detgection des frauds

Herrag Mohammed et El Hadef Ilias

12/27/2021

La détection des transactions frauduleuses est d' une grande importance pour toute société émettrice de cartes de crédit. Nous sommes chargés par une entreprise bien connue de détecter les fraudes potentielles afin que les clients ne soient pas facturés pour les articles qu'ils n'ont pas achetés. L'objectif est donc de construire un classificateur qui indique si une transaction est une fraude ou non.

1.Description de données

```
knitr::opts_chunk$set(warning = TRUE, message = TRUE)
d<- read.csv("/Users/mohammedherrag/python-data-assignment/input/creditcard.csv")
```

```
head(d,10)
```

```
##    Time         V1          V2          V3         V4          V5          V6
## 1     0 -1.3598071 -0.07278117  2.53634674  1.3781552 -0.33832077  0.46238778
## 2     0  1.1918571  0.26615071  0.16648011  0.4481541  0.06001765 -0.08236081
## 3     1 -1.3583541 -1.34016307  1.77320934  0.3797796 -0.50319813  1.80049938
## 4     1 -0.9662717 -0.18522601  1.79299334 -0.8632913 -0.01030888  1.24720317
## 5     2 -1.1582331  0.87773675  1.54871785  0.4030339 -0.40719338  0.09592146
## 6     2 -0.4259659  0.96052304  1.14110934 -0.1682521  0.42098688 -0.02972755
## 7     4  1.2296576  0.14100351  0.04537077  1.2026127  0.19188099  0.27270812
## 8     7 -0.6442694  1.41796355  1.07438038 -0.4921990  0.94893409  0.42811846
## 9     7 -0.8942861  0.28615720 -0.11319221 -0.2715261  2.66959866  3.72181806
## 10    9 -0.3382618  1.11959338  1.04436655 -0.2221873  0.49936081 -0.24676110
##              V7          V8         V9         V10        V11         V12
## 1   0.239598554  0.09869790  0.3637870  0.09079417 -0.5515995 -0.61780086
## 2  -0.078802983  0.08510165 -0.2554251 -0.16697441  1.6127267  1.06523531
## 3   0.791460956  0.24767579 -1.5146543  0.20764287  0.6245015  0.06608369
## 4   0.237608940  0.37743587 -1.3870241 -0.05495192 -0.2264873  0.17822823
## 5   0.592940745 -0.27053268  0.8177393  0.75307443 -0.8228429  0.53819555
## 6   0.476200949  0.26031433 -0.5686714 -0.37140720  1.3412620  0.35989384
## 7  -0.005159003  0.08121294  0.4649600 -0.09925432 -1.4169072 -0.15382583
## 8   1.120631358 -3.80786424  0.6153747  1.24937618 -0.6194678  0.29147435
## 9   0.370145128  0.85108444 -0.3920476 -0.41043043 -0.7051166 -0.11045226
## 10  0.651583206  0.06953859 -0.7367273 -0.36684564  1.0176145  0.83638957
##           V13         V14         V15        V16          V17         V18
## 1  -0.9913898 -0.31116935  1.46817697 -0.4704005  0.207971242  0.02579058
## 2   0.4890950 -0.14377230  0.63555809  0.4639170 -0.114804663 -0.18336127
## 3   0.7172927 -0.16594592  2.34586495 -2.8900832  1.109969379 -0.12135931
## 4   0.5077569 -0.28792375 -0.63141812 -1.0596472 -0.684092786  1.96577500
## 5   1.3458516 -1.11966983  0.17512113 -0.4514492 -0.237033239 -0.03819479
## 6  -0.3580907 -0.13713370  0.51761681  0.4017259 -0.058132823  0.06865315
## 7  -0.7510627  0.16737196  0.05014359 -0.4435868  0.002820512 -0.61198734
## 8   1.7579642 -1.32386522  0.68613250 -0.0761270 -1.222127345 -0.35822157
## 9  -0.2862536  0.07435536 -0.32878305 -0.2100773 -0.499767969  0.11876486
```

```
## 10  1.0068435 -0.44352282  0.15021910  0.7394528 -0.540979922  0.47667726
##            V19         V20          V21          V22         V23         V24
## 1   0.40399296  0.25141210 -0.018306778  0.277837576 -0.11047391  0.06692807
## 2  -0.14578304 -0.06908314 -0.225775248 -0.638671953  0.10128802 -0.33984648
## 3  -2.26185710  0.52497973  0.247998153  0.771679402  0.90941226 -0.68928096
## 4  -1.23262197 -0.20803778 -0.108300452  0.005273597 -0.19032052 -1.17557533
## 5   0.80348692  0.40854236 -0.009430697  0.798278495 -0.13745808  0.14126698
## 6  -0.03319379  0.08496767 -0.208253515 -0.559824796 -0.02639767 -0.37142658
## 7  -0.04557504 -0.21963255 -0.167716266 -0.270709726 -0.15410379 -0.78005542
## 8   0.32450473 -0.15674185  1.943465340 -1.015454710  0.05750353 -0.64970901
## 9   0.57032817  0.05273567 -0.073425100 -0.268091632 -0.20423267  1.01159180
## 10  0.45177296  0.20371145 -0.246913937 -0.633752642 -0.12079408 -0.38504993
##            V25         V26          V27          V28 Amount Class
## 1   0.12853936 -0.18911484  0.133558377 -0.021053053 149.62     0
## 2   0.16717040  0.12589453 -0.008983099  0.014724169   2.69     0
## 3  -0.32764183 -0.13909657 -0.055352794 -0.059751841 378.66     0
## 4   0.64737603 -0.22192884  0.062722849  0.061457629 123.50     0
## 5  -0.20600959  0.50229222  0.219422230  0.215153147  69.99     0
## 6  -0.23279382  0.10591478  0.253844225  0.081080257   3.67     0
## 7   0.75013694 -0.25723685  0.034507430  0.005167769   4.99     0
## 8  -0.41526657 -0.05163430 -1.206921081 -1.085339188  40.80     0
## 9   0.37320468 -0.38415731  0.011747356  0.142404330  93.20     0
## 10 -0.06973305  0.09419883  0.246219305  0.083075649   3.68     0
```

```
print(paste("nembre des lignes:",dim(d)[1]))
```

```
## [1] "nembre des lignes: 284807"
```

```
print(paste("nembre des colonnes:",dim(d)[2]))
```

```
## [1] "nembre des colonnes: 31"
```

```
print(summary(d))
```

```
##       Time              V1                   V2                  V3
##  Min.   :     0   Min.   :-56.40751   Min.   :-72.71573   Min.   :-48.3256
##  1st Qu.: 54202   1st Qu.: -0.92037   1st Qu.: -0.59855   1st Qu.: -0.8904
##  Median : 84692   Median :  0.01811   Median :  0.06549   Median :  0.1799
##  Mean   : 94814   Mean   :  0.00000   Mean   :  0.00000   Mean   :  0.0000
##  3rd Qu.:139320   3rd Qu.:  1.31564   3rd Qu.:  0.80372   3rd Qu.:  1.0272
##  Max.   :172792   Max.   :  2.45493   Max.   : 22.05773   Max.   :  9.3826
##        V4                  V5                   V6                  V7
##  Min.   :-5.68317   Min.   :-113.74331   Min.   :-26.1605   Min.   :-43.5572
##  1st Qu.:-0.84864   1st Qu.:  -0.69160   1st Qu.: -0.7683   1st Qu.: -0.5541
##  Median :-0.01985   Median :  -0.05434   Median : -0.2742   Median :  0.0401
##  Mean   : 0.00000   Mean   :   0.00000   Mean   :  0.0000   Mean   :  0.0000
##  3rd Qu.: 0.74334   3rd Qu.:   0.61193   3rd Qu.:  0.3986   3rd Qu.:  0.5704
##  Max.   :16.87534   Max.   :  34.80167   Max.   : 73.3016   Max.   :120.5895
##        V8                  V9                  V10                 V11
##  Min.   :-73.21672   Min.   :-13.43407   Min.   :-24.58826   Min.   :-4.79747
##  1st Qu.: -0.20863   1st Qu.: -0.64310   1st Qu.: -0.53543   1st Qu.:-0.76249
##  Median :  0.02236   Median : -0.05143   Median : -0.09292   Median :-0.03276
##  Mean   :  0.00000   Mean   :  0.00000   Mean   :  0.00000   Mean   : 0.00000
##  3rd Qu.:  0.32735   3rd Qu.:  0.59714   3rd Qu.:  0.45392   3rd Qu.: 0.73959
##  Max.   : 20.00721   Max.   : 15.59500   Max.   : 23.74514   Max.   :12.01891
##       V12                 V13                 V14                 V15
```

```
##  Min.   :-18.6837   Min.   :-5.79188   Min.   :-19.2143   Min.   :-4.49894
##  1st Qu.: -0.4056   1st Qu.:-0.64854   1st Qu.: -0.4256   1st Qu.:-0.58288
##  Median :  0.1400   Median :-0.01357   Median :  0.0506   Median : 0.04807
##  Mean   :  0.0000   Mean   : 0.00000   Mean   :  0.0000   Mean   : 0.00000
##  3rd Qu.:  0.6182   3rd Qu.: 0.66251   3rd Qu.:  0.4931   3rd Qu.: 0.64882
##  Max.   :  7.8484   Max.   : 7.12688   Max.   : 10.5268   Max.   : 8.87774
##       V16                V17                V18
##  Min.   :-14.12985   Min.   :-25.16280   Min.   :-9.498746
##  1st Qu.: -0.46804   1st Qu.: -0.48375   1st Qu.:-0.498850
##  Median :  0.06641   Median : -0.06568   Median :-0.003636
##  Mean   :  0.00000   Mean   :  0.00000   Mean   : 0.000000
##  3rd Qu.:  0.52330   3rd Qu.:  0.39968   3rd Qu.: 0.500807
##  Max.   : 17.31511   Max.   :  9.25353   Max.   : 5.041069
##       V19                V20                V21
##  Min.   :-7.213527   Min.   :-54.49772   Min.   :-34.83038
##  1st Qu.:-0.456299   1st Qu.: -0.21172   1st Qu.: -0.22839
##  Median : 0.003735   Median : -0.06248   Median : -0.02945
##  Mean   : 0.000000   Mean   :  0.00000   Mean   :  0.00000
##  3rd Qu.: 0.458949   3rd Qu.:  0.13304   3rd Qu.:  0.18638
##  Max.   : 5.591971   Max.   : 39.42090   Max.   : 27.20284
##       V22                V23                V24
##  Min.   :-10.933144   Min.   :-44.80774   Min.   :-2.83663
##  1st Qu.: -0.542350   1st Qu.: -0.16185   1st Qu.:-0.35459
##  Median :  0.006782   Median : -0.01119   Median : 0.04098
##  Mean   :  0.000000   Mean   :  0.00000   Mean   : 0.00000
##  3rd Qu.:  0.528554   3rd Qu.:  0.14764   3rd Qu.: 0.43953
##  Max.   : 10.503090   Max.   : 22.52841   Max.   : 4.58455
##       V25                V26                V27
##  Min.   :-10.29540   Min.   :-2.60455   Min.   :-22.565679
##  1st Qu.: -0.31715   1st Qu.:-0.32698   1st Qu.: -0.070840
##  Median :  0.01659   Median :-0.05214   Median :  0.001342
##  Mean   :  0.00000   Mean   : 0.00000   Mean   :  0.000000
##  3rd Qu.:  0.35072   3rd Qu.: 0.24095   3rd Qu.:  0.091045
##  Max.   :  7.51959   Max.   : 3.51735   Max.   : 31.612198
##       V28                Amount             Class
##  Min.   :-15.43008   Min.   :    0.00   Min.   :0.000000
##  1st Qu.: -0.05296   1st Qu.:    5.60   1st Qu.:0.000000
##  Median :  0.01124   Median :   22.00   Median :0.000000
##  Mean   :  0.00000   Mean   :   88.35   Mean   :0.001728
##  3rd Qu.:  0.07828   3rd Qu.:   77.17   3rd Qu.:0.000000
##  Max.   : 33.84781   Max.   :25691.16   Max.   :1.000000
```

```
d$Amount<- scale(d$Amount)
round(apply(d$Amount, MARGIN = 2, FUN = mean), digits = 5)
```

```
## [1] 0
```

```
apply(d$Amount, MARGIN = 2, FUN = sd)
```

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

```
#il existe aucun valeure manquante
print(sapply(d, function(x)sum(is.na(x))))
```

```
##   Time     V1     V2     V3     V4     V5     V6     V7     V8     V9    V10
##      0      0      0      0      0      0      0      0      0      0      0
```

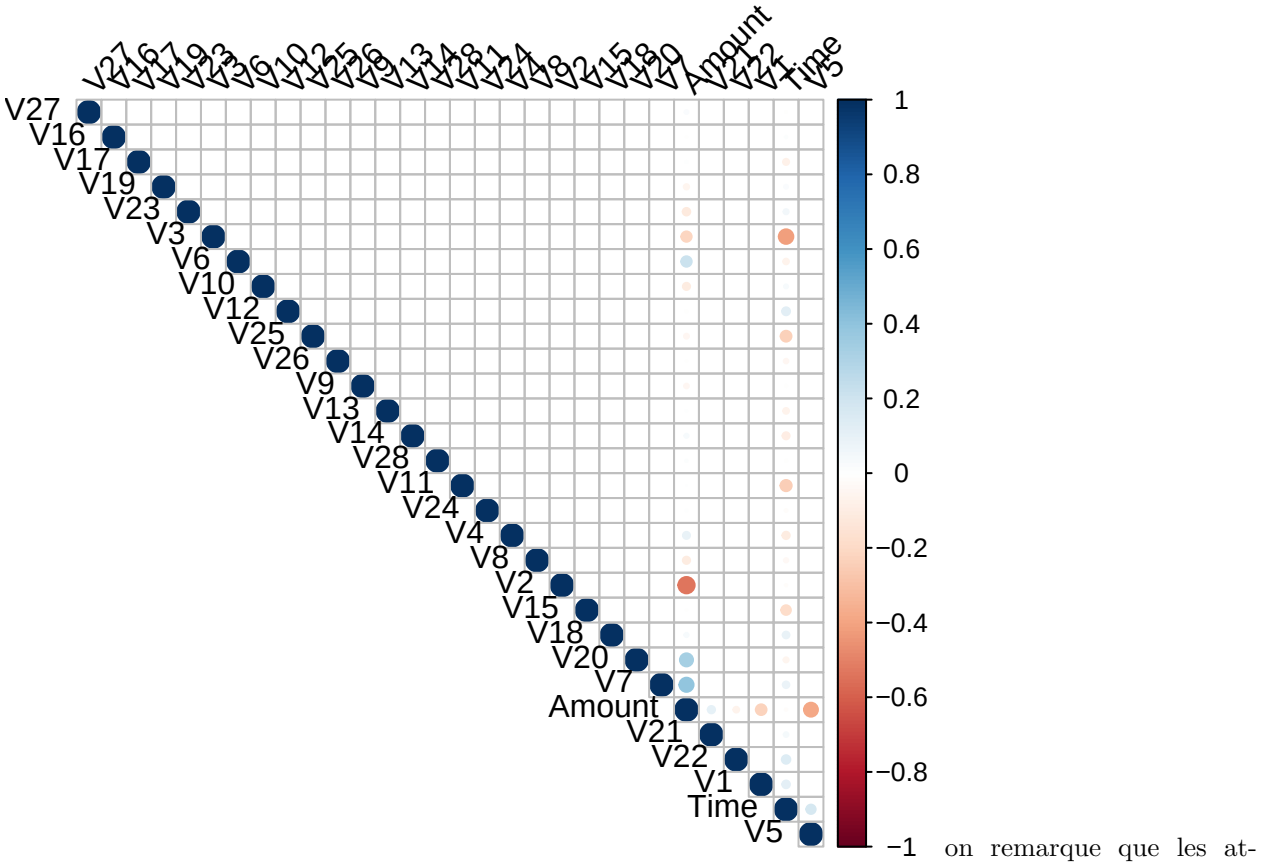
```
##    V11    V12    V13    V14    V15    V16    V17    V18    V19    V20    V21
##      0      0      0      0      0      0      0      0      0      0      0
##    V22    V23    V24    V25    V26    V27    V28 Amount  Class
##      0      0      0      0      0      0      0      0      0
```

2. Analyse exploratoire :

```
library(corrplot)
```

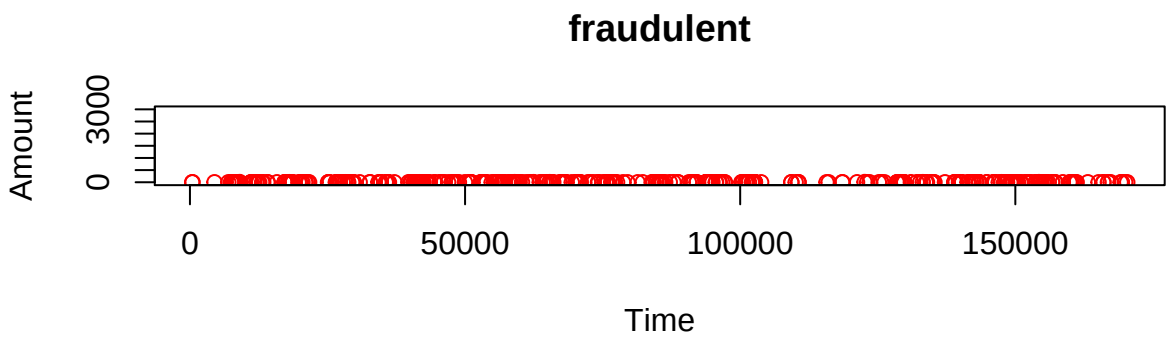
```
## corrplot 0.92 loaded
```

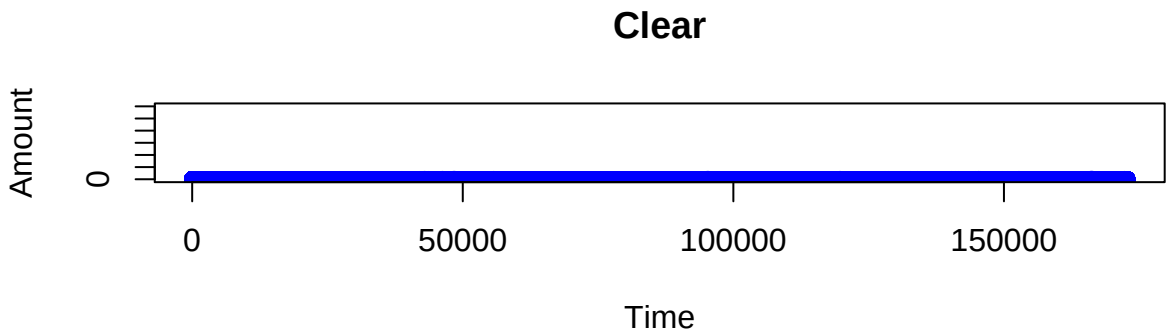
```
corrplot(cor(d[,-31]), type="upper", order="hclust", tl.col="black", tl.srt=45)
```



on remarque que les attribues V1,...,V28 ne sont pas correller entre eux et il existe une corellation entre les (V1,...,V28) et (Time ,Amount,Class)

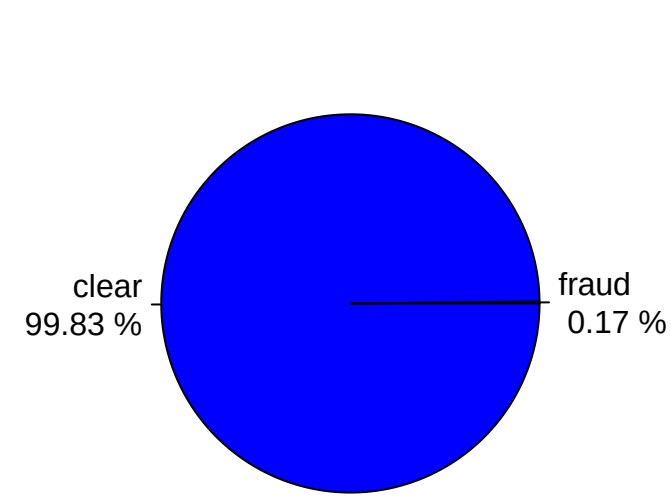
```
d.fraud = d[d$Class==1,]
d.clear = d[d$Class==0,]
par(mfrow=c(2,1))
plot(d.fraud$Time,d.fraud$Amount,col="red",
     ylim = c(0, 3000),xlab="Time",ylab="Amount",main="fraudulent")
plot(d.clear$Time,d.clear$Amount,col="blue",
     ylim = c(0, 30000),xlab="Time",ylab="Amount",main="Clear")
```





Nous avons des informations concernant seulement deux jours. Ainsi, nous ne pouvons tirer aucune conclusion concernant Time auxquels la fréquence des transaction Fraudulent est plus élevée. en remarque que les montante de ransaction Fraudulent ne depasse pas 2000.

```
d.fraud = d[d$Class==1,]
d.clear = d[d$Class==0,]
d.fraud.percent=round(dim(d.fraud)[1]/dim(d)[1]*100,2)
d.clear.percent=round(dim(d.clear)[1]/dim(d)[1]*100,2)

library(plotrix)
pie(c(d.fraud.percent,d.clear.percent),
      labels = c(paste("fraud\n",d.fraud.percent,"%"),paste("clear\n",d.clear.percent,"%")),col = c("re
```



```
d$Class <- as.factor(d$Class)
print(table(d$Class))
```

```
##
##      0      1
## 284315    492
```

le graphique cérculaire montre que les classes sont déséquilibrées. Les transaction Clear (99.8 %) sont plus fréquents que les Fraudulent(0.2 %).

3. pré_processing :

plusieurs étapes sont nécessaires avant de pouvoir travailler avec les algorithmes de machine learning :

a| Utilise la methode Oversampling de SMOTE pour equilibré les données(Clear et Fraudulent) de train.

b| Diviser les données en sous-ensembles de train et de test aléatoires.

```
library(UBL)
```

```
## Loading required package: MBA
```

```
## Loading required package: gstat
```

```
## Loading required package: automap
```

```
## Loading required package: sp
```

```
## Loading required package: randomForest
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
over1 <- RandOverClassif(Class~.,d)
```

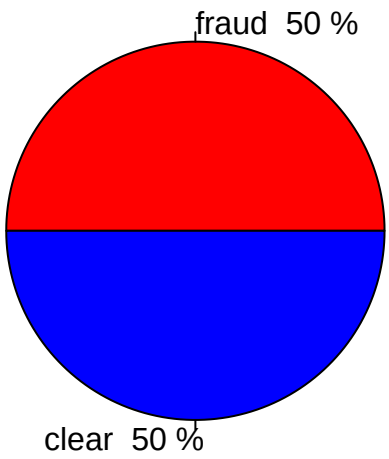
```
over1.fraud = over1[over1$Class==1,]
over1.clear = over1[over1$Class==0,]
over1.fraud.percent=round(dim(over1.fraud)[1]/dim(over1)[1]*100,2)
over1.clear.percent=round(dim(over1.clear)[1]/dim(over1)[1]*100,2)

#library(plotrix)
pie(c(over1.fraud.percent,over1.clear.percent),
      labels = c(paste("fraud\t",over1.fraud.percent,"%"),paste("clear\t",over1.clear.percent,"%")),col
```

```
## Warning in text.default(1.1 * P$x, 1.1 * P$y, labels[i], xpd = TRUE, adj =
## ifelse(P$x < : font width unknown for character 0x9
```

```
## Warning in text.default(1.1 * P$x, 1.1 * P$y, labels[i], xpd = TRUE, adj =
## ifelse(P$x < : font metrics unknown for character 0x9
```

```
## Warning in text.default(1.1 * P$x, 1.1 * P$y, labels[i], xpd = TRUE, adj =
## ifelse(P$x < : font width unknown for character 0x9
```

```
## Warning in text.default(1.1 * P$x, 1.1 * P$y, labels[i], xpd = TRUE, adj =
## ifelse(P$x < : font metrics unknown for character 0x9
```



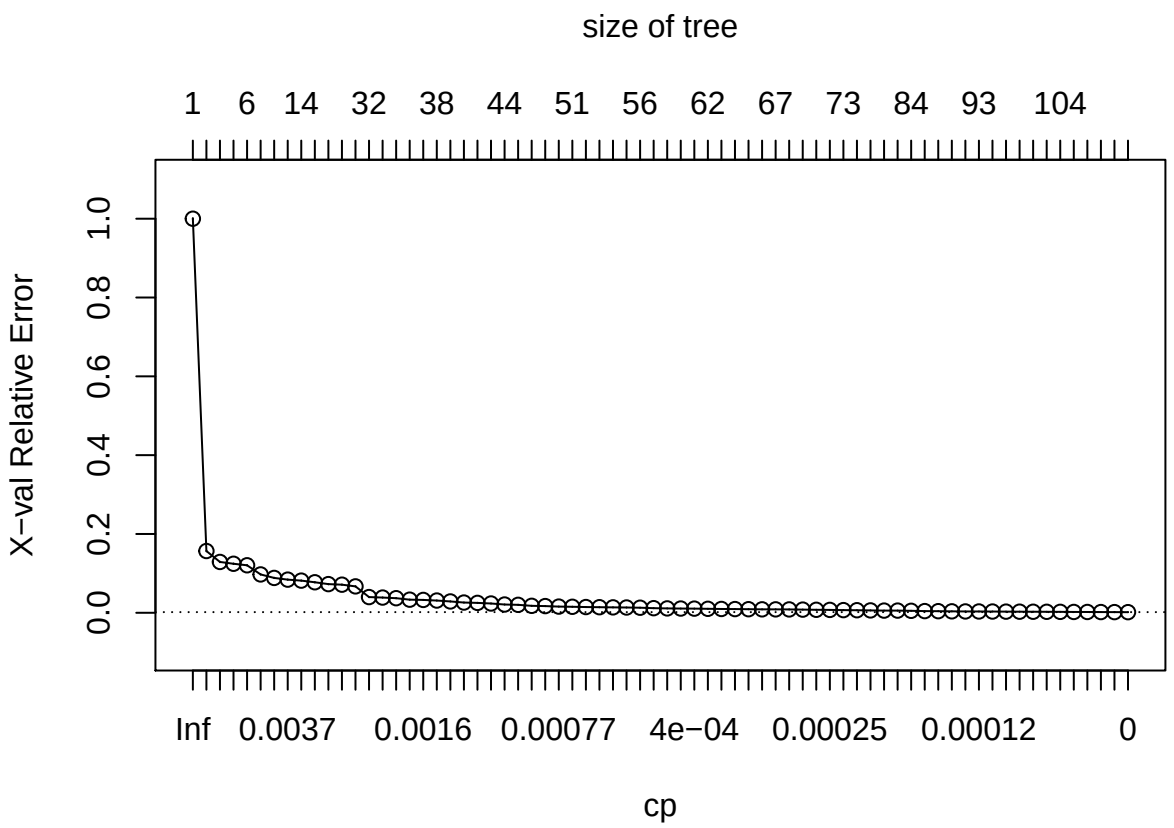

```r
print(table(over1$Class))
```

```
##
##      0      1
## 284315 284315
```

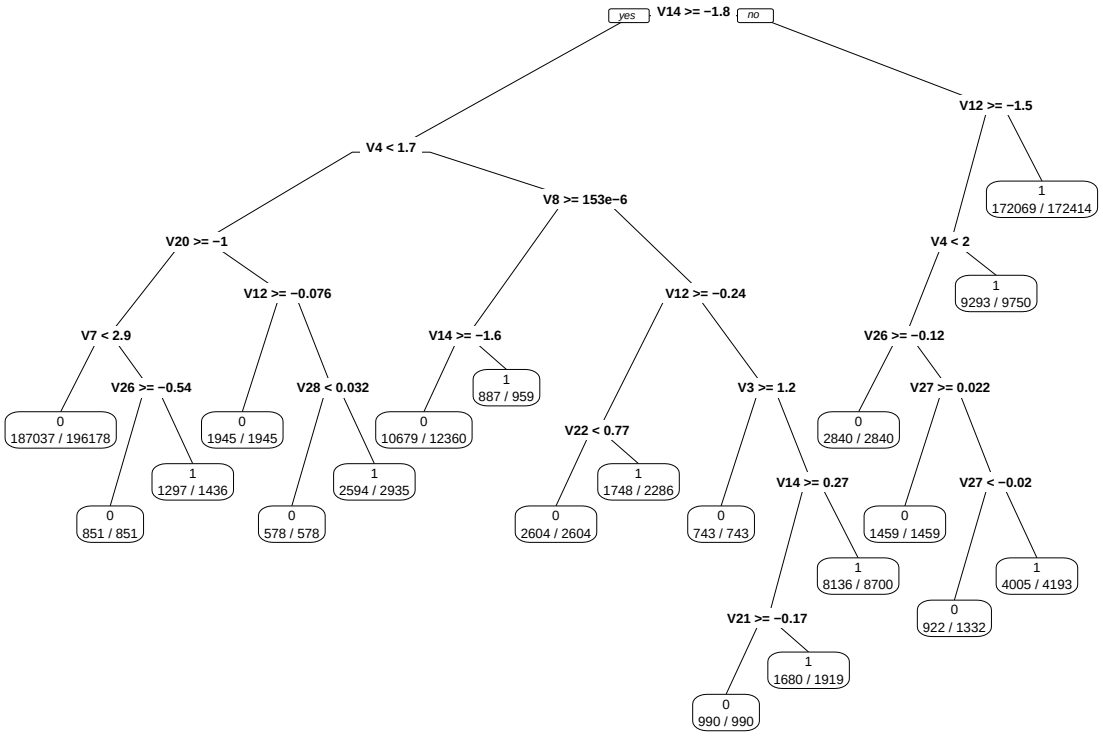
```r
#Création d'un dataset d'apprentissage et d'un dataset de validation
nb_lignes <- floor((nrow(over1)*0.75)) #Nombre de lignes de l'échantillon d'apprentissage : 75% du data
creditcard <- over1[sample(nrow(over1)), ] #Ajout de numéros de lignes
creditcard.train <- creditcard[1:nb_lignes,-1 ] #Echantillon d'apprentissage
creditcard.test <- creditcard[(nb_lignes+1):nrow(creditcard), -1] #Echantillon de test
```

4. classification:

```r
library(rpart)# Pour l'arbre de décision
library(rpart.plot) #pour tracer l'arbre de décision
creditcard.Tree <- rpart(Class~.,data=creditcard.train,method ="class",control=rpart.control(minsplit=5

plotcp(creditcard.Tree)
```




```
cp<-creditcard.Tree$cptable[which.min(creditcard.Tree$cptable[,4]),1]
creditcard.Tree_Opt <- prune(creditcard.Tree,cp=0.002)

#Représentation graphique de l'arbre optimal
prp(creditcard.Tree_Opt,extra=2)
```




```
creditcard.test_Predict<-predict(creditcard.Tree_Opt,newdata=creditcard.test, type="class")
```

```
#Matrice de confusion
True.label<-creditcard.test$Class
Predicted.label<-creditcard.test_Predict
mc<-table(True.label,Predicted.label)
print(mc)
```

```
##           Predicted.label
## True.label     0     1
##          0 69854   930
##          1  3832 67542
```

```
#Erreur de classement
erreur.classement<-1.0-(mc[1,1]+mc[2,2])/sum(mc)
print(paste(round(erreur.classement*100,2),"%"))
```

```
## [1] "3.35 %"
```

```
#Taux de prédictio
prediction=mc[2,2]/sum(mc[2,])
print(paste(round(prediction*100,2),"%"))
```

```
## [1] "94.63 %"
```