

Credit Card Fraud

Práctica de Aprendizaje Automático

Álvaro Barrio Hernández (alvaro.barrio.hernandez@gmail.com)

Código: [GitHub](#)



En este proyecto aprenderemos a realizar la detección de transacciones fraudulentas con tarjetas de crédito. Aplicaremos varios algoritmos como árboles de decisión, regresión logística, redes neuronales o el clasificador de Gradient Boosting. Para llevar a cabo la detección de fraudes con tarjetas de crédito, utilizaremos el conjunto de datos de tarjetas que contiene una combinación de transacciones fraudulentas y transacciones no fraudulentas.

El objetivo de este proyecto es construir un clasificador que pueda detectar transacciones fraudulentas con tarjetas de crédito. Usaremos una variedad de algoritmos de aprendizaje automático que podrán distinguir los fraudulentos de los no fraudulentos. Al final de este proyecto de aprendizaje automático, aprenderemos a implementar algoritmos de aprendizaje automático para realizar la clasificación.

Comencemos por instalar los paquetes necesarios para la correcta realización del ejercicio:

```
library(ranger)
library(caret)
library(data.table)
library(caTools)
library(pROC)
library(rpart)
library(rpart.plot)
library(neuralnet)
library(gbm, quietly=TRUE)
```

1. Importación del conjunto de datos

Importamos los conjuntos de datos que contienen transacciones realizadas con tarjetas de crédito.

```
creditcard_data <- read.csv("creditcard.csv")
#View(creditcard_data) #para abrir el conjunto en otra ventana
```

2. Exploración de datos

En esta sección del proyecto ML de detección de fraude, exploraremos los datos contenidos en el marco de datos creditcard_data. Continuaremos mostrando creditcard_data usando la función head () así como la función tail (). Luego procederemos a explorar los otros componentes de este marco de datos:

```
dim(creditcard_data)

## [1] 284807    31
```

```
head(creditcard_data,6)
```

```
##      Time      V1      V2      V3      V4      V5
V6
## 1      0 -1.3598071 -0.07278117 2.5363467 1.3781552 -0.33832077
0.46238778
## 2      0 1.1918571 0.26615071 0.1664801 0.4481541 0.06001765 -
0.08236081
## 3      1 -1.3583541 -1.34016307 1.7732093 0.3797796 -0.50319813
1.80049938
## 4      1 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888
1.24720317
## 5      2 -1.1582331 0.87773675 1.5487178 0.4030339 -0.40719338
0.09592146
## 6      2 -0.4259659 0.96052304 1.1411093 -0.1682521 0.42098688 -
0.02972755
##              V7      V8      V9      V10      V11
V12
## 1 0.23959855 0.09869790 0.3637870 0.09079417 -0.5515995 -
0.61780086
## 2 -0.07880298 0.08510165 -0.2554251 -0.16697441 1.6127267
1.06523531
## 3 0.79146096 0.24767579 -1.5146543 0.20764287 0.6245015
0.06608369
## 4 0.23760894 0.37743587 -1.3870241 -0.05495192 -0.2264873
0.17822823
## 5 0.59294075 -0.27053268 0.8177393 0.75307443 -0.8228429
0.53819555
## 6 0.47620095 0.26031433 -0.5686714 -0.37140720 1.3412620
0.35989384
##              V13      V14      V15      V16      V17      V18
## 1 -0.9913898 -0.3111694 1.4681770 -0.4704005 0.20797124 0.02579058
## 2 0.4890950 -0.1437723 0.6355581 0.4639170 -0.11480466 -0.18336127
## 3 0.7172927 -0.1659459 2.3458649 -2.8900832 1.10996938 -0.12135931
## 4 0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.68409279 1.96577500
## 5 1.3458516 -1.1196698 0.1751211 -0.4514492 -0.23703324 -0.03819479
## 6 -0.3580907 -0.1371337 0.5176168 0.4017259 -0.05813282 0.06865315
##              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
```

```
##          V25          V26          V27          V28 Amount Class
## 1  0.1285394 -0.1891148  0.133558377 -0.02105305 149.62      0
## 2  0.1671704  0.1258945 -0.008983099  0.01472417   2.69      0
## 3 -0.3276418 -0.1390966 -0.055352794 -0.05975184 378.66      0
## 4  0.6473760 -0.2219288  0.062722849  0.06145763 123.50      0
## 5 -0.2060096  0.5022922  0.219422230  0.21515315  69.99      0
## 6 -0.2327938  0.1059148  0.253844225  0.08108026   3.67      0
```

```
tail(creditcard_data,6)
```

```
##          Time          V1          V2          V3          V4
V5
## 284802 172785   0.1203164  0.93100513 -0.5460121 -0.7450968
1.13031398
## 284803 172786 -11.8811179 10.07178497 -9.8347835 -2.0666557 -
5.36447278
## 284804 172787  -0.7327887 -0.05508049  2.0350297 -0.7385886
0.86822940
## 284805 172788   1.9195650 -0.30125385 -3.2496398 -0.5578281
2.63051512
## 284806 172788  -0.2404400  0.53048251  0.7025102  0.6897992 -
0.37796113
## 284807 172792  -0.5334125 -0.18973334  0.7033374 -0.5062712 -
0.01254568
##          V6          V7          V8          V9          V10
V11
## 284802 -0.2359732  0.8127221  0.1150929 -0.2040635 -0.6574221
0.6448373
## 284803 -2.6068373 -4.9182154  7.3053340  1.9144283  4.3561704 -
1.5931053
## 284804  1.0584153  0.0243297  0.2948687  0.5848000 -0.9759261 -
0.1501888
## 284805  3.0312601 -0.2968265  0.7084172  0.4324540 -0.4847818
0.4116137
## 284806  0.6237077 -0.6861800  0.6791455  0.3920867 -0.3991257 -
1.9338488
## 284807 -0.6496167  1.5770063 -0.4146504  0.4861795 -0.9154266 -
1.0404583
##          V12          V13          V14          V15          V16
V17
## 284802  0.19091623 -0.5463289 -0.73170658 -0.80803553  0.5996281
0.07044075
## 284803  2.71194079 -0.6892556  4.62694203 -0.92445871  1.1076406
1.99169111
## 284804  0.91580191  1.2147558 -0.67514296  1.16493091 -0.7117573 -
0.02569286
## 284805  0.06311886 -0.1836987 -0.51060184  1.32928351  0.1407160
0.31350179
## 284806 -0.96288614 -1.0420817  0.44962444  1.96256312 -0.6085771
```

```

0.50992846
## 284807 -0.03151305 -0.1880929 -0.08431647 0.04133346 -0.3026201 -
0.66037665
##          V18          V19          V20          V21          V22
V23
## 284802 0.3731103 0.1289038 0.0006758329 -0.3142046 -0.8085204
0.05034266
## 284803 0.5106323 -0.6829197 1.4758291347 0.2134541 0.1118637
1.01447990
## 284804 -1.2211789 -1.5455561 0.0596158999 0.2142053 0.9243836
0.01246304
## 284805 0.3956525 -0.5772518 0.0013959703 0.2320450 0.5782290 -
0.03750086
## 284806 1.1139806 2.8978488 0.1274335158 0.2652449 0.8000487 -
0.16329794
## 284807 0.1674299 -0.2561169 0.3829481049 0.2610573 0.6430784
0.37677701
##          V24          V25          V26          V27          V28
Amount Class
## 284802 0.102799590 -0.4358701 0.1240789 0.217939865 0.06880333
2.69 0
## 284803 -0.509348453 1.4368069 0.2500343 0.943651172 0.82373096
0.77 0
## 284804 -1.016225669 -0.6066240 -0.3952551 0.068472470 -0.05352739
24.79 0
## 284805 0.640133881 0.2657455 -0.0873706 0.004454772 -0.02656083
67.88 0
## 284806 0.123205244 -0.5691589 0.5466685 0.108820735 0.10453282
10.00 0
## 284807 0.008797379 -0.4736487 -0.8182671 -0.002415309 0.01364891
217.00 0

table(creditcard_data$Class) # show in table how many 0-1

##
##      0      1
## 284315  492

summary(creditcard_data$Amount) # info for Amount column

##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      0.00   5.60   22.00   88.35   77.17 25691.16

names(creditcard_data) # column names

## [1] "Time"  "V1"    "V2"    "V3"    "V4"    "V5"    "V6"
"V7"
## [9] "V8"    "V9"    "V10"   "V11"   "V12"   "V13"   "V14"
"V15"
## [17] "V16"   "V17"   "V18"   "V19"   "V20"   "V21"   "V22"

```

```
"V23"
## [25] "V24"      "V25"      "V26"      "V27"      "V28"      "Amount" "Class"

var(creditcard_data$Amount)

## [1] 62560.07

sd(creditcard_data$Amount)

## [1] 250.1201
```

3. Manipulación de datos

En esta sección del proyecto de ciencia de datos de R, escalaremos nuestros datos usando la función `scale()`. Aplicaremos esto a la variable 'amount' para poder disponer de muestras más equilibradas. La escala también se conoce como estandarización de características. Con la ayuda del escalado, los datos se estructuran de acuerdo con un rango específico. Por lo tanto, no hay valores extremos en nuestro conjunto de datos que puedan interferir con el funcionamiento de nuestro modelo. Lo realizaremos de la siguiente manera:

```
head(creditcard_data)

##   Time          V1          V2          V3          V4          V5
## V6
## 1      0 -1.3598071 -0.07278117 2.5363467 1.3781552 -0.33832077
##    0.46238778
## 2      0 1.1918571 0.26615071 0.1664801 0.4481541 0.06001765 -
##    0.08236081
## 3      1 -1.3583541 -1.34016307 1.7732093 0.3797796 -0.50319813
##    1.80049938
## 4      1 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888
##    1.24720317
## 5      2 -1.1582331 0.87773675 1.5487178 0.4030339 -0.40719338
##    0.09592146
## 6      2 -0.4259659 0.96052304 1.1411093 -0.1682521 0.42098688 -
##    0.02972755
##           V7          V8          V9          V10          V11
## V12
## 1 0.23959855 0.09869790 0.3637870 0.09079417 -0.5515995 -
##    0.61780086
## 2 -0.07880298 0.08510165 -0.2554251 -0.16697441 1.6127267
##    1.06523531
## 3 0.79146096 0.24767579 -1.5146543 0.20764287 0.6245015
##    0.06608369
## 4 0.23760894 0.37743587 -1.3870241 -0.05495192 -0.2264873
```

```

0.17822823
## 5  0.59294075 -0.27053268  0.8177393  0.75307443 -0.8228429
0.53819555
## 6  0.47620095  0.26031433 -0.5686714 -0.37140720  1.3412620
0.35989384
##          V13          V14          V15          V16          V17          V18
## 1 -0.9913898 -0.3111694  1.4681770 -0.4704005  0.20797124  0.02579058
## 2  0.4890950 -0.1437723  0.6355581  0.4639170 -0.11480466 -0.18336127
## 3  0.7172927 -0.1659459  2.3458649 -2.8900832  1.10996938 -0.12135931
## 4  0.5077569 -0.2879237 -0.6314181 -1.0596472 -0.68409279  1.96577500
## 5  1.3458516 -1.1196698  0.1751211 -0.4514492 -0.23703324 -0.03819479
## 6 -0.3580907 -0.1371337  0.5176168  0.4017259 -0.05813282  0.06865315
##          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
##          V25          V26          V27          V28 Amount Class
## 1  0.1285394 -0.1891148  0.133558377 -0.02105305 149.62      0
## 2  0.1671704  0.1258945 -0.008983099  0.01472417  2.69      0
## 3 -0.3276418 -0.1390966 -0.055352794 -0.05975184 378.66      0
## 4  0.6473760 -0.2219288  0.062722849  0.06145763 123.50      0
## 5 -0.2060096  0.5022922  0.219422230  0.21515315  69.99      0
## 6 -0.2327938  0.1059148  0.253844225  0.08108026  3.67      0

creditcard_data$Amount=scale(creditcard_data$Amount)

NewData=creditcard_data[, -c(1)]

head(NewData)
##          V1          V2          V3          V4          V5          V6
## 1 -1.3598071 -0.07278117 2.5363467  1.3781552 -0.33832077  0.46238778
## 2  1.1918571  0.26615071 0.1664801  0.4481541  0.06001765 -0.08236081
## 3 -1.3583541 -1.34016307 1.7732093  0.3797796 -0.50319813  1.80049938
## 4 -0.9662717 -0.18522601 1.7929933 -0.8632913 -0.01030888  1.24720317
## 5 -1.1582331  0.87773675 1.5487178  0.4030339 -0.40719338  0.09592146
## 6 -0.4259659  0.96052304 1.1411093 -0.1682521  0.42098688 -0.02972755

```

##	V7	V8	V9	V10	V11	V12
## 1	0.23959855	0.09869790	0.3637870	0.09079417	-0.5515995	-0.61780086
## 2	-0.07880298	0.08510165	-0.2554251	-0.16697441	1.6127267	1.06523531
## 3	0.79146096	0.24767579	-1.5146543	0.20764287	0.6245015	0.06608369
## 4	0.23760894	0.37743587	-1.3870241	-0.05495192	-0.2264873	0.17822823
## 5	0.59294075	-0.27053268	0.8177393	0.75307443	-0.8228429	0.53819555
## 6	0.47620095	0.26031433	-0.5686714	-0.37140720	1.3412620	0.35989384
##	V13	V14	V15	V16	V17	V18
## 1	-0.9913898	-0.3111694	1.4681770	-0.4704005	0.20797124	0.02579058
## 2	0.4890950	-0.1437723	0.6355581	0.4639170	-0.11480466	-0.18336127
## 3	0.7172927	-0.1659459	2.3458649	-2.8900832	1.10996938	-0.12135931
## 4	0.5077569	-0.2879237	-0.6314181	-1.0596472	-0.68409279	1.96577500
## 5	1.3458516	-1.1196698	0.1751211	-0.4514492	-0.23703324	-0.03819479
## 6	-0.3580907	-0.1371337	0.5176168	0.4017259	-0.05813282	0.06865315
##	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
##	V25	V26	V27	V28	Amount	Class
## 1	0.1285394	-0.1891148	0.133558377	-0.02105305	0.24496383	0
## 2	0.1671704	0.1258945	-0.008983099	0.01472417	-0.34247394	0
## 3	-0.3276418	-0.1390966	-0.055352794	-0.05975184	1.16068389	0
## 4	0.6473760	-0.2219288	0.062722849	0.06145763	0.14053401	0
## 5	-0.2060096	0.5022922	0.219422230	0.21515315	-0.07340321	0
## 6	-0.2327938	0.1059148	0.253844225	0.08108026	-0.33855582	0

4. Modelado de datos

Una vez que hayamos estandarizado todo nuestro conjunto de datos, dividiremos nuestro conjunto de datos en un conjunto de entrenamiento y un conjunto de prueba con una proporción de división de 0,80. Esto significa que el 80% de nuestros datos se

atribuirán a `train_data`, mientras que el 20% se atribuirá a los datos de prueba. Luego comprobaremos las dimensiones usando la función `dim()`.

```
set.seed(123)
data_sample = sample.split(NewData$Class, SplitRatio=0.80)
train_data = subset(NewData, data_sample==TRUE)
test_data = subset(NewData, data_sample==FALSE)
dim(train_data)

## [1] 227846      30

dim(test_data)

## [1] 56961      30
```

5. Ajuste del modelo de regresión logística

En esta sección del proyecto de detección de fraudes con tarjetas de crédito, ajustaremos nuestro primer modelo. Comenzaremos con la regresión logística. Se utiliza una regresión logística para modelar la probabilidad de resultado de una clase como positivo / negativo y, en nuestro caso, fraude / no fraude. Procedemos a implementar este modelo en nuestros datos de prueba de la siguiente manera:

```
Logistic_Model = glm(Class~., test_data, family=binomial())

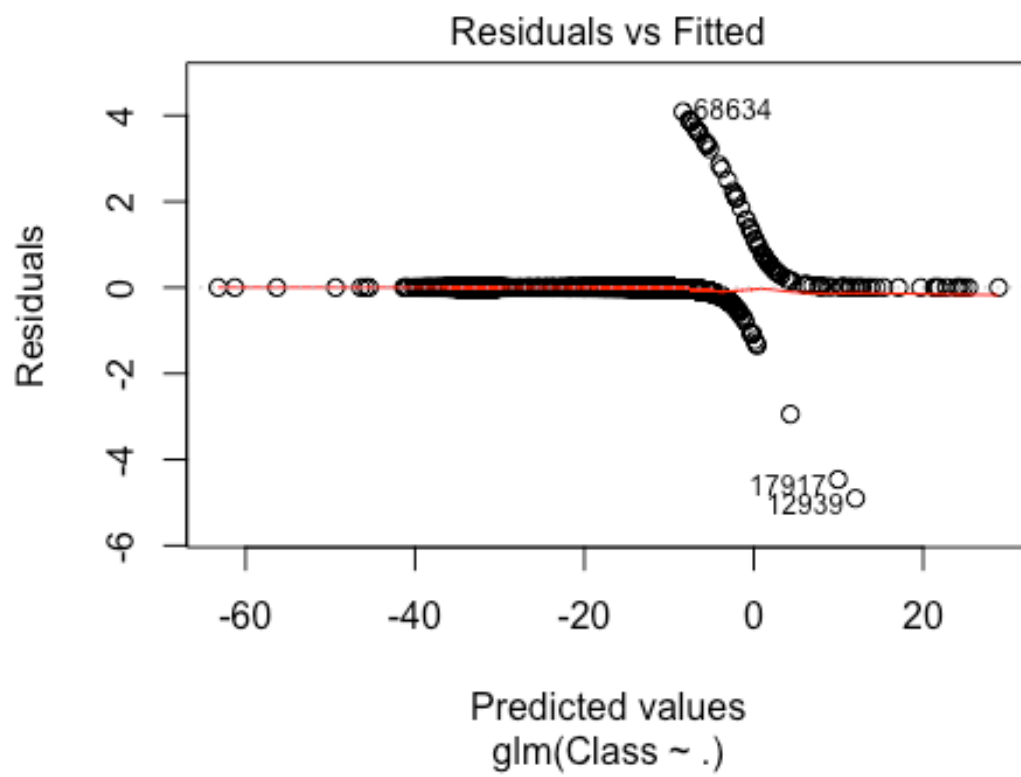
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

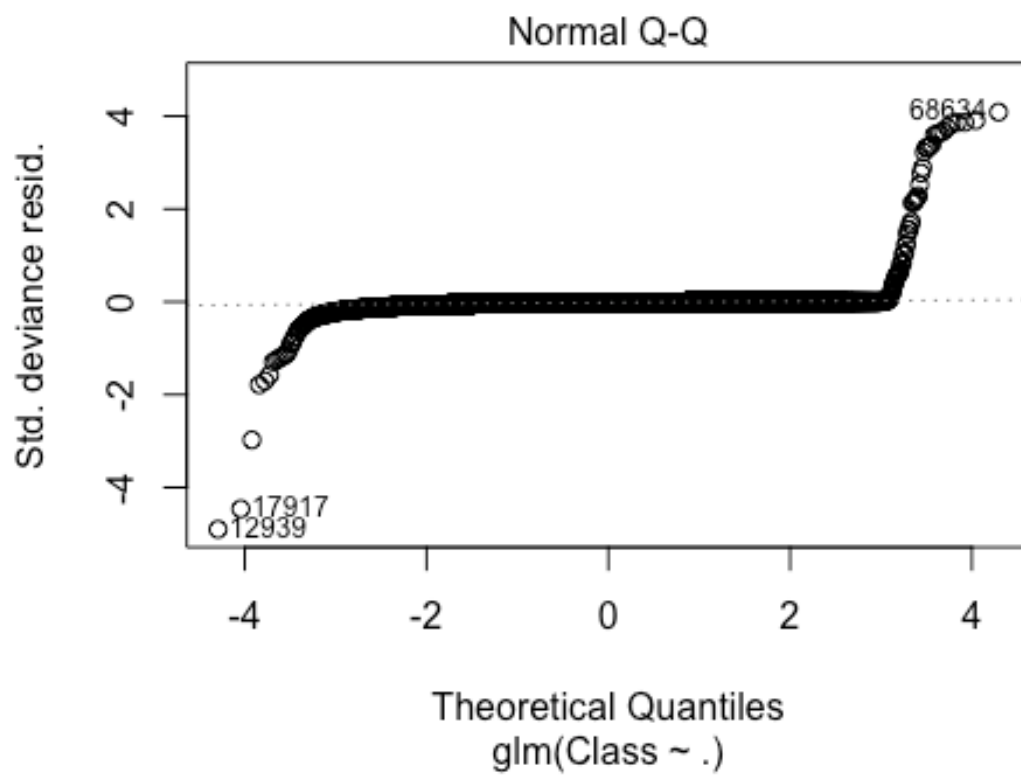
summary(Logistic_Model)

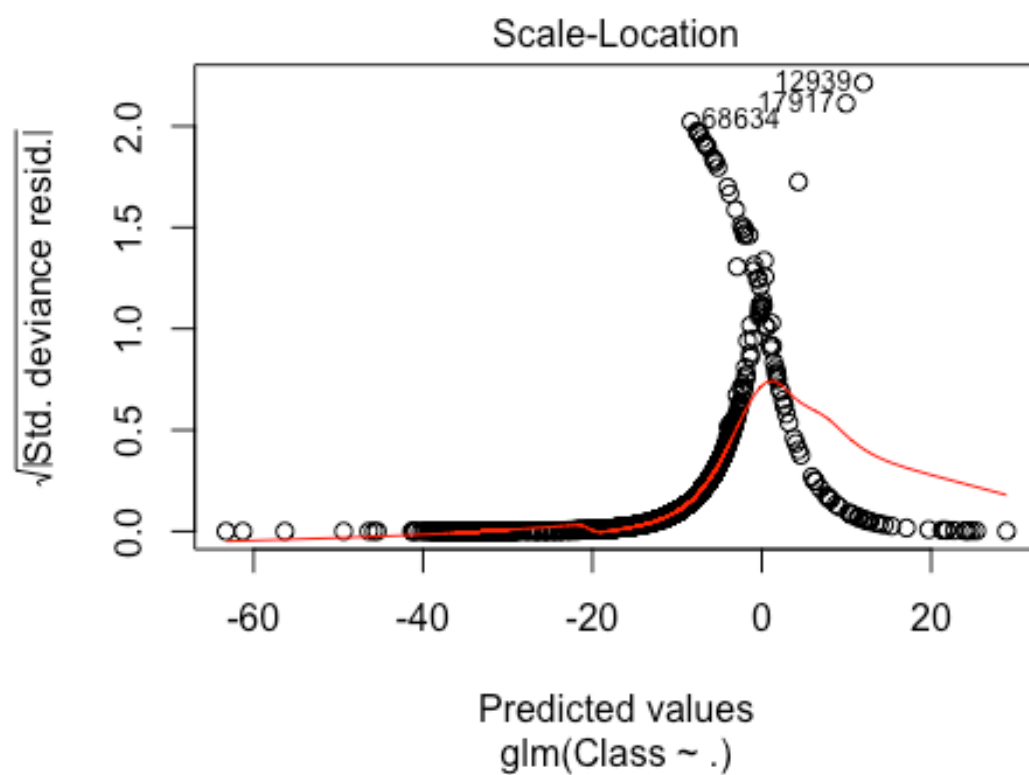
##
## Call:
## glm(formula = Class ~ ., family = binomial(), data = test_data)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -4.9019  -0.0254  -0.0156  -0.0078   4.0877
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -12.52800   10.30537  -1.216   0.2241
## V1           -0.17299    1.27381  -0.136   0.8920
## V2            1.44512    4.23062   0.342   0.7327
## V3            0.17897    0.24058   0.744   0.4569
## V4            3.13593    7.17768   0.437   0.6622
## V5            1.49014    3.80369   0.392   0.6952
## V6           -0.12428    0.22202  -0.560   0.5756
## V7            1.40903    4.22644   0.333   0.7388
## V8           -0.35254    0.17462  -2.019   0.0435 *
## V9            3.02176    8.67262   0.348   0.7275
```

```
## V10      -2.89571    6.62383   -0.437    0.6620
## V11      -0.09769    0.28270   -0.346    0.7297
## V12       1.97992    6.56699    0.301    0.7630
## V13      -0.71674    1.25649   -0.570    0.5684
## V14       0.19316    3.28868    0.059    0.9532
## V15       1.03868    2.89256    0.359    0.7195
## V16      -2.98194    7.11391   -0.419    0.6751
## V17      -1.81809    4.99764   -0.364    0.7160
## V18       2.74772    8.13188    0.338    0.7354
## V19      -1.63246    4.77228   -0.342    0.7323
## V20      -0.69925    1.15114   -0.607    0.5436
## V21      -0.45082    1.99182   -0.226    0.8209
## V22      -1.40395    5.18980   -0.271    0.7868
## V23       0.19026    0.61195    0.311    0.7559
## V24      -0.12889    0.44701   -0.288    0.7731
## V25      -0.57835    1.94988   -0.297    0.7668
## V26       2.65938    9.34957    0.284    0.7761
## V27      -0.45396    0.81502   -0.557    0.5775
## V28      -0.06639    0.35730   -0.186    0.8526
## Amount    0.22576    0.71892    0.314    0.7535
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 1443.40  on 56960  degrees of freedom
## Residual deviance:  378.59  on 56931  degrees of freedom
## AIC: 438.59
##
## Number of Fisher Scoring iterations: 17

plot(Logistic_Model)
```

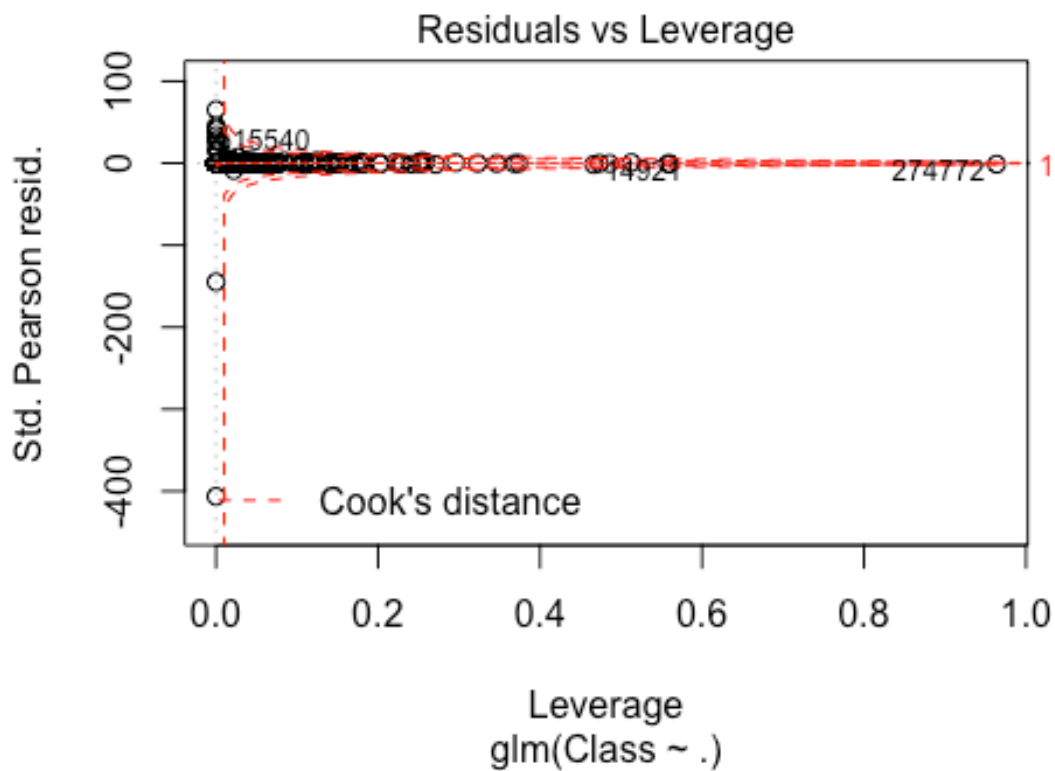






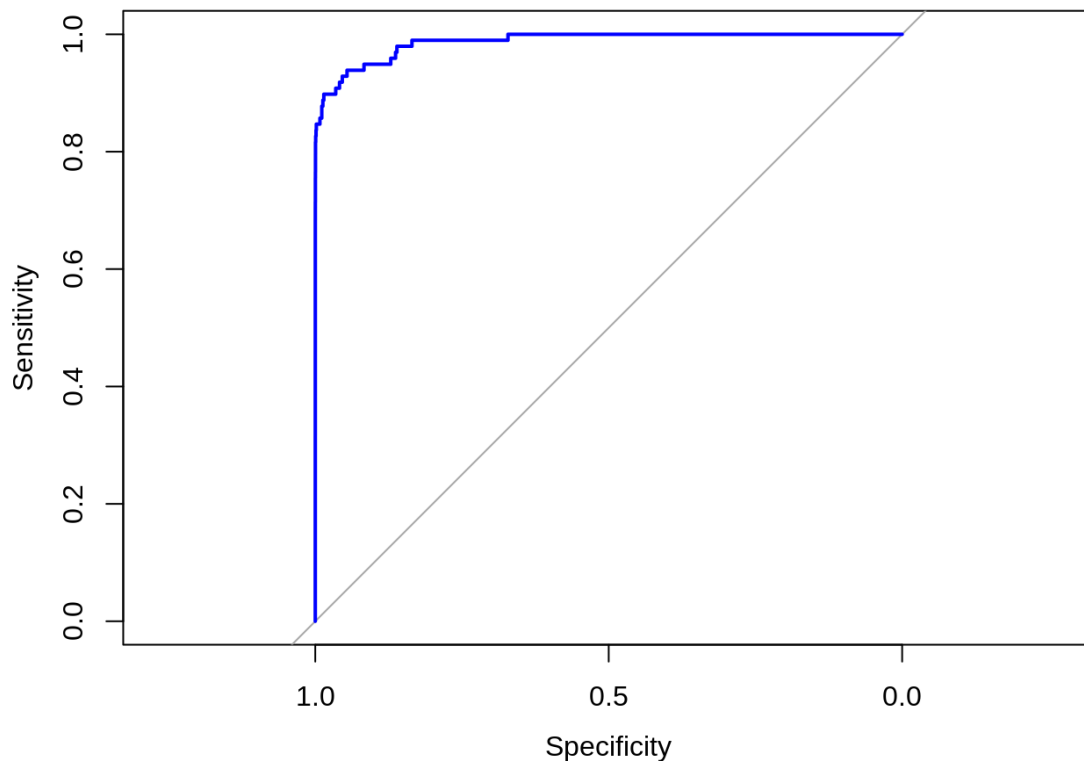
```
## Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced
```

```
## Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced
```



Para evaluar el desempeño de nuestro modelo, delinearemos la curva ROC. ROC también se conoce como características optimistas del receptor. Para esto, primero importaremos el paquete ROC y luego trazaremos nuestra curva ROC para analizar su desempeño.

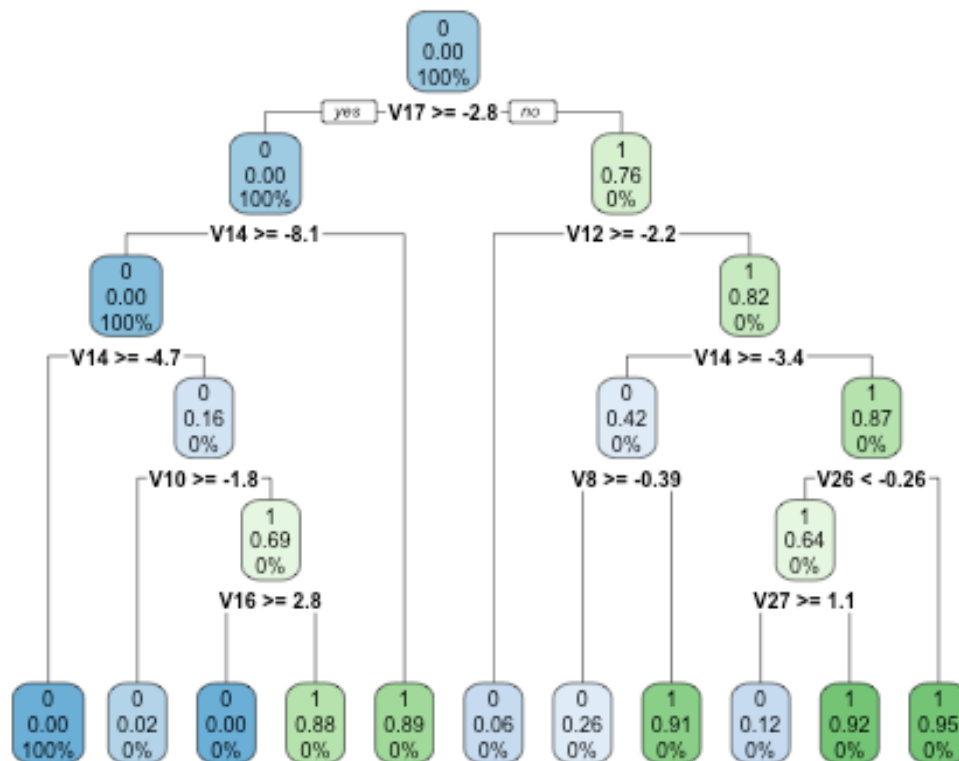
```
lr.predict <- predict(Logistic_Model,train_data, probability = TRUE)
auc.gbm = roc(test_data$Class, lr.predict, plot = TRUE, col = "blue")
```



6. Ajuste de un modelo de árbol de decisiones

En esta sección, implementaremos un algoritmo de árbol de decisión para trazar los resultados. Estos resultados son básicamente una consecuencia a través de la cual podemos concluir a qué clase pertenece el objeto. Ahora implementaremos nuestro modelo de árbol de decisión y lo trazaremos usando la función `rpart.plot()`. Usaremos específicamente la partición recursiva para trazar el árbol de decisión.

```
decisionTree_model <- rpart(Class ~ . , creditcard_data, method =  
'class')  
predicted_val <- predict(decisionTree_model, creditcard_data, type =  
'class')  
probability <- predict(decisionTree_model, creditcard_data, type =  
'prob')  
rpart.plot(decisionTree_model)
```



7. Red neuronal artificial

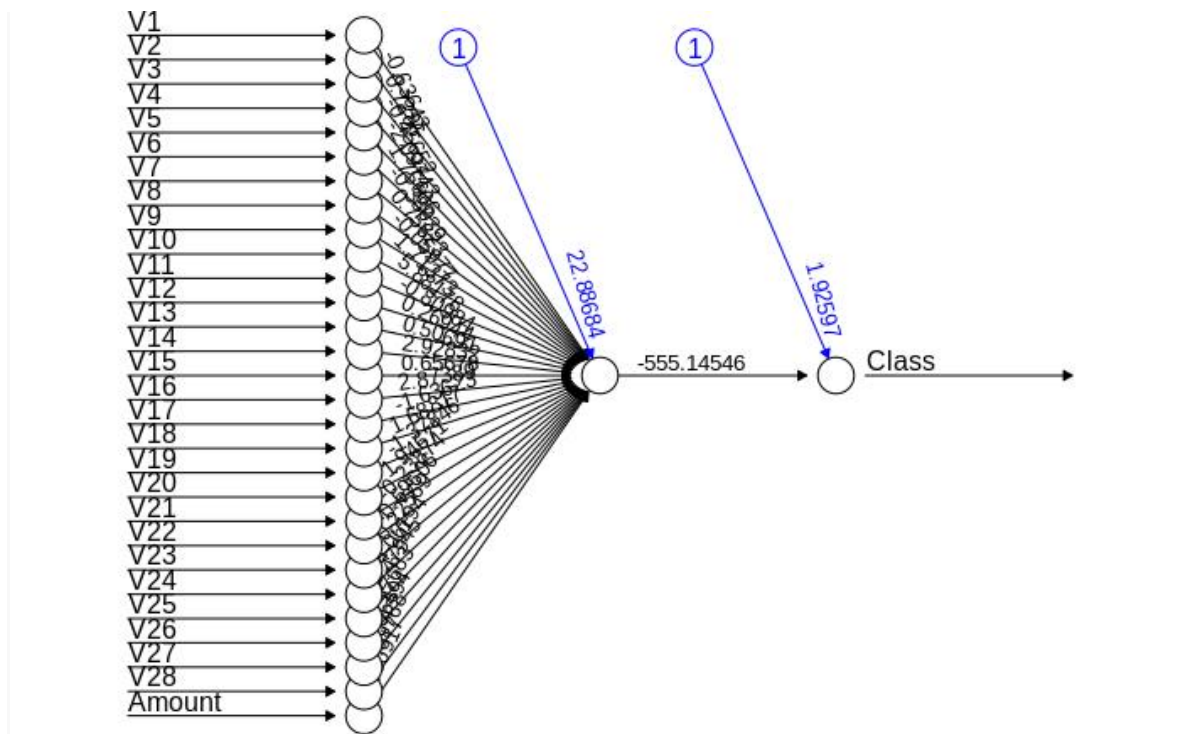
Las redes neuronales artificiales son un tipo de algoritmo de aprendizaje automático que se modela a partir del sistema nervioso humano. Los modelos ANN pueden aprender los patrones utilizando los datos históricos y pueden realizar una clasificación en los datos de entrada. Importamos el paquete `neuralnet` que nos permitiría implementar nuestras ANN. Luego procedimos a trazarlo usando la función `plot()`. Ahora bien, en el caso de las Redes Neuronales Artificiales, existe un rango de valores que está entre 1 y 0. Establecemos un umbral en 0.5, es decir, valores por encima de 0.5 corresponderán a 1 y el resto será 0. Implementamos esto como sigue:

```

library(neuralnet)
ANN_model =neuralnet (Class~.,train_data,linear.output=FALSE)
plot(ANN_model)

predANN=compute(ANN_model,test_data)
resultANN=predANN$net.result
resultANN=ifelse(resultANN>0.5,1,0)

```

8. Aumento de gradiente (GBM)

Gradient Boosting es un popular algoritmo de aprendizaje automático que se utiliza para realizar tareas de clasificación y regresión. Este modelo se compone de varios modelos de conjuntos subyacentes como árboles de decisión débiles. Estos árboles de decisión se combinan para formar un modelo sólido de aumento de gradiente. Implementaremos el algoritmo de descenso de gradiente en nuestro modelo de la siguiente manera:

```
library(gbm, quietly=TRUE)

## Loaded gbm 2.1.8

# Get the time to train the GBM model
system.time(
  model_gbm <- gbm(Class ~ .
    , distribution = "bernoulli"
    , data = rbind(train_data, test_data)
    , n.trees = 500
    , interaction.depth = 3
    , n.minobsinnode = 100
    , shrinkage = 0.01
    , bag.fraction = 0.5
    , train.fraction = nrow(train_data) / (nrow(train_data) +
nrow(test_data))
```

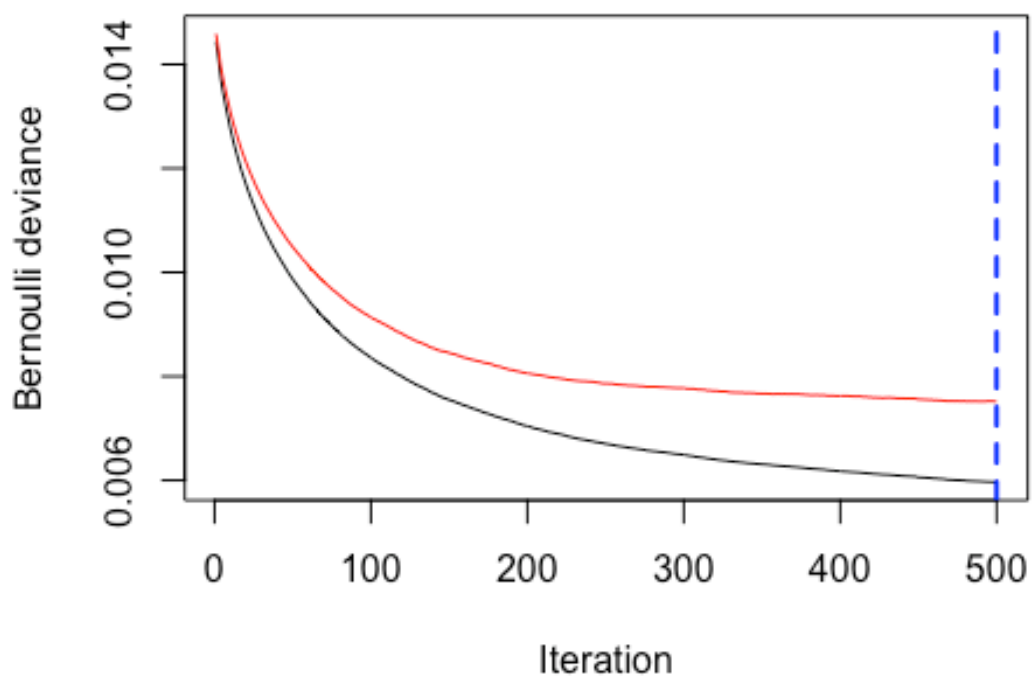
```

)
)

##      user  system elapsed
## 305.771   0.684 306.611

# Determine best iteration based on test data
gbm.iter = gbm.perf(model_gbm, method = "test")

```

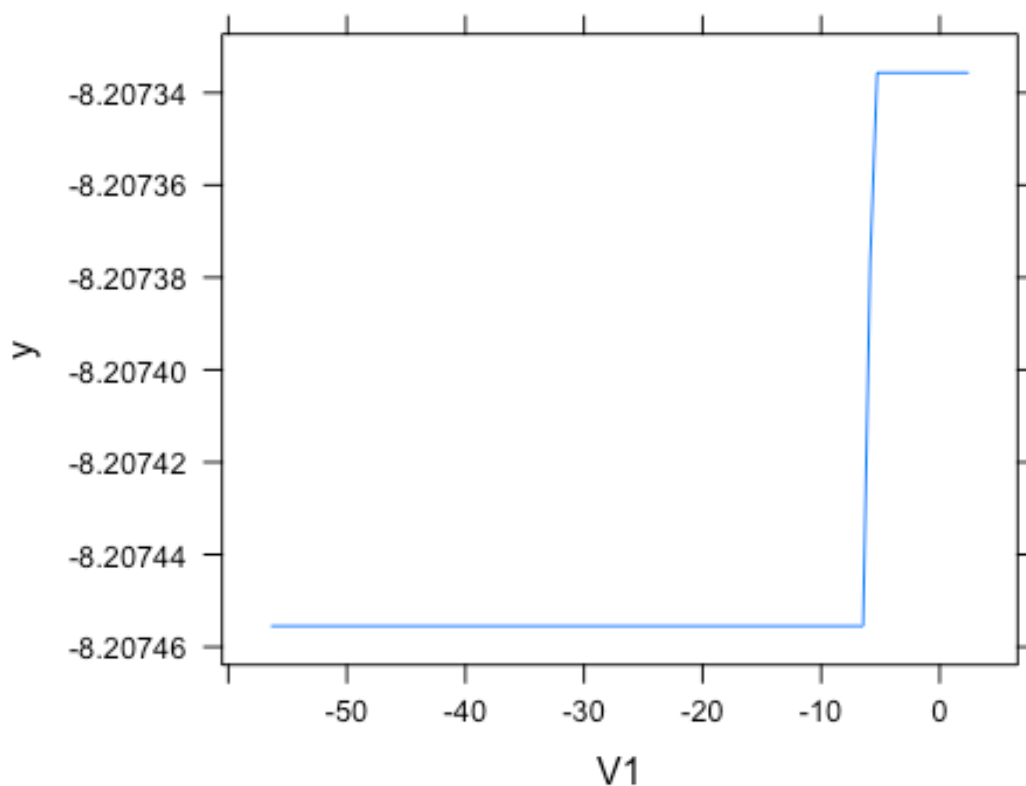


```

model.influence = relative.influence(model_gbm, n.trees = gbm.iter, sort.
= TRUE)
#Plot the gbm model

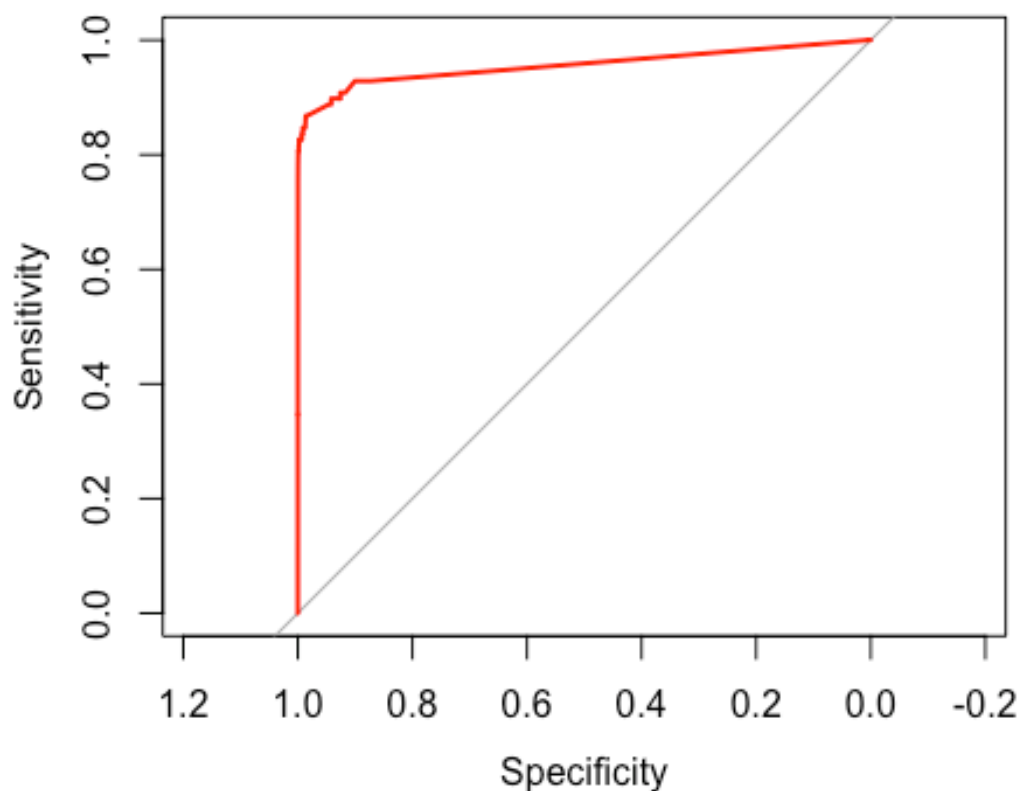
plot(model_gbm)

```



```
# Plot and calculate AUC on test data
gbm_test = predict(model_gbm, newdata = test_data, n.trees = gbm.iter)
gbm_auc = roc(test_data$Class, gbm_test, plot = TRUE, col = "red")

## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```



```
print(gbm_auc)

##
## Call:
## roc.default(response = test_data$Class, predictor = gbm_test,      plot
##             = TRUE, col = "red")
##
## Data: gbm_test in 56863 controls (test_data$Class 0) < 98 cases
##       (test_data$Class 1).
## Area under the curve: 0.9555
```

Concluimos el proyecto habiendo aprendido a cómo desarrollar un modelo de detección de fraude con tarjetas de crédito mediante el aprendizaje automático. Usamos una variedad de algoritmos ML para implementar este modelo y también trazamos las respectivas curvas de desempeño para los modelos. Aprendimos cómo se pueden analizar y visualizar los datos para distinguir transacciones fraudulentas de otros tipos de datos.

Pueden encontrar todo el código disponible en mi [GitHub](#)

¡Saludos!