesiluoma

17.11.2019

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

```
## Loading required package: tidyverse
```

```
## -- Attaching packages --------------------------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.2.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

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

```
## Loading required package: knitr
```

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

```
## Loading required package: MASS
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```r
library(MASS)
```

Introduction

The goal in this project is to learn how to predict a fraudulent financial transaction. The data used here is called Synthetic Financial Datasets for Fraud Detection generated by the PaySim mobile money simulator (https://www.kaggle.com/ntnu-testimon/paysim1).As described on the web page, the dataset is a synthetic one, generated using the simulator called PaySim. It uses aggregated data from a private dataset to generate a synthetic dataset that resembles the normal operation of transactions and injects malicious behaviour.

PaySim simulates mobile money transactions based on a sample of real transactions extracted from one month of financial logs from a mobile money service implemented in an African country. The synthetic dataset is scaled down 1/4 of the original dataset.

I have downloaded the dataset from the net (the link above) and I have unzipped it to the same folder where my R script and the rmd file are. This csv file is also available in the GitHub repository, where this rmd and r scripts are too.

The dataset, here referred with a variable name fraud_or_not, has the following dimensions

```
## [1] 6362620      11
```

Next, I will analyse the data and split it to training and test sets. I will use different machine learning algorithms to predict which transaction is fraudulent and which not. In this kind of a case the speciality is that the amount of fraudulent transactions is very minor compared to the amount of non-fraudulent transactions, as we will see.

Analysis

Understanding the data

Let's look the data first as is. As can be seen from the summary below, there are e.g. no NA values which would need to be cleaned. Zero values do exist, but those seem to be correct values needed, because the data seems to include different kind of transactios, where different features are relevant and the others might be zero as a value.

```r
summary(fraud_or_not)
```

```
##       step           type               amount           nameOrig
##  Min.   :  1.0   Length:6362620     Min.   :       0   Length:6362620
##  1st Qu.:156.0   Class :character   1st Qu.:   13390   Class :character
##  Median :239.0   Mode  :character   Median :   74872   Mode  :character
##  Mean   :243.4                      Mean   :  179862
##  3rd Qu.:335.0                      3rd Qu.:  208721
##  Max.   :743.0                      Max.   :92445517
##  oldbalanceOrg      newbalanceOrig       nameDest
##  Min.   :       0   Min.   :       0   Length:6362620
##  1st Qu.:       0   1st Qu.:       0   Class :character
##  Median :   14208   Median :       0   Mode  :character
##  Mean   :  833883   Mean   :  855114
##  3rd Qu.:  107315   3rd Qu.:  144258
##  Max.   :59585040   Max.   :49585040
##  oldbalanceDest      newbalanceDest         isFraud
##  Min.   :        0   Min.   :        0   Min.   :0.000000
##  1st Qu.:        0   1st Qu.:        0   1st Qu.:0.000000
##  Median :   132706   Median :   214661   Median :0.000000
##  Mean   :  1100702   Mean   :  1224996   Mean   :0.001291
##  3rd Qu.:   943037   3rd Qu.:  1111909   3rd Qu.:0.000000
##  Max.   :356015889   Max.   :356179279   Max.   :1.000000
##  isFlaggedFraud
```

```
##  Min.   :0.0e+00
##  1st Qu.:0.0e+00
##  Median :0.0e+00
##  Mean   :2.5e-06
##  3rd Qu.:0.0e+00
##  Max.   :1.0e+00
```

```r
str(fraud_or_not)
```

```
## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 6362620 obs. of  11 variables:
##  $ step          : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ type          : chr  "PAYMENT" "PAYMENT" "TRANSFER" "CASH_OUT" ...
##  $ amount        : num  9840 1864 181 181 11668 ...
##  $ nameOrig      : chr  "C1231006815" "C1666544295" "C1305486145" "C840083671" ...
##  $ oldbalanceOrg : num  170136 21249 181 181 41554 ...
##  $ newbalanceOrig: num  160296 19385 0 0 29886 ...
##  $ nameDest      : chr  "M1979787155" "M2044282225" "C553264065" "C38997010" ...
##  $ oldbalanceDest: num  0 0 0 21182 0 ...
##  $ newbalanceDest: num  0 0 0 0 0 ...
##  $ isFraud       : num  0 0 1 1 0 0 0 0 0 0 ...
##  $ isFlaggedFraud: num  0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   step = col_double(),
##   ..   type = col_character(),
##   ..   amount = col_double(),
##   ..   nameOrig = col_character(),
##   ..   oldbalanceOrg = col_double(),
##   ..   newbalanceOrig = col_double(),
##   ..   nameDest = col_character(),
##   ..   oldbalanceDest = col_double(),
##   ..   newbalanceDest = col_double(),
##   ..   isFraud = col_double(),
##   ..   isFlaggedFraud = col_double()
##   .. )
```

```r
fraud_or_not %>% head()
```

```
## # A tibble: 6 x 11
##    step type  amount nameOrig oldbalanceOrg newbalanceOrig nameDest
##   <dbl> <chr>  <dbl> <chr>            <dbl>          <dbl> <chr>
## 1     1 PAYM~  9840. C123100~        170136        160296. M197978~
## 2     1 PAYM~  1864. C166654~         21249         19385. M204428~
## 3     1 TRAN~   181  C130548~           181             0  C553264~
## 4     1 CASH~   181  C840083~           181             0  C389970~
## 5     1 PAYM~ 11668. C204853~         41554         29886. M123070~
## 6     1 PAYM~  7818. C900456~         53860         46042. M573487~
## # ... with 4 more variables: oldbalanceDest <dbl>, newbalanceDest <dbl>,
## #   isFraud <dbl>, isFlaggedFraud <dbl>
```

The data has 11 columns which are:

Table 1: Explanations of the features

feature	expl
step	Maps a unit of time in the real world. 1 step is 1 hour of time. Total steps 744 (30 days simulation).

feature	expl
type	CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.
amount	Amount of the transaction in local currency.
nameOrig	Customer who started the transaction
oldbalanceOrg	Initial balance before the transaction
newbalanceOrig	New balance after the transaction
nameDest	Customer who is the recipient of the transaction
oldbalanceDest	Initial balance recipient before the transaction
newbalanceDest	New balance recipient after the transaction
isFraud	Transactions made by the fraudulent agents inside the simulation
isFlaggedFraud	An illegal attempt in this dataset is an attempt to transfer more than 200.000 in a single transaction.

The feature isFraud refers to a fraudulent transaction and is the "lable" we are predicting. The feature isFlaggedFraud refers to a transaction, which is a suspect for a fraud because it is an attempt to transfer a bigger amount than 200 000. The amount of fraudulent transactions is very small compared to the amount of non-fraudulent transactions.

When studying the data, there seem to be two variables which may have a potential challenge. nameOrig and nameDest are both char having many values and in some algorithms those would be treated as factors, actually as really many of those. Let's first check the amount of unique values in these.

```
n_distinct(fraud_or_not$nameOrig)
```

```
## [1] 6353307
```

```
n_distinct(fraud_or_not$nameDest)
```

```
## [1] 2722362
```

Like can be seen from above, the amount of unique values of name Orig is very close to the amount of rows in this data. Let's remove this field, because it does not have much explanatory value.

```
fraud_or_not <- subset(fraud_or_not, select = -c(nameOrig))
```

The amount of nameDest is smaller even though being still quite high. Clearly, there would be too many factors based on this feature. The format of the values seems to have first either letter C or M and then a number. Let's replace the current values with a feature having only the first letter.

```
fraud_or_not <- fraud_or_not %>% mutate(dest = str_sub(nameDest, 1, 1))
fraud_or_not <- subset(fraud_or_not, select = -c(nameDest))
n_distinct(fraud_or_not$dest)
```

```
## [1] 2
```

Let's now check the amount of fraudulent transactions among this data. It is:

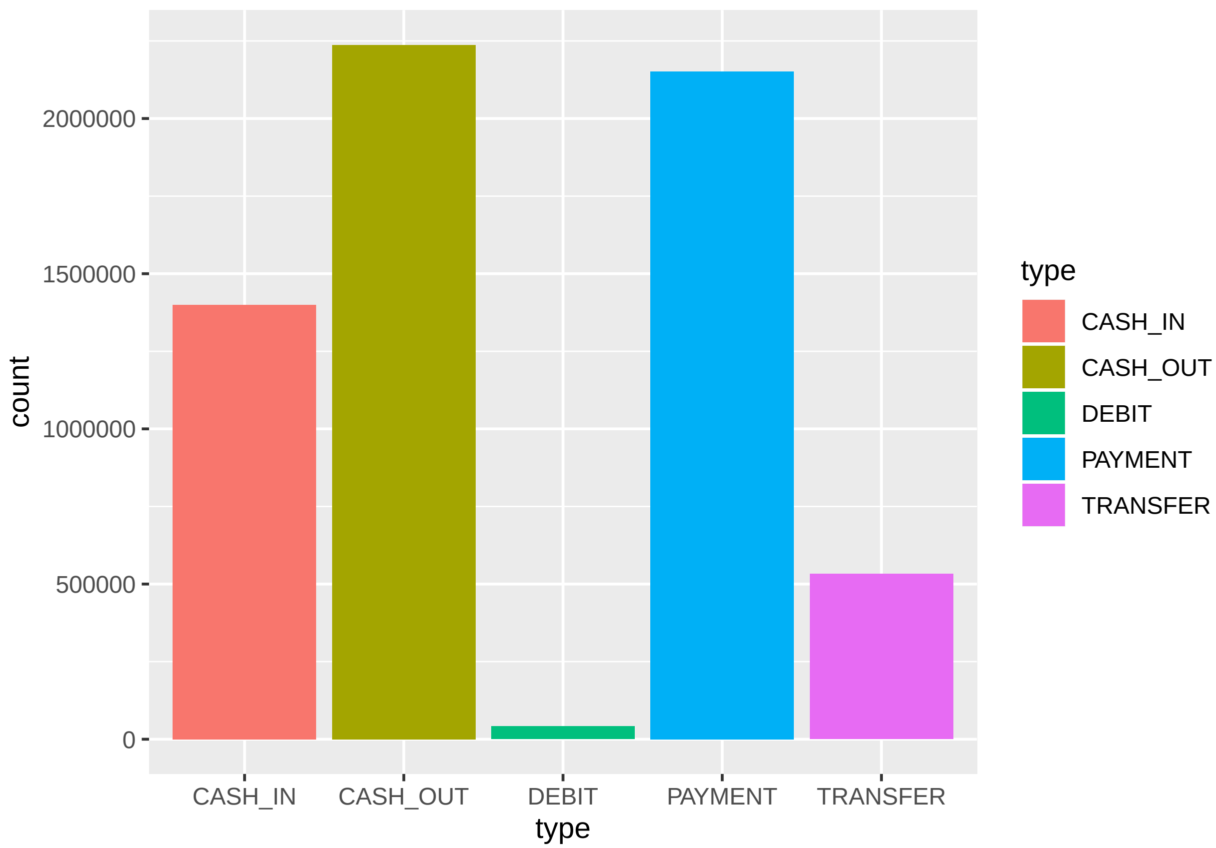
```
mean(fraud_or_not$isFraud)
```

```
## [1] 0.00129082
```

This means that 99.870918 % of the transactions are non-fraudulent. We would get a very high accuracy if predicting that a transaction is always non-fraudulent.

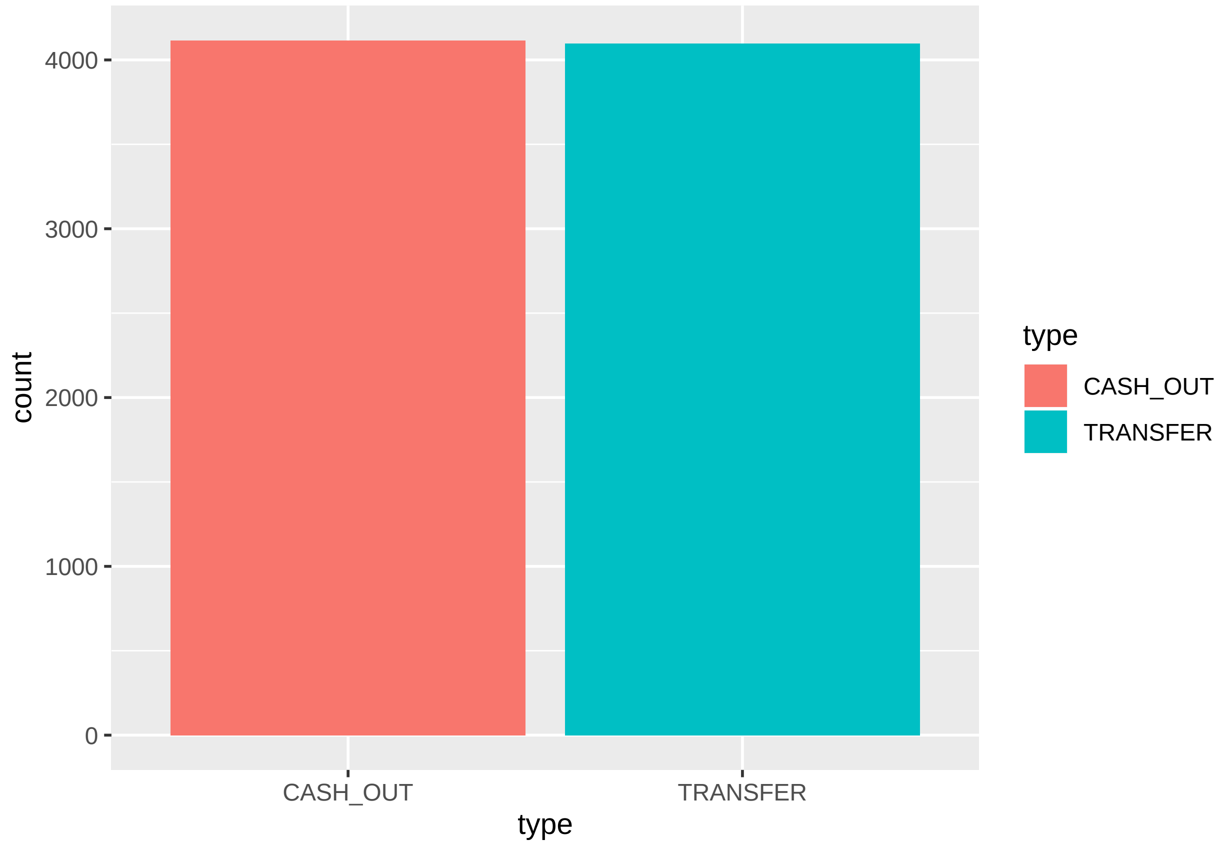
The amount of transactions per type is described at the next histogram. The biggest amount being CASH_OUT transactions and the smallest amount being DEBIT transactions.

```
fraud_or_not %>% ggplot(aes(type , fill = type)) + geom_bar()
```



More interesting is actually how the fraudulent transactions are positioned among the transactions. As can be seen from the next histogram, only the categories CASH_OUT and TRANSFER have fraudulent transactions.

```
fraud_or_not %>% filter(isFraud==1) %>%
  ggplot(aes(type , fill = type)) + geom_bar()
```



Because we know that fraudulent transactions are present only in these two transaction type categories, we will remove the data of all other categories to make the processing smoother.

```
fraud_or_not <- fraud_or_not %>% filter(type == "CASH_OUT" | type == "TRANSFER")
table(fraud_or_not$type)
```

```
##
## CASH_OUT TRANSFER
##  2237500   532909
```

After this clean-up, it looks that the created feature dest has anymore one value, meaning that it will not contribute in predicting. Let's remove it and let's check what we have left now.

```
table(fraud_or_not$dest)
```

```
##
##       C
## 2770409
```

```
fraud_or_not <- subset(fraud_or_not, select = -c(dest))
str(fraud_or_not)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    2770409 obs. of  9 variables:
##  $ step          : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ type          : chr  "TRANSFER" "CASH_OUT" "CASH_OUT" "TRANSFER" ...
##  $ amount        : num  181 181 229134 215310 311686 ...
##  $ oldbalanceOrg : num  181 181 15325 705 10835 ...
##  $ newbalanceOrig: num  0 0 0 0 0 ...
##  $ oldbalanceDest: num  0 21182 5083 22425 6267 ...
```

```
##  $ newbalanceDest: num  0 0 51513 0 2719173 ...
##  $ isFraud       : num  1 1 0 0 0 0 0 0 0 0 ...
##  $ isFlaggedFraud: num  0 0 0 0 0 0 0 0 0 0 ...
```

As we can see, most of the features are now numeric, with one exception. Let's replace the feature type (values: TRANSFER/CASH_OUT) with a numeric feature, where TRANSFER = 1 and CASH_OUT = 2. After that. Let's see what the resulting features look like.

```r
# Replacing chr type with numeric field typeNbr, where TRANSFER = 1 and CASH_OUT = 2
fraud_or_not <- fraud_or_not %>% mutate(typeNbr = ifelse(type == "TRANSFER", 1, 2))
# Removing the type field
fraud_or_not <- subset(fraud_or_not, select = -c(type))
str(fraud_or_not)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    2770409 obs. of  9 variables:
##  $ step          : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ amount        : num  181 181 229134 215310 311686 ...
##  $ oldbalanceOrg : num  181 181 15325 705 10835 ...
##  $ newbalanceOrig: num  0 0 0 0 0 ...
##  $ oldbalanceDest: num  0 21182 5083 22425 6267 ...
##  $ newbalanceDest: num  0 0 51513 0 2719173 ...
##  $ isFraud       : num  1 1 0 0 0 0 0 0 0 0 ...
##  $ isFlaggedFraud: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ typeNbr       : num  1 2 2 1 1 2 2 2 2 1 ...
```

The data includes now totally 6,3 M rows. That is quite much for this project to be run on an ordinary laptop. The amount of data will need to be dramatically reduced. Because the most critical thing would be to find the fraudulent transactions and the amount of those is very minor, let's include all those to the reduced set. In addition, I will include 100 000 lines of non-fraudulent actions randomly selected.

```r
# Data still too big to be efficiently processed.
# Amount of fraudulent transactions - all to be included
reduced_set <- fraud_or_not %>% filter(isFraud == 1)
dim(reduced_set)
```

```
## [1] 8213    9
```

```r
# Amount of non-fraud
non_fraud <- fraud_or_not %>% filter(isFraud == 0)
dim(non_fraud)
```

```
## [1] 2762196       9
```

```r
#  a Select a sample of 100000 from the non_fraud
set.seed(14, sample.kind = "Rounding")
```

```
## Warning in set.seed(14, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```r
non_fraud_selected <- sample_n(non_fraud, 100000)
dim(non_fraud_selected)
```

```
## [1] 100000      9
```

```r
# Combine the selected non-fraud to the reduced set
reduced_set <- rbind(reduced_set, non_fraud_selected)
dim(reduced_set)
```

```
## [1] 108213      9
```

Now, we are ready to start working with machine learning algorithms.

Training and test sets

For the prediction, we need training and test sets. In this project those were created based on the reduced_set data the following way, and checking that the dimensions of the resulting sets are correct and that the prevalence of fraudulent transactions is similar between the cases.

```
# Creating training and test sets - to have big enough temp to have big enough test_set
set.seed(28, sample.kind = "Rounding")
```

```
## Warning in set.seed(28, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
test_index <- createDataPartition(y = reduced_set$isFraud, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- reduced_set[-test_index,]
test_set <- reduced_set[test_index,]
# Checking the dimensions and prevalence of fraud in these sets
dim(train_set)
```

```
## [1] 86570     9
```

```
mean(train_set$isFraud == "1")
```

```
## [1] 0.0759732
```

```
dim(test_set)
```

```
## [1] 21643     9
```

```
mean(test_set$isFraud == "1")
```

```
## [1] 0.07559026
```

Machine learning

Now we are ready to start trials with machine learning algorithms. To start with, we should get a pretty accurate results if guessing that none of the transactions are fraudulent. Let's try.

```
# Algorithm 1: quess that none of the transactions are fraud
# Splitting the training and test data to X and y to make it easier to use those
y_train <- as.factor(train_set$isFraud)
X_train <- as.data.frame(train_set[,which(names(train_set) != "isFraud")])
y_test <- as.factor(test_set$isFraud)
X_test <- as.data.frame(test_set[,which(names(test_set) != "isFraud")])

# Define mu_hat as a predition of no fraudulent transactions
mu_hat <- factor(replicate(length(y_test), 0))
# Changing the levels of mu_hat to match with y_test
levels(mu_hat) <- levels(y_test)

# Check the results with a confusionMatrix - ensure the same levels to be used
cm <- confusionMatrix(data = mu_hat, reference = y_test)
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction     0     1
##          0 20007  1636
##          1     0     0
##
##                Accuracy : 0.9244
##                  95% CI : (0.9208, 0.9279)
##     No Information Rate : 0.9244
##     P-Value [Acc > NIR] : 0.5066
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9244
##          Neg Pred Value :    NaN
##              Prevalence : 0.9244
##          Detection Rate : 0.9244
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

```r
# Store the results of this algorithm
acc <- cm$overall[['Accuracy']]
specif <- sensitivity(mu_hat, y_test, positive= "1")
algorithm_results <- data.frame(method="1: Assume none is fraud",
                                accuracy=acc, specificity=specif)
```

As we can see from the confusion matrix, the accuracy is really high 0.9244097 but, the specificity is 0. In practice, this means, that we have failed in predicting any of the frauds. It is important to notice, that in addition to following accuracy getting better, we have to improve radically the specificity, meaning that we will have as correctly as possible predicted the fraudulent actions being fraudulent.

Next, let's look what kind of a result we will gain using logistic regression.

```r
# Algorithm 2: Logistic regression model
# Training & predicting
train_glm <- train(X_train, y_train, method = "glm")
glm_preds <- factor(predict(train_glm, X_test))
# Define accuracy and specificity
cm <- confusionMatrix(data = glm_preds,reference = y_test)
acc <- cm$overall[['Accuracy']]
acc
```

```
## [1] 0.9773137
```

```r
specif <- sensitivity(glm_preds, y_test, positive= "1")
specif
```

```
## [1] 0.75489
```

```r
# Store the results of this algorithm
algorithm_results <- bind_rows(algorithm_results,
                               data_frame(method="2: Logistic regression",
```

```
                                    accuracy=acc,
                                    specificity=specif))
```

Now also the specificity starts to have some real value. What about using LDA?

```
# Algorithm 3: LDA
# Training & predicting
train_lda <- train(X_train, y_train, method = "lda")
lda_preds <- factor(predict(train_lda, X_test))
# Define accuracy and specificity
cm <- confusionMatrix(data = lda_preds,reference = y_test)
acc <- cm$overall[['Accuracy']]
specif <- sensitivity(lda_preds, y_test, positive= "1")
# Store the results of this algorithm
algorithm_results <- bind_rows(algorithm_results,
                               data_frame(method="3: LDA",
                                          accuracy=acc,
                                          specificity=specif))
```

Not so convincing, so still to predict with k nearest neighbours algorithm. The tuning parameters I originally used included much wider set of values, but here, and based on my earlier results I selected one value below and over the best tune to shorten the time to produce this report. Let's see the results of knn.

´´´{r knn, warning=FALSE} # Algorithm 4: K-nearest neighbors # Training & predicting - trials with k values from 3 to 21 (step: 2) set.seed(1234, sample.kind = “Rounding”) # Original tuning <- data.frame(k=seq(3, 21, 2)) tuning <- data.frame(k=seq(9, 13, 2)) train_knn <- train(X_train, y_train, method = “knn”, tuneGrid = tuning) train_knn$bestTune knn_preds < -factor(predict(train_knn, X_test)) cm < -confusionMatrix(data = knn_preds, reference = y_test) acc < -cm$overall[[‘Accuracy’]] acc specif <- sensitivity(knn_preds, y_test, positive= “1”) specif # Store the results of this algorithm algorithm_results <- bind_rows(algorithm_results, data_frame(method=“4: knn”, accuracy=acc, specificity=specif)) “‘

Still one other algorithm

```
# Algorithm 5: Random forest
# Training & predicting - trials with mtry values from 3 to 9 (step: 2)
set.seed(1234, sample.kind = "Rounding")
```

```
## Warning in set.seed(1234, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
tuning <- data.frame(mtry=seq(3, 9, 2))
train_rf <- train(X_train, y_train, method = "rf", tuneGrid = tuning, importance=TRUE)
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
train_rf$bestTune
```

```
##   mtry
## 4    9
```

```
rf_preds <- factor(predict(train_rf, X_test))
cm <- confusionMatrix(data = rf_preds,reference = y_test)
acc <- cm$overall[['Accuracy']]
specif <- sensitivity(rf_preds, y_test, positive= "1")
# Store the results of this algorithm
algorithm_results <- bind_rows(algorithm_results,
                               data_frame(method="5: Random forest",
                                          accuracy=acc,
                                          specificity=specif))
```

Results

The results over all algorithms used here look the following:

```
algorithm_results
```

```
##                     method  accuracy specificity
## 1 1: Assume none is fraud 0.9244097   0.0000000
## 2  2: Logistic regression 0.9773137   0.7548900
## 3                  3: LDA 0.9482974   0.3178484
## 4        5: Random forest 0.9960264   0.9786064
```

Conclusion