Frauds in credit card in Europe

by Cléber Rodrigo de Souza

This is a project based on the project provided By Luis Otávio of the youtube channel Luis Otávio.Pro (https://www.youtube.com/channel/UCC3Vw7R-fKS-uYXYRhJ983A).

The database if from kaggle (https://www.kaggle.com/mlg-ulb/creditcardfraud) and consists of transactions made by credit cards in September 2013 by european cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

This data were modified by Luis Otavio and it is not equal to eh original database. Here I follow the script and aplication from Luis Otavio and made some changes according to my preferences in analysing. In the Luis Otavio channel he presents a [video-aula] (https://www.youtube.com/watch?v=rSnPkYnaH4E&feature=youtu.be&ab_channel=LuisOtavio) about this example.

The data has 8 columms: ## correct_fill value representing whether the all information were filled correctly ## cpf_died value representing whether the client used a cpf (brazilian Id number) of a died person ## cpf_dirty value representing whether the client used a cpf that has some pendencies related for not paying bills. ## max_value the maximum value the person has ever paid in the credit card. # days_last the days since the person made has last buying using the credit card. ## charge_back represent the number of times person has declared the occurrence of charge back in the credit card, that is the situation when you see a buy you not did. ## amount the amount of cash spend in this transaction. #class: the response variable (1 = fraud; 0 = not fraude).

*** It is important to highlight that the first three values are not in real scales due to data confidence and are presented as PCA axes. This is the original data format.

Let's to start the analysis

```r
data<-read.csv("creditcard.csv", sep=";",h=T)
data<-as.data.frame(data)

######################## To explore the data to understand better their properties


## data dimensions
dim(data)
```

```
## [1] 40900     8
```

```r
## data structure
str(data)
```

```
## 'data.frame':    40900 obs. of  8 variables:
##  $ correct_fill: num  0.302 0.259 0.256 0.199 0.257 ...
```

```
##  $ cpf_died    : num  0.531 0.521 0.535 0.525 0.555 ...
##  $ cpf_dirty   : num  0.34 0.519 0.547 0.522 0.623 ...
##  $ max_value   : num  0.703 0.709 0.708 0.703 0.673 ...
##  $ days_last   : num  0.442 0.434 0.432 0.438 0.446 ...
##  $ charge_back : num  1 0.448 0.38 -0.863 0.403 ...
##  $ amount      : num  149.62 2.69 378.66 123.5 69.99 ...
##  $ class       : int  0 0 0 0 0 0 0 0 0 0 ...
```

```r
## data head
head(data)
```

```
##   correct_fill  cpf_died cpf_dirty max_value days_last charge_back amount class
## 1    0.3023821 0.5306466 0.3404988 0.7025489 0.4416094   1.0000000 149.62     0
## 2    0.2594534 0.5210528 0.5194281 0.7087703 0.4343750   0.4481541   2.69     0
## 3    0.2562972 0.5349955 0.5470077 0.7079462 0.4320216   0.3797796 378.66     0
## 4    0.1989171 0.5252221 0.5216835 0.7034128 0.4380143  -0.8632913 123.50     0
## 5    0.2573706 0.5552957 0.6229744 0.6725004 0.4459672   0.4030339  69.99     0
## 6    0.2310001 0.5134441 0.4170385 0.7090171 0.4477142  -0.1682521   3.67     0
```

```r
## final rows of the data
tail(data)
```

```
##       correct_fill  cpf_died cpf_dirty max_value days_last charge_back amount
## 40895    0.3495842 0.3638466 0.3121882 0.5395094 0.4579225   2.0000000 349.08
## 40896    0.3195138 0.3192978 0.3110417 0.4663978 0.4496852   2.0000000 390.00
## 40897    0.3039375 0.4069713 0.3645325 0.5204752 0.4545814   1.0000000   0.76
## 40898    0.2603837 0.3983464 0.4096831 0.5633265 0.4543762   0.4683084  77.89
## 40899    0.3226435 0.3320195 0.3240543 0.4756353 0.4797408   2.0000000 245.00
## 40900    0.2576308 0.4941904 0.5062947 0.6416821 0.4349825   0.4086700  42.53
##       class
## 40895     1
## 40896     1
## 40897     1
## 40898     1
## 40899     1
## 40900     1
```

```r
## variables names (columns)
names(data)
```

```
## [1] "correct_fill" "cpf_died"     "cpf_dirty"    "max_value"    "days_last"
## [6] "charge_back"  "amount"       "class"
```

```r
## let's to see the frequency of the response value (class)
table(data$class)
```

```
##
##     0     1
## 40408   492
```

Let's to start the data management

```
######### data management

# Let's to scale the non-binary variables (amount and max value)
data$amount<-c(scale(data$amount))
data$max_value<-c(scale(data$max_value))



##### Split data into train and test
set.seed(2452)
data$r<-runif(nrow(data),min=0,max=1)

train<- data[data$r>=0.8,]
test<-data[data$r<0.2,]

dim(train)
```

```
## [1] 8100    9
```

```
dim(test)
```

```
## [1] 8322    9
```

```
table(train$class)
```

```
##
##    0    1
## 7990  110
```

```
table(test$class)
```

```
##
##    0    1
## 8221  101
```

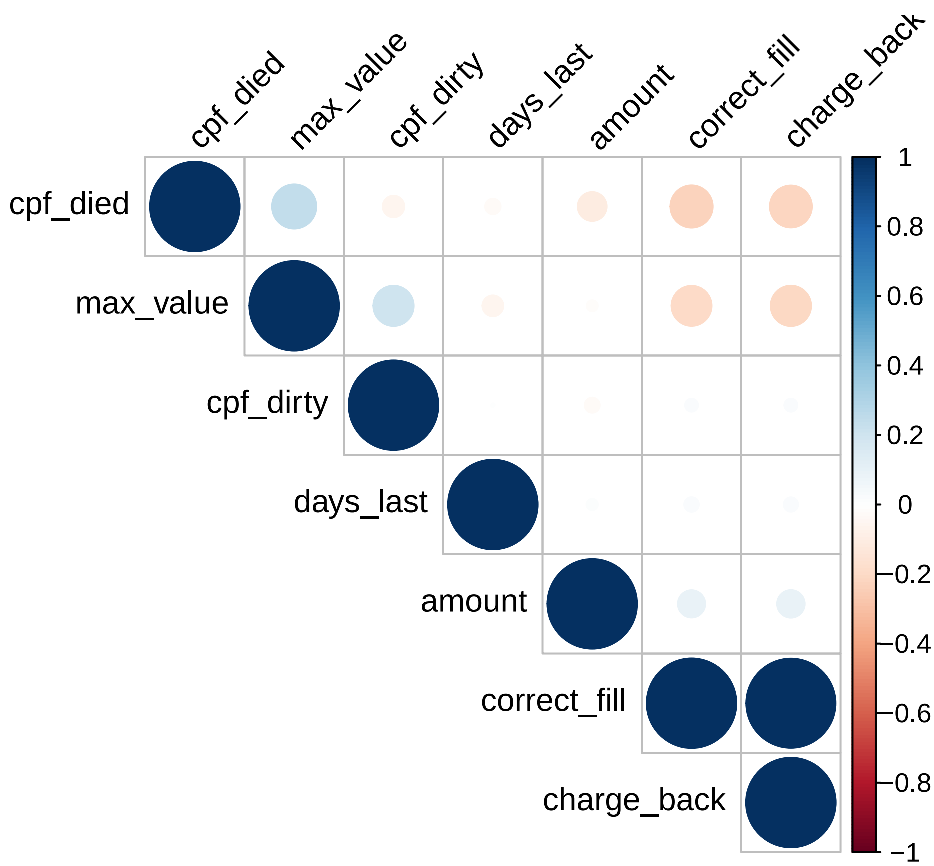
Before start modelling we need to evaluate whether there are correlation between variables and possibly remove any.

```
############ evaluate correlation between variables
corr<-cor(data[,1:7], method = c("pearson"))
corr
```

```
##              correct_fill    cpf_died    cpf_dirty   max_value    days_last
## correct_fill   1.00000000 -0.22160598  0.020202734 -0.19864009  0.025766205
## cpf_died      -0.22160598  1.00000000 -0.057511564  0.24271071 -0.029219726
## cpf_dirty      0.02020273 -0.05751156  1.000000000  0.20113215  0.001192196
```

```
## max_value     -0.19864009  0.24271071  0.201132146  1.00000000 -0.053691608
## days_last      0.02576620 -0.02921973  0.001192196 -0.05369161  1.000000000
## charge_back    0.99077450 -0.21891623  0.020388043 -0.20188525  0.024999556
## amount         0.09208451 -0.10346499 -0.027585016 -0.01477249  0.015280569
##              charge_back      amount
## correct_fill  0.99077450  0.09208451
## cpf_died     -0.21891623 -0.10346499
## cpf_dirty     0.02038804 -0.02758502
## max_value    -0.20188525 -0.01477249
## days_last     0.02499956  0.01528057
## charge_back   1.00000000  0.09399216
## amount        0.09399216  1.00000000
```

```r
library(corrplot)
corrplot(corr, type = "upper", order = "hclust",
         tl.col = "black", tl.srt = 45)
```



We have correlated variables (correct_fill and charge back), but in this case this is not a great problem, since we are interested mainly in the outcome, not in determining each variable importance I recognize correlated variables may imply a problem in estimation, but here I will desconsider this in order to only do the full process of build a model and deploying

Let's build the model

```
######### Adjusting the logistic regression

### First I will obtain a global model with the full variables

log_mod<- glm(class ~correct_fill+cpf_died+cpf_dirty+days_last+max_value+charge_back+amount,train,famil

# But i have correlation between two variables (correct_fill and charge back). I will do two models wit
log_mod2<- glm(class ~correct_fill+cpf_died+cpf_dirty+days_last+max_value+amount,train,family = binomia
log_mod3<- glm(class ~cpf_died+cpf_dirty+days_last+max_value+charge_back+amount,train,family = binomial


## let's compare
library(MuMIn)
model.sel(log_mod,log_mod2,log_mod3)
```

```
## Model selection table
##          (Int)      amn chr_bck crr_fll cpf_did cpf_drt dys_lst max_val
## log_mod2 14.99 -0.10360           12.34  -19.26  -4.849  -28.35 -0.7638
## log_mod3 17.62 -0.09023  0.5377          -19.14  -4.723  -27.81 -0.7844
## log_mod  13.04 -0.10920 -0.4013   21.09  -19.32  -4.880  -28.57 -0.7606
##                    family df   logLik  AICc delta weight
## log_mod2 binomial(logit)  7 -144.726 303.5  0.00  0.504
## log_mod3 binomial(logit)  7 -145.288 304.6  1.12  0.287
## log_mod  binomial(logit)  8 -144.606 305.2  1.76  0.209
## Models ranked by AICc(x)
```

```
##3 the second model is the best, thus i will just use it
## model

summary(log_mod2)
```

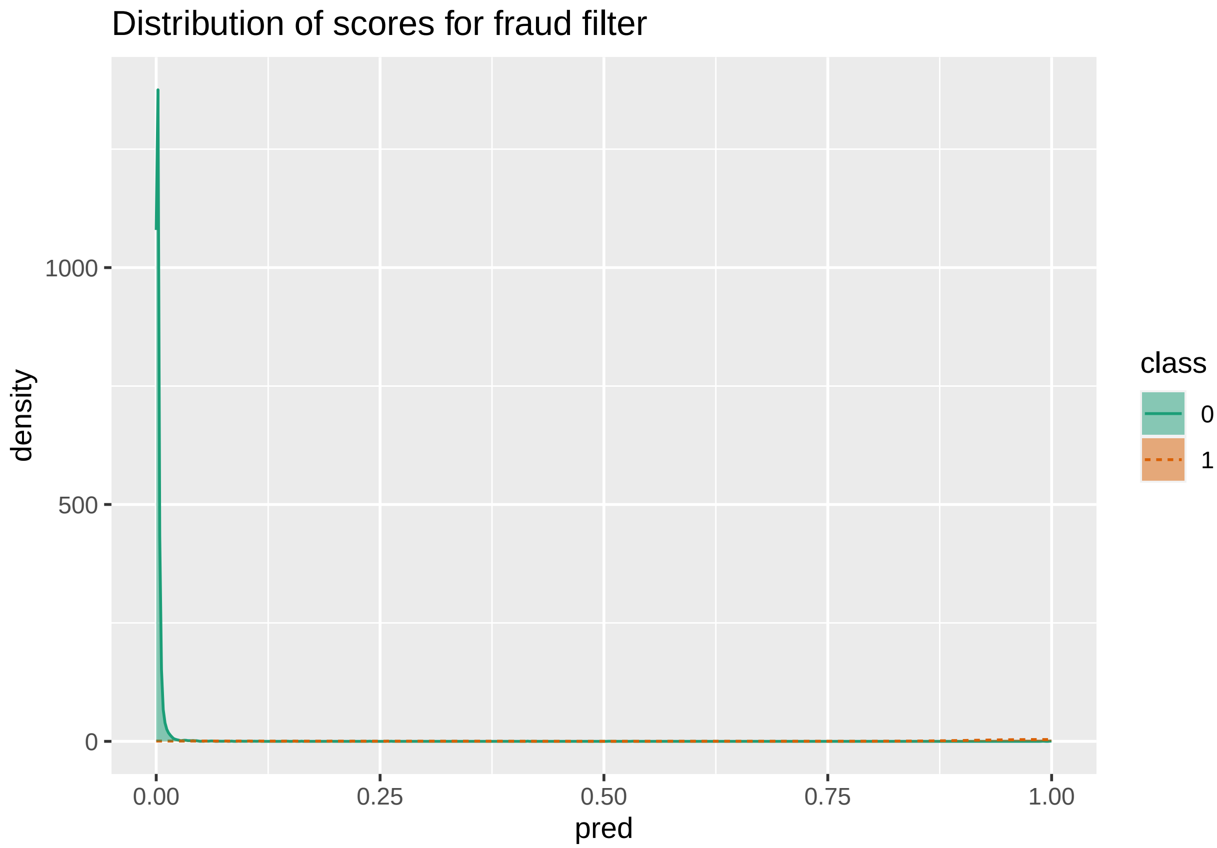
```
##
## Call:
## glm(formula = class ~ correct_fill + cpf_died + cpf_dirty + days_last +
##     max_value + amount, family = binomial, data = train, na.action = "na.fail")
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.1242  -0.0705  -0.0472  -0.0298   4.4236
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   14.9948     4.5675   3.283  0.00103 **
## correct_fill  12.3439     3.0947   3.989 6.64e-05 ***
## cpf_died     -19.2601     4.6509  -4.141 3.46e-05 ***
## cpf_dirty     -4.8494     1.5746  -3.080  0.00207 **
## days_last    -28.3466     5.6342  -5.031 4.87e-07 ***
## max_value     -0.7638     0.1654  -4.617 3.90e-06 ***
## amount        -0.1036     0.1405  -0.737  0.46103
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1164.31  on 8099  degrees of freedom
## Residual deviance:  289.45  on 8093  degrees of freedom
## AIC: 303.45
##
## Number of Fisher Scoring iterations: 9
```

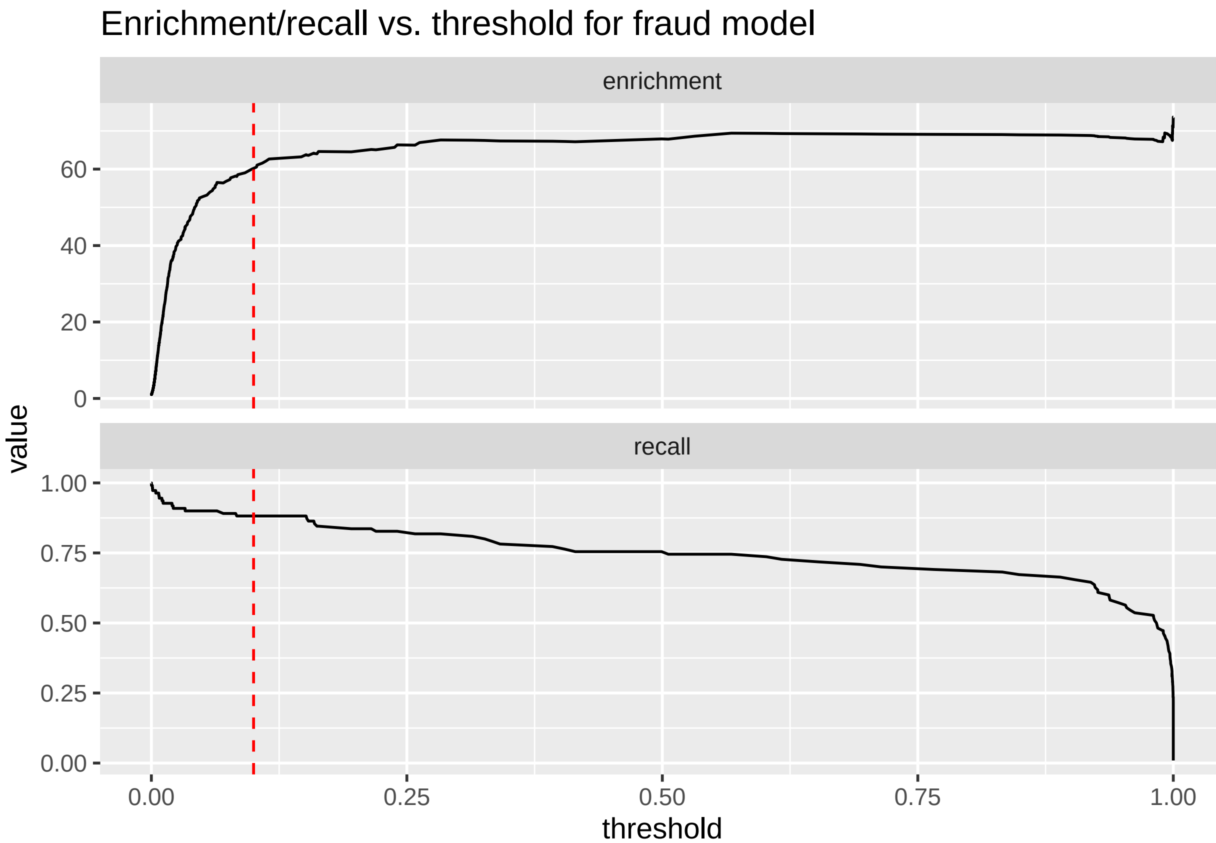
```r
## lets evaluate somethings in global model
#overdispersion
log_mod2$deviance/log_mod2$df.residual
```

```
## [1] 0.03576579
```

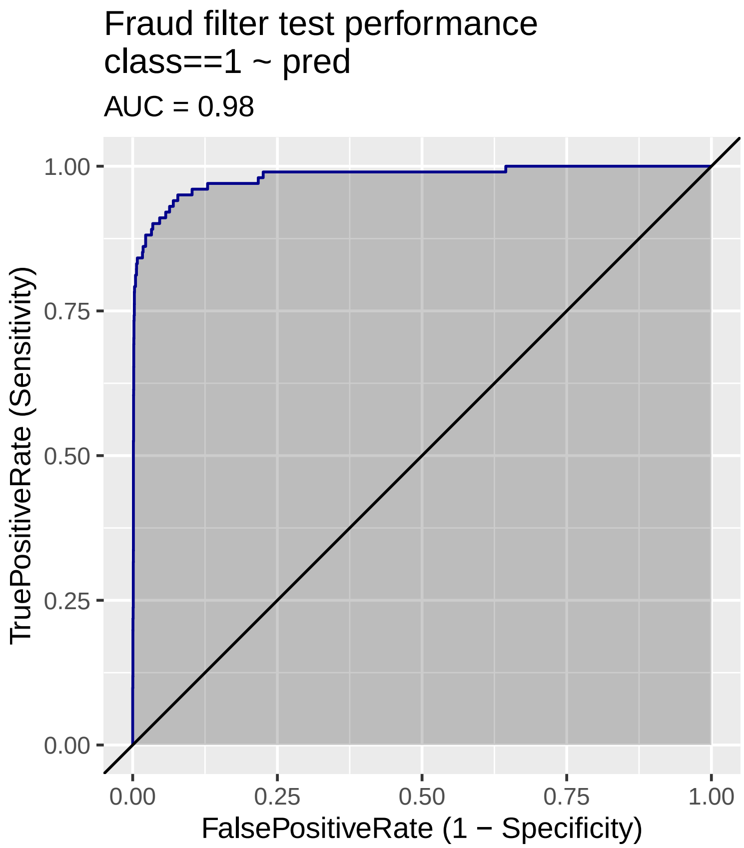
```r
### since it is lower than 1, there is no overdispersion

## compare the model deviance to the null deviance (deviance under a null model)
## since model deviance is lower, our model has a good fit


### let's do the prediction
train$pred<-predict(log_mod2,newdata = train,type="response")
test$pred<-predict(log_mod2,newdata = test,type="response")

### let's do a double plot to see how the prediction are distributed.
## the ideal is to have "1" in the rigth side of plot
library(WVPlots)
library(ggplot2)
DoubleDensityPlot(train,
                  xvar = "pred",
                  truthVar = "class",
                  title = "Distribution of scores for fraud filter")
```




```r
###### one important thing is to determine the limit of the prediction values to differentiate classes
### one option for this is to see the relationship between enrichment and recall
##3 I do this using the training data

### enrichment is the rate of precision to the average rate of positives

plt <- PRTPlot(train, "pred", "class", 1,       # Note: 1
               plotvars = c("enrichment", "recall"),
               thresholdrange = c(0,1),
               title = "Enrichment/recall vs. threshold for fraud model")
plt + geom_vline(xintercept = 0.1, color="red", linetype = 2)
```




```r
## base the results of image i will adopt the threshold od 0.1

### ROC plot: greater the ROC value, better the fitting

ROCPlot(test,
        xvar = "pred",
        truthVar = "class",
        truthTarget = "1",
        title = 'Fraud filter test performance')
```



let's to do a confusion matrix

but it is important to remember confusion matrix is not a goo tool for situations we have unbalanced classes. in this situations the null vs model residues comparison is enough. But let's doing it to see the results

```
confmat <- table(truth = test$class,
                 prediction = ifelse(test$pred > 0.1,
                                     "1", "0"))
print(confmat)
```

```
##       prediction
## truth    0    1
##     0 8195   26
##     1   21   80
```

There are a series of measures to evaluate in the confusion matrix

Accuracy:

how much I got it right within the whole. It is the result of the relationship between the correct predictions and the total of predictions. The correct predictions are diagonal (true negative and true positive). I used this measure here, but remember that accuracy is not a very good measure for unbalanced classes or #with rare events, considering that in these situations the null model already tends to be good

```r
(confmat[1,1]+confmat[2,2])/sum(confmat)
```

```
## [1] 0.9943523
```

Precision and Recall

precision: relationship between what was predicted correct (true positive) and what was predicted as positive. this measure answers the question "if the model says it is fraud, what is the chance of it really being?"

recall: relationship between what was correctly predicted (TP) and what is in fact fraud (FN + TP). this measure answers the question "of all the fraud in my dataset, how many were classified correctly as such?

```r
#precision
confmat[2,2] / (confmat[2,2]+ confmat[1,2])
```

```
## [1] 0.754717
```

```r
#recall
confmat[2,2] / (confmat[2,1]+ confmat[2,2])
```

```
## [1] 0.7920792
```

F1 Score

the F1 score measure is a good measure to use as a comparison metric between classifiers.This is better and easier to look at one measurement, than two. This measure quantifies a trade-off between precision and recall, and is defined as the harmonic average of precision and recall.

F1 is 1 when the classifier has perfect accuracy and recall

```r
precision <- confmat[2,2] / (confmat[2,2]+ confmat[1,2])
recall <- confmat[2,2] / (confmat[2,1]+ confmat[2,2])

(F1 <- 2 * precision * recall / (precision + recall) )
```

```
## [1] 0.7729469
```

Specificity and Sensitivity

Sensitivity equals recall.

Specificity is the rate of true negatives and answers the question: "What fraction of what is not fraud was actually considered how not fraud?

1-specificity is equal to false positive rate, which answers the question: What fraction of non-fraud will be classified as frau by the classifier?

```r
#specificity
confmat[1,1] / (confmat[1,1] + confmat[1,2])
```

```
## [1] 0.9968374
```

Since we finally our model by evaluating its quality, now we will save it to use latter in the app.

```r
summary(log_mod2)
```

```
##
## Call:
## glm(formula = class ~ correct_fill + cpf_died + cpf_dirty + days_last +
##     max_value + amount, family = binomial, data = train, na.action = "na.fail")
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.1242  -0.0705  -0.0472  -0.0298   4.4236
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   14.9948     4.5675   3.283  0.00103 **
## correct_fill  12.3439     3.0947   3.989 6.64e-05 ***
## cpf_died     -19.2601     4.6509  -4.141 3.46e-05 ***
## cpf_dirty     -4.8494     1.5746  -3.080  0.00207 **
## days_last    -28.3466     5.6342  -5.031 4.87e-07 ***
## max_value     -0.7638     0.1654  -4.617 3.90e-06 ***
## amount        -0.1036     0.1405  -0.737  0.46103
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1164.31  on 8099  degrees of freedom
## Residual deviance:  289.45  on 8093  degrees of freedom
## AIC: 303.45
##
## Number of Fisher Scoring iterations: 9
```

```r
gdata::keep(log_mod2,sure=T)
save.image(".RData")
```