Práctica Minería de Datos y Modelización predictiva

Alejandro Sierra Fernández

En el siguiente documento se procederá a resolver el ejercicio de evaluación del primer módulo de minería de datos del máster de Big Data y Data Science de la UCM. El problema consiste en predecir, con la máxima eficacia posible, la fuga de clientes de una compañía telefónica. Para ello disponemos de una base de datos con distintas variables, continuas y discretas, que describen a estos clientes. Entre estas variables cabe destacas la *fuga*, que es una variable binaria que toma el valor *1* si el cliente ha dejado de pagar o *0* en el caso contrario. Esta es nuestra variable objetivo, la que trataremos de predecir.

Por el camino tendremos que ir realizando distintas tareas como cargar e inspeccionar los datos, depurarlos, plantear y comparar modelos o evaluar e interpretar los parámetros más importantes.

Finalmente cabe decir que debido a que la extensión del PDF se encuentra limitada, no podremos explicar cada detalle del código ni cada salida, sino que nos centraremos en las más relevantes.

1- Lectura, inspección y depuración de Datos:

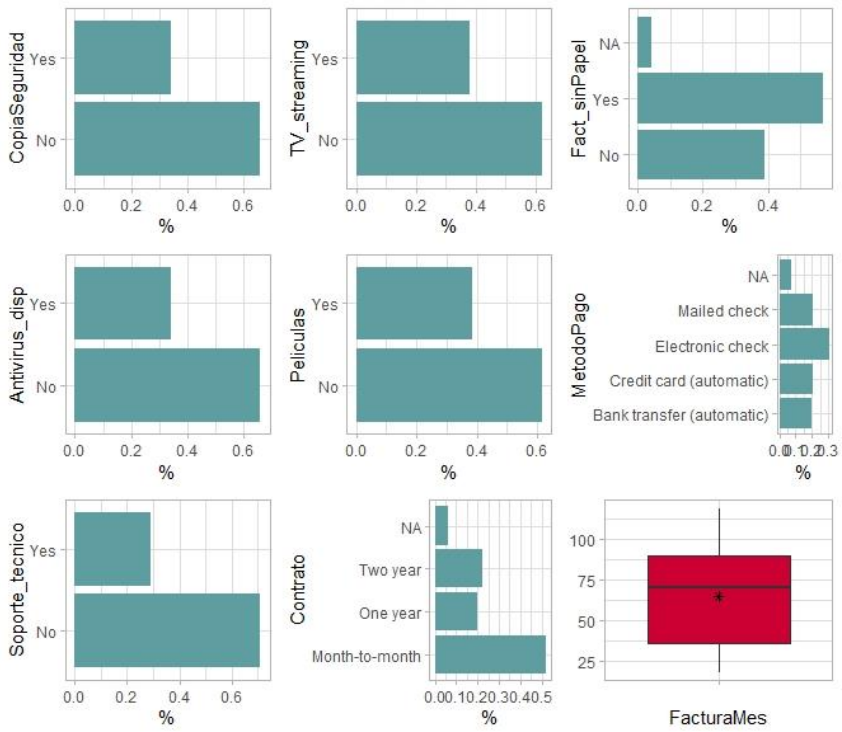
Comenzamos por fijar el directorio en el que vamos a trabajar y a cargar funciones y librerías que emplearemos más adelante. El directorio es un directorio absoluto, es decir, que para hacer funcionar el código en otro ordenador deberíamos modificarlo al correspondiente. También cargamos e inspeccionamos los datos brevemente con las funciones *names()*, *str()* y *summary().*

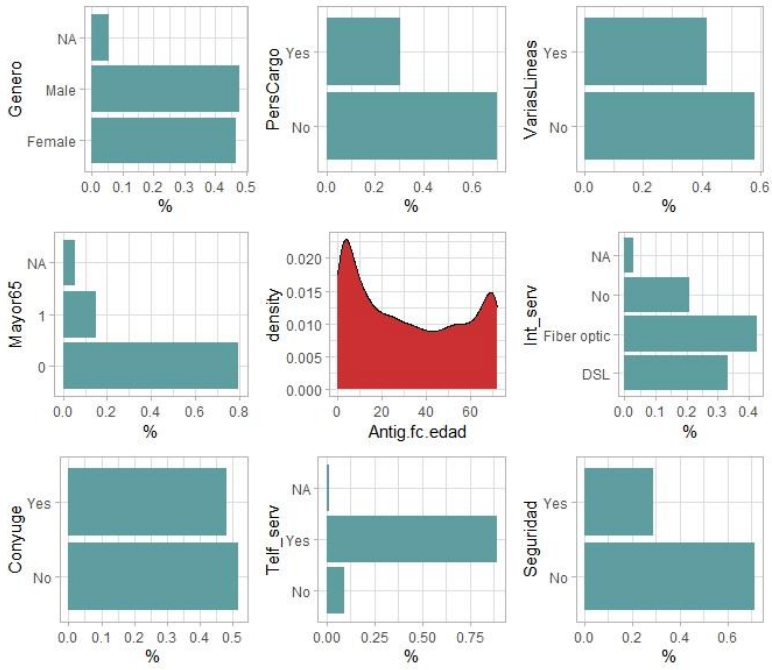
```
rm(list=ls())

getwd()

setwd('C:/Users/aleja/OneDrive/Escritorio/R_Data_Mining')

source("Funciones_R.R")

paquetes(c('questionr','psych','car','corrplot','caret',

    'ggplot2', 'lmSupport','pROC','gridExtra'))

datos_test <-
readRDS("C:/Users/aleja/OneDrive/Escritorio/R_Data_Mining/FugaClientes_test.RDS")

datos_training <-
readRDS("C:/Users/aleja/OneDrive/Escritorio/R_Data_Mining/FugaClientes_Training.R
DS")

str(datos_training)

names(datos_training)

summary(datos_training)
```

Vemos que nos encontramos ante un dataset de 6352 observaciones con 20 variables, de las que cuatro son numéricas y el resto categóricas. Fijándonos en información como los cuartiles, la media, los valores máximos y mínimos y el número de instancias de cada categoría, analizamos superficialmente los datos para darnos cuenta de que no hay, en principio, nada especialmente extraño.

Pasamos al análisis gráfico a través de histogramas y boxplot's, donde obtenemos imágenes como las siguientes:

```
listaGraf <- dfplot_box(datos_training[,-1])

listaHist<- dfplot_his(datos_training[,-1])

gridExtra::marrangeGrob(listaGraf, nrow = 3, ncol = 3)

gridExtra::marrangeGrob(listaHist, nrow = 3, ncol = 3)
```



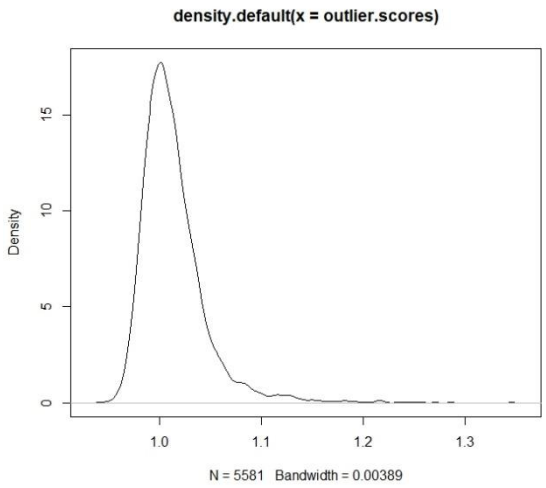


Confirmamos gráficamente que no hay valores que no hay nada que nos llame la atención y pasamos al tratamiento de *missings* y *outliers*. Previamente preparamos la información, separando la variable objetivo y convirtiendo la variable ID de categórica a numérica para no tener problemas posteriormente con temas de correlación. Después aplicamos la función *AtipicosAmissing* para detectar los outliers, una función que utiliza dos criterios simultáneamente, que cada valor debe cumplir para ser considerado como extremo (Media + 3 sd (Distribuciones Simétricas)/Mediana + 8 mad(Distribuciones Asimétricas) y los umbrales Cuartil1 – 3RIC/Cuartil3 +3RIC).

```
varObj<-datos_training$Fuga

datos_training<-datos_training[,-21]

datos_training$ID<-as.numeric(datos_training$ID)

str(datos_training)


sapply(Filter(is.numeric, datos_training),

function(x) atipicosAmissing(x)[[2]])/nrow(datos_training)
```

No detectamos ningún outlier en las variables continuas, lo que ya intuíamos después de haber analizado los boxplots. De todas formas, probamos ahora con un método multivariante. Estos métodos no analizan exclusivamente la variable, sino que estudian todos los valores de la instancia para determinar si es más o menos extrema, ordenándolas en función a la distancia a la que se encuentren del origen de un plano n-dimensional.

```
outlier.scores <- lofactor(na.omit(Filter(is.numeric, datos_training)), k=20)

plot(density(outlier.scores))
```



Como vemos en la imagen, hay algunos valores que se alejan un poco del pico de densidad. Esto sin embargo no es algo excepcional, ni la distancia es lo suficientemente grande como para que nos planteemos eliminar estas instancias de nuestro dataset. De todas formas, vamos a acceder a las cinco instancias que el algoritmo considera más extremas para analizar si realmente hay algo fuera de lo normal:

```
outliers <- order(outlier.scores, decreasing=T)[1:5]
(na.omit(Filter(is.numeric, datos_training))[outliers,] ->out5)
out5$ID -> ID_out5
```

	ID	Antig.fc.edad	FacturaMes	FacturaTotal
2135	4272	72	117.15	8529.50
4460	6345	1	18.90	18.90
6109	6195	72	117.50	8670.10
6295	6343	53	94.00	4871.45
5626	6336	1	20.05	20.05

Los comparamos con la media y con la mediana de las variables continuas:

```
data.frame(t(round(apply(na.omit(Filter(is.numeric, datos_training)),2,mean),3)))
data.frame(t(round(apply(na.omit(Filter(is.numeric, datos_training)),2,median),3)))
```

	ID	Antig.fc.edad	FacturaMes	FacturaTotal
1	3173.774	32.507	65.071	2293.917
1	3169	29	70.55	1405.3

Podríamos pensar que las facturas totales de las instancias son muy altas o muy bajas comparadas con la media, pero accediendo a los valores máximos y mínimos de estas comprobamos que no estamos ante ningún caso extremo, habiendo valores que están por encima y por debajo.

```
datos_training %>% slice_min(FacturaTotal)

datos_training %>% slice_max(FacturaTotal)
```

Pasamos a continuación al estudio de los *missings (NA'S),* prestando atención a si existe una correlación entre ellos, a la proporción que hay en cada instancia y en cada variable. Además, añadimos una nueva variable con la proporción de missings que presenta cada instancia.

```
corrplot(cor(is.na(datos_training[colnames(

datos_training)[colSums(is.na(datos_training))>0]])),method = "ellipse",type =
"upper")

prop_missingsVars<-apply(is.na(datos_training),2,mean)

t<-data.frame(sort(prop_missingsVars*100, decreasing = T))

names(t)<-"% Missing por Variable"

datos_training$prop_missings<-apply(is.na(datos_training),1,mean)
```

Comprobamos que no existen correlaciones. Además, la variable que más missings tiene presenta un 7% de ellos, que nos parece un porcentaje más que aceptable. Podemos ahora inputar valores a estos missings gracias a la función *ImputacionCuant*. Lo haremos por aleatoriedad, aunque esta aleatoriedad está sometida a una distribución de probabilidad determinada por la variable continua. Las variables categóricas también la inputamos por aleatoriedad. Finalmente resaltar que pasamos dos veces la función para eliminar todos los NA's. Finalmente guardamos nuestros datos depurados en un archivo .RDS

```
datos_training_al <- copy(datos_training)


datos_training_al[,as.vector(which(sapply(datos_training, class)=="numeric"))]<-
sapply(

Filter(is.numeric, datos_training),function(x) ImputacionCuant(x,"aleatorio"))


datos_training_al[,as.vector(which(sapply(datos_training, class)=="factor"))]<-
sapply(

Filter(is.factor, datos_training),function(x) ImputacionCuali(x,"aleatorio"))


datos_training_al[,as.vector(which(sapply(datos_training, class)=="character"))]
<- lapply(

datos_training[,as.vector(which(sapply(datos_training, class)=="character"))] ,
factor)


indx <- apply(datos_training_al, 2, function(x) any(is.na(x) | is.infinite(x)))

colnames(datos_training_avg)[indx]
```

```
datos_training_al[,as.vector(which(sapply(datos_training, class)=="numeric"))]<-
sapply(

Filter(is.numeric, datos_training_al),function(x) ImputacionCuant(x,"aleatorio"))


apply(datos_training_al, 2, function(x) any(is.na(x)))

saveRDS(datos_training_al,"datos_training_al.RDS")
```

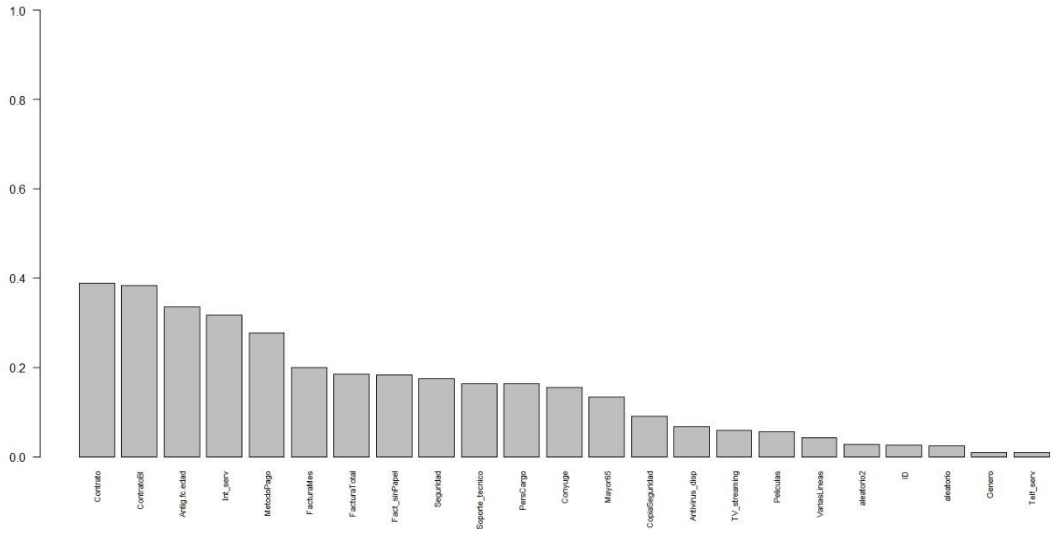
2- *Estudio de variables con la variable objetivo:*

Ahora incluimos dos variables aleatorias de control y una variable "contrato" binaria. La original, con tres categorías, se encuentra un poco desbalanceada y quizás esta variable nos sea de ayuda más adelante:

```
datos_training_al$ContratoBI <- as.character(datos_training_al$Contrato)

datos_training_al$ContratoBI[datos_training_al$ContratoBI == "One year"] <-
"+Month"

datos_training_al$ContratoBI[datos_training_al$ContratoBI == "Two year"] <-
"+Month"

datos_training_al$ContratoBI <- as.factor(datos_training_al$ContratoBI)


datos_training_al$aleatorio<-runif(nrow(datos_training_al))


datos_training_al$aleatorio2<-runif(nrow(datos_training_al))
```

Pasamos a buscar la relación de nuestras variables con la variable objetivo a través del test de Cramer:

```
graficoVcramer(datos_training_al,varObj)
```

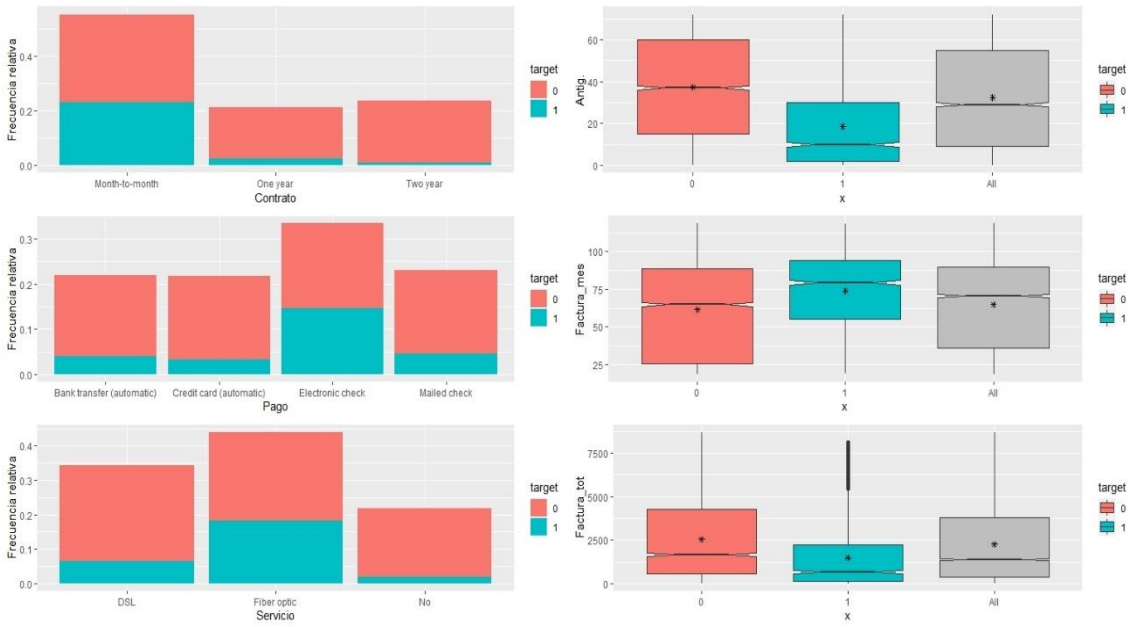


Vemos variables que se destacan como el contrato, la antigüedad o el tipo de servicio. Tendremos en cuenta esta información cuando modelicemos. Cabe destacar que la variable que habíamos introducido con la proporción de missings no aparece por ser numérica y tener menos de seis valores diferentes. La categorizamos y añadimos una nueva variable binaria:

```
unique(datos_training_al$prop_missings)

questionr::freq(datos_training_al$prop_missings)

datos_training_al$prop_missings_cat <- as.character(datos_training_al$prop_missings)



        datos_training_al$prop_missings_cat[datos_training_al$prop_missings_cat != "0"]
        <- "Con Missings"

        datos_training_al$prop_missings_cat[datos_training_al$prop_missings_cat == "0"]
        <- "Sin Missings"

        datos_training_al$prop_missings_cat <-
        as.factor(datos_training_al$prop_missings_cat)



datos_training_al$prop_missings <- as.factor(datos_training_al$prop_missings)
```

Mostramos también algunos gráficos a través de los que se pueden detectar relaciones y comportamientos de cada variable con respecto a la original. Lo hacemos a través de las funciones *boxplot_targerbinaria* y *barras_targetbinaria*.

```
bx1<-boxplot_targetbinaria(datos_training_al$Antig.fc.edad,varObj,"Antig.")

bx2<-boxplot_targetbinaria(datos_training_al$FacturaMes,varObj,"Factura_mes")

bx3<-boxplot_targetbinaria(datos_training_al$FacturaTotal,varObj,"Factura_tot")


h1<-barras_targetbinaria(datos_training_al$Contrato,varObj,"Contrato")

h2<-barras_targetbinaria(datos_training_al$MetodoPago,varObj,"Pago")

h3<-barras_targetbinaria(datos_training_al$Int_serv,varObj,"Servicio")


marrangeGrob(list(h1,h2,h3,bx1,bx2,bx3),nrow = 3, ncol = 2)
```



Vemos como aparece una cierta asimetría con respecto a la frecuencia relativa de las distintas categorías en los casos de compra y venta. Son estas diferencias entre un caso y otro las que nos indican si nos encontramos ante una variable a partir de la que el modelo podrá obtener información de utilidad a la hora de predecir. Esto vemos que también ocurre para las variables continuas, a partir de las que se observa cómo las personas con más antigüedad y con una factura total mayor tienen menos tendencia a fugarse, mientras que con la factura mensual ocurre lo contrario. La cuestión es hasta qué punto estas diferencias que se aprecian en el gráfico serán relevantes dentro de nuestro modelo.

A continuación, buscamos las mejores transformaciones de las variables numéricas con respecto a la variable binaria y almacenamos todo en un nuevo dataframe de nombre *data_tavg*:

```
data_tavg<-cbind(datos_training_al,Transf_Auto(Filter(is.numeric,
datos_training_al),varObj))

graficoVcramer(data_tavg,varObj)
```

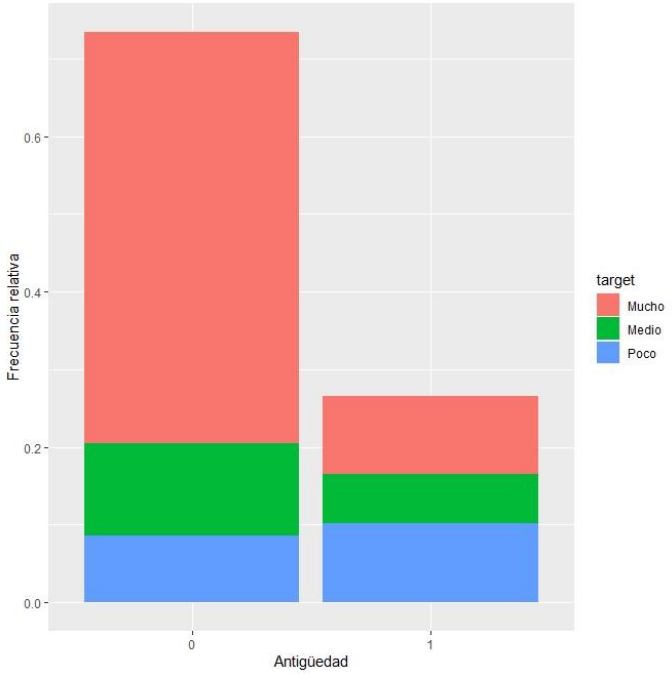
Después de hacer Cramer otra vez se observa como transformadas de las variables continuas tienen más capacidad predictiva que las originales. Es el caso de la raíz de la antigüedad o la raíz cuarta de las facturas total y mensual.

Por último, vamos a tramificar la variable raíz de la antigüedad para convertirla en categórica. Elegimos esta en lugar de la original porque consideramos que puede ser más relevante que la original por lo visto en la prueba V de Cramer. Esto lo hacemos buscando puntos de corte óptimos con respecto a la variable objetivo. Cabe decir que lo hemos intentado aplicar también a las facturas totales y mensuales pero el resultado no es bueno, apareciendo en muchos casos categorías excesivamente subrepresentadas.

```
tree_Antiguedad <-rpart::rpart(varObj~sqrtxAntig.fc.edad, data = data_tavg, cp=0.01)

tree_Antiguedad

table(tree_Antiguedad$where)

data_tavg$tree_Antiguedad<-factor(tree_Antiguedad$where)

levels(data_tavg$tree_Antiguedad) = c('Mucho','Medio','Poco')

table(data_tavg$tree_Antiguedad)
```

```
Mucho Medio  Poco
 3998  1162  1193
```

h4<-barras_targetbinaria(varObj,data_tavg$tree_Antiguedad,"Antigüedad")



No estamos descontentos con el resultado, pues aparece un desequilibrio en la categoría "mucho", reflejándose más claramente la relación que ya intuíamos con la variable continua. Puede ser una variable de utilidad para el modelo.

3- Modelado Manual:

Empezamos renombrando a nuestro dataset y realizamos la partición de este entre el set de entrenamiento y el de test. El 80% del mismo se corresponderá con el de entrenamiento. Empleamos la función *createDataPartition* de la librería *caret*

```
data <- data.frame(data_tavg,varObj)

set.seed(123456)

trainIndexx <- createDataPartition(data$varObj, p=0.8, list=FALSE)

data_train <- data[trainIndexx,]

data_test <- data[-trainIndexx,]
```

A continuación, hacemos dos primeros modelos de regresión logística que tendremos como referencia, un primero solamente con las variables originales y un segundo que incluirá todas las variables:

```
modelo_0<-glm(varObj~.,data=data_train[,-c(1,21:31)],family=binomial)

summary(modelo_0)

modelo_1<-glm(varObj~., data=data_train,family=binomial)

summary(modelo_1)
```

Mostramos el primer modelo:

```
> summary(modelo_0)

Call:
glm(formula = varObj ~ ., family = binomial, data = data_train[,
    -c(1, 21:31)])

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.0619  -0.6611  -0.3115   0.6651   3.1533

Coefficients:
                                  Estimate Std. Error z value Pr(>|z|)
(Intercept)                     -0.2027214  0.2050258  -0.989 0.322780
GeneroMale                      -0.0410402  0.0764344  -0.537 0.591313
Mayor65l                         0.2614111  0.1016603   2.571 0.010128 *
ConyugeYes                      -0.0901239  0.0919437  -0.980 0.326984
PersCargoYes                    -0.1541666  0.1050274  -1.468 0.142140
Antig.fc.edad                   -0.0209447  0.0039569  -5.293 1.20e-07 ***
Telf_servYes                    -0.6004943  0.1613253  -3.722 0.000197 ***
VariasLineasYes                  0.3554766  0.0949228   3.745 0.000180 ***
Int_servFiber optic              1.0006864  0.1305823   7.663 1.81e-14 ***
Int_servNo                      -0.6331534  0.1665317  -3.802 0.000144 ***
SeguridadYes                    -0.4803443  0.1031040  -4.659 3.18e-06 ***
CopiaSeguridadYes               -0.1747730  0.0943552  -1.852 0.063985 .
Antivirus_dispYes               -0.0323822  0.0962720  -0.336 0.736598
Soporte_tecnicoYes              -0.3367222  0.1032671  -3.261 0.001111 **
TV_streamingYes                  0.3664752  0.1023648   3.580 0.000343 ***
PeliculasYes                     0.2197763  0.1010307   2.175 0.029605 *
ContratoOne year                -0.7176958  0.1193627  -6.013 1.82e-09 ***
ContratoTwo year                -1.3154806  0.1741296  -7.555 4.20e-14 ***
Fact_sinPapelYes                 0.3661217  0.0861039   4.252 2.12e-05 ***
MetodoPagoCredit card (automatic) -0.1793578  0.1311833  -1.367 0.171553
MetodoPagoElectronic check       0.1880148  0.1091810   1.722 0.085061 .
MetodoPagoMailed check          -0.0198381  0.1275565  -0.156 0.876408
FacturaMes                       0.0021193  0.0034371   0.617 0.537503
FacturaTotal                    -0.0001586  0.0000510  -3.110 0.001872 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 5882.3  on 5082  degrees of freedom
Residual deviance: 4246.1  on 5059  degrees of freedom
AIC: 4294.1

Number of Fisher Scoring iterations: 6
```

La calidad de los modelos se mide a través del criterio de información de Akaike (AIC), que representan el ajuste y el número de términos del modelo. Cuanto menos sea este número mayor calidad tendrá el modelo. También es importante analizar el p-valor de cada una de las variables, que nos indica si una variable es representativa o no a la hora de predecir la variable objetivo. Esto viene también representado gráficamente a través de las estrellas, que aumentan junto con la calidad de la variable. Un modelo muy estrellado será un modelo con muchas variables representativas y por tanto con capacidad de predicción.

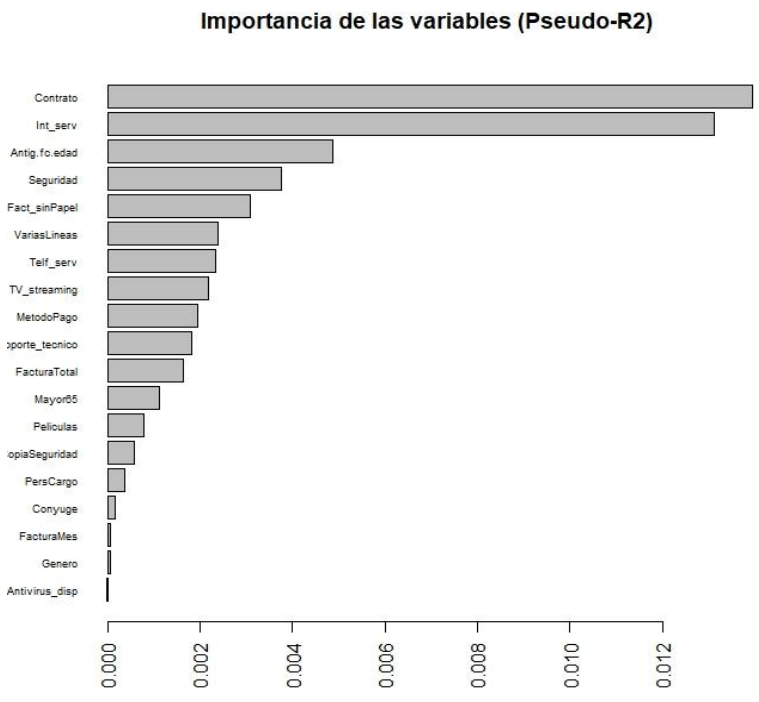
También podemos calcular la pseudo-R-cuadrado, que sería una adaptación de la R-cuadrado para modelos logarítmicos. Esta compara tu modelo con el modelo nulo, sin coeficientes, y te indica en qué medida tus variables están aportando información útil a la hora de predecir. Hay que tener en cuenta que un pseudo-R-cuadrado de entorno a 0.4 equivale a R's-cuadrados de 0.8. Por último, con la variable *rank* del modelo podemos acceder al número de parámetros que tiene, para hacernos una idea de la complejidad de unos y otros.

```
pseudoR2(modelo_0,data_train,"varObj")
pseudoR2(modelo_0,data_test,"varObj")
modeloIniciall$rank
```

Encontramos entonces que el modelo_0 tiene una pseudoR2 de 0.27 para el train y 0.25 para los datos del test. Esto indica que está sobre ajustando un poco los datos del set de entrenamiento, pero tampoco en exceso. Además, tiene 24 parámetros. Por otra parte, el modelo_1 mejora un par de centésimas el pseudoR2 pero aumenta el número de parámetros a 38, lo que lo vuelve más complejo y más difícil de interpretar.

Con la función *impVariablesLog* para ordenar las variables por orden de importancia con respecto al pseudoR2 de McFaden. Nos da una idea visual muy clara sobre la importancia de las variables dentro del modelo. Lo hacemos para el modelo_0 con las variables originales del dataframe.

```
impVariablesLog(modelo_0,"varObj")
```



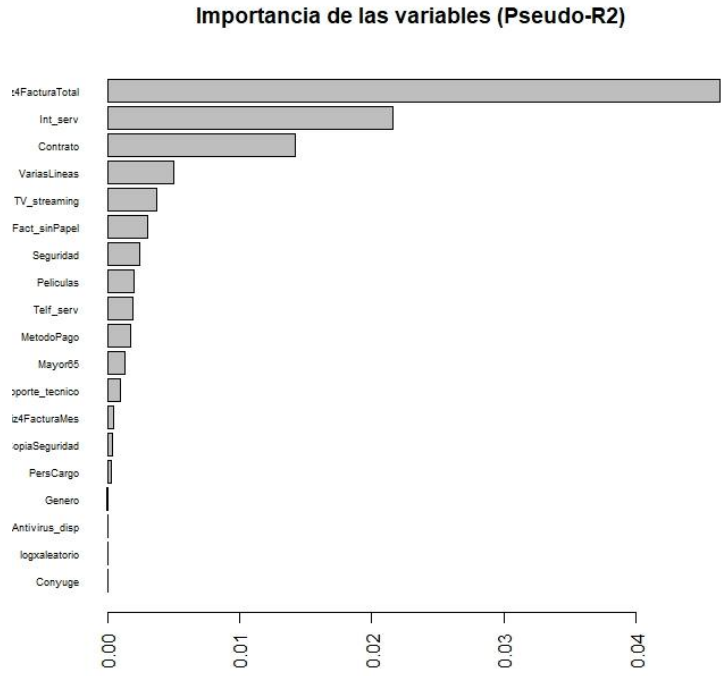
Vemos como variables muy destacadas en cuanto a importancia son el contrato y el servicio. Dentro de las continuas, parece que de momento es la antigüedad la que toma cierta importancia. Veremos si reemplazando estas variables por algunas de sus transformadas ganamos algo de rendimiento. Veremos también qué ocurre con la variable antigüedad truncada y con las variables que nos informan sobre los missings.

Probamos a continuación diferentes modelos, donde iremos jugando con la complejidad de los mismos y con su capacidad predictiva, reemplazando unas variables por otras y estudiando si merece la pena incluir alguna interacción entre variables. Por brevedad no nos vamos a detener en la explicación pormenorizada de cada modelo. De esta forma generamos:

```
modelo_2<-glm(varObj~.,data=data_train[,-c(1,6,19,20,21:26,30:31)],family=binomial)

modelo_3<-glm(varObj~.,data=data_train[,-c(1,6,19,20,21:28,30:31)],family=binomial)

modelo_4<-glm(varObj~.,data=data_train[,-c(1:6,11,12,18:28,30:31)],family=binomial)

modelo_5<-glm(varObj~Contrato+TV_streaming+Mayor65+Soporte_tecnico+Peliculas

+Int_serv+VariasLineas+Seguridad+Fact_sinPapel+Telf_serv+raiz4FacturaTotal+TV_streaming

                +MetodoPago*Int_serv, data=data_train,family=binomial)
```

Cabe destacar que a partir del modelo_2 intercambiamos las variables continuas por sus transformadas más importantes y el cambio resulta ser muy bueno, convirtiéndose la raíz cuarta de la factura total en la variable con una mayor capacidad predictora.

```
impVariablesLog(modeloIniciall,"varObj")
```



4- *Modelado por selección de variables*

A continuación, generaremos modelos por selección de variables. Es decir, será el algoritmo el que automáticamente elegirá las variables que mejoren el modelo. Iremos aumentando progresivamente la complejidad de los modelos, de tal manera que empezaremos por no considerar interacciones entre las variables para hacerlo después. Emplearemos la función *step(),* que tiene distintas formas de proceder: O bien parte del modelo nulo para ir añadiendo variables en orden de aporte al R2, o bien hace el camino contrario, partiendo de un modelo con todas las variables y eliminando variables. También existe una tercera funcionalidad, en la que se realiza el forward, pero valorando el efecto de las salidas de variables que ya fueron incluidas en el modelo. Necesitamos entonces generar un modelo vacío y un modelo con todas las variables, para que el algoritmo vaya haciendo el recorrido del uno hacia el otro.

```
full <- glm(varObj~.,data=data_train[,-c(1,21:31)],family=binomial)

null <- glm(varObj~1,data=data_train[,-c(1,21:31)],family=binomial)

modeloStepAIC<-step(null, scope=list(lower=null, upper=full), direction="both", trace =
F)

summary(modeloStepAIC)

pseudoR2(modeloStepAIC,data_train,"varObj")

pseudoR2(modeloStepAIC,data_test,"varObj")


modeloBackAIC<-step(full, scope=list(lower=null, upper=full), direction="backward", trace
= F)

summary(modeloBackAIC)

pseudoR2(modeloBackAIC,data_train,"varObj")

pseudoR2(modeloBackAIC,data_test,"varObj")


modeloStepBIC<-step(null,              scope=list(lower=null,              upper=full),
direction="both",k=log(nrow(data_train)), trace = F)

summary(modeloStepBIC)

pseudoR2(modeloStepBIC,data_train,"varObj")

pseudoR2(modeloStepBIC,data_test,"varObj")


modeloBackBIC<-step(full,              scope=list(lower=null,              upper=full),
direction="backward",k=log(nrow(data_train)), trace = F)

summary(modeloBackBIC)

pseudoR2(modeloBackBIC,data_train,"varObj")

pseudoR2(modeloBackBIC,data_test,"varObj")
```

Consideraremos también la existencia de interacciones entre las variables. Para generar las interacciones empleo la función *formIntT.* Tenemos que volver a generar los dos modelos, el completo y el nulo:

```
formIntT<-formulaInteracciones(data_train,32)

fullIntT <- glm(formIntT,data=data_train,family=binomial)

null2 <- glm(varObj~1,data=data_train,family=binomial)


modeloStepAIC_transInt<-step(null2,       scope=list(lower=null2,       upper=fullIntT),
direction="both", trace = F)

summary(modeloStepAIC_transInt)

pseudoR2(modeloStepAIC_transInt,data_train,"varObj")

pseudoR2(modeloStepAIC_transInt,data_test,"varObj")


modeloStepBIC_transInt<-step(null2,       scope=list(lower=null2,       upper=fullIntT),
direction="both",k=log(nrow(data_train)), trace = F)

summary(modeloStepBIC_transInt)

pseudoR2(modeloStepBIC_transInt,data_train,"varObj")

pseudoR2(modeloStepBIC_transInt,data_test,"varObj")


modeloStepAIC_transInt$rank

modeloStepBIC_transInt$rank
```

Nos damos cuenta de que las interacciones no son de gran relevancia, pues ninguna ha sobrevivido en los dos modelos anteriores.

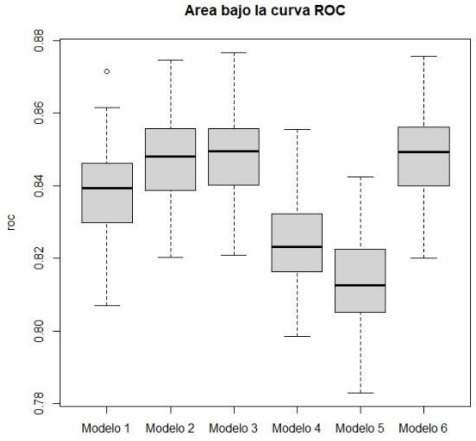
Consideramos que con estos seis modelos generados por selección automática tenemos suficiente. Además, hemos cubierto desde el caso más simple, teniendo en cuenta solo las variables originales, hasta el más complejo, donde consideramos todas las variables y sus interacciones.

5- Comparación por validación cruzada:

A continuación, comprobamos por validación cruzada los modelos. Primero estudiamos los que hemos programado manualmente y después haremos lo mismo con los modelos generados por selección automática:

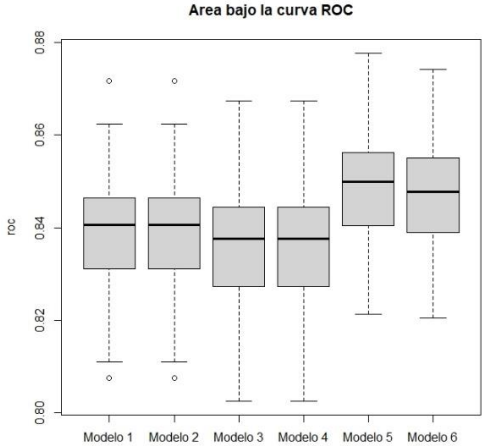
```
auxVarObj<- data$varObj

data$varObj<-make.names(data$varObj)´

totall<-c()

modeloss<-sapply(list(modelo_0,modelo_1,modelo_2,modelo_3,modelo_4,modelo_5),formula)

for (i in 1:length(modeloss)){

  set.seed(1712)
```

```
  vcr<-train(as.formula(modeloss[[i]]), data = data,
             method = "glm", family="binomial",metric = "ROC",
             trControl = trainControl(method="repeatedcv", number=5, repeats=20,
                                      summaryFunction=twoClassSummary,
                                      classProbs=TRUE,returnResamp="all")
  )
  totall<-rbind(totall,data.frame(roc=vcr$resample[,1],modeloo=rep(paste("Modelo",i),
                                                                   nrow(vcr$resample))))
}
boxplot(roc~modeloo,data=totall,main="Area bajo la curva ROC")
```



Como vemos, según la variable ROC, los más interesantes son el 2, 3 y el 6 (modelo_1, modelo_2 y modelo_5). En cuanto a simpleza, los más interesantes son el modelo_2 y el modelo_5, con 24 y 23 parámetros cada uno. La pseudoR2 de estos dos modelos solo difiere a partir de las milésimas y el modelo_2 no considera interacciones, mientras que el modelo_5 sí que lo hace (Int_serv*MetodoPago). Decidimos entones quedarnos con el modelo_2 como mejor candidato dentro del modelado manual.

Hacemos lo propio con los modelos de selección automática y obtenemos el siguiente plot:



En este caso destaca el modelo5 = modeloStepAIC_transInt, que fue generado teniendo en cuenta todas las variables y sus posibles interacciones. Este tiene 20 parámetros (bastante simple) y el pseudoR2 sólo difiere en el orden de las milésimas del pseudoR2 del modelo_2, por lo que decidimos quedarnos con el modeloStepAIX_transInt como nuestro ganador.

```
> pseudoR2(modeloStepAIC_transInt,data_train,"varObj")
[1] 0.2958017
> pseudoR2(modeloStepAIC_transInt,data_test,"varObj")
[1] 0.2780246
> modeloStepAIC_transInt$rank
[1] 20
```

6- *Interpretación de parámetros:*

Ajustamos el modelo final a los datos completos para tener estimadores más fiables:

```
modelo_final<-glm(formula(modeloStepAIC_transInt),
                  data=data,family=binomial)
summary(modelo_final)
```

```
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.1874  -0.6555  -0.3065   0.5823   3.1180

Coefficients:
                              Estimate Std. Error z value Pr(>|z|)
(Intercept)                    0.73495    0.31496   2.333 0.019624
*
ContratoOne year              -0.57380    0.10650  -5.388 7.13e-08
***
ContratoTwo year              -1.32294    0.15557  -8.504  < 2e-16
***
Int_servFiber optic            1.17510    0.10498  11.193  < 2e-16
***
Int_servNo                    -0.97734    0.19004  -5.143 2.71e-07
***
raiz4FacturaTotal             -3.37571    0.18668 -18.083  < 2e-16
***
TV_streamingYes                0.43353    0.08873   4.886 1.03e-06
***
VariasLineasYes                0.44322    0.08678   5.108 3.26e-07
***
Fact_sinPapelYes               0.37942    0.07836   4.842 1.28e-06
***
PeliculasYes                   0.37164    0.08766   4.240 2.24e-05
***
SeguridadYes                  -0.32724    0.09088  -3.601 0.000317
***
Mayor651                       0.21734    0.09002   2.414 0.015770
*
Telf_servYes                  -0.44198    0.14291  -3.093 0.001984
**
```

```
MetodoPagoCredit card (automatic) -0.21728    0.11801  -1.841 0.065600
.
MetodoPagoElectronic check         0.18474    0.09813   1.883 0.059764
.
MetodoPagoMailed check            -0.14581    0.11738  -1.242 0.214176
Soporte_tecnicoYes                -0.27428    0.09062  -3.027 0.002472
**
PersCargoYes                      -0.14503    0.08606  -1.685 0.091947
.
raiz4FacturaMes                    0.64194    0.31930   2.010 0.044384
*
CopiaSeguridadYes                 -0.13180    0.08192  -1.609 0.107624
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 7351.9  on 6352  degrees of freedom
Residual deviance: 5197.8  on 6333  degrees of freedom
AIC: 5237.8
```

Con el modelo presente, calculamos los OR para poder interpretar:

```
epiDisplay::logistic.display(modelo_fin, simplified = TRUE)
```

(simplified = TRUE por las colinealidades de algunas de las variables del modelo, como se indica en la documentación de la librería “epiDisplay”)

```
                                        OR  lower95ci  upper95ci
Pr(>|Z|)
ContratoOne year                0.56337887 0.45724511 0.69414794 7.1
28215e-08
ContratoTwo year                0.26634997 0.19635046 0.36130451 1.8
33322e-17
Int_servFiber optic             3.23846363 2.63618477 3.97834280 4.4
10210e-29
Int_servNo                      0.37630889 0.25928687 0.54614558 2.7
06924e-07
raiz4FacturaTotal               0.03419388 0.02371614 0.04930068 4.3
68660e-73
TV_streamingYes                 1.54269209 1.29644431 1.83571239 1.0
28809e-06
VariasLineasYes                 1.55771087 1.31408124 1.84650924 3.2
63633e-07
Fact_sinPapelYes                1.46144381 1.25337807 1.70404930 1.2
84857e-06
PeliculasYes                    1.45011139 1.22120558 1.72192387 2.2
37401e-05
SeguridadYes                    0.72090800 0.60328904 0.86145829 3.1
70405e-04
Mayor651                        1.24276083 1.04173860 1.48257392 1.5
77039e-02
Telf_servYes                    0.64276541 0.48573938 0.85055360 1.9
84229e-03
MetodoPagoCredit card (automatic) 0.80470652 0.63853600 1.01412072 6.5
60015e-02
MetodoPagoElectronic check      1.20290586 0.99242965 1.45802024 5.9
76441e-02
MetodoPagoMailed check          0.86432411 0.68669039 1.08790827 2.1
41764e-01
Soporte_tecnicoYes              0.76011895 0.63642631 0.90785189 2.4
71537e-03
PersCargoYes                    0.86499745 0.73073671 1.02392637 9.1
94683e-02
raiz4FacturaMes                 1.90015555 1.01624977 3.55285798 4.4
38392e-02
CopiaSeguridadYes               0.87651629 0.74650527 1.02916998 1.0
76244e-01
```

Interpretamos algunas de las variables del modelo:

-La probabilidad de fuga de aquellos clientes con contratos de un año y de dos años disminuye un 44% y un 74% respectivamente con respecto a los clientes con contratos mensuales.

-La probabilidad de fuga con respecto a no-fuga de aquellos clientes con fibra óptica como servicio es 3.23 veces superior a aquellos que tienen DSL. Por otra parte, para aquellos clientes que no tienen servicio de internet contratado la probabilidad de fuga se reduce en un 63% con respecto a aquellos que tienen DSL

-Cada aumento unitario de la raíz cuarta de la factura total produce una reducción del 97% en la probabilidad de fuga. (Por ejemplo si pasamos de 20 euros de factura total a 94)

-Los clientes que tienen personas a su cargo tienen un 14% menos de probabilidad de fugarse que aquellos que no las tienen.

-La probabilidad de fuga con respecto a no-fuga de aquellos clientes que tienen televisión es 1.54 veces superior a aquellos que no la tienen.

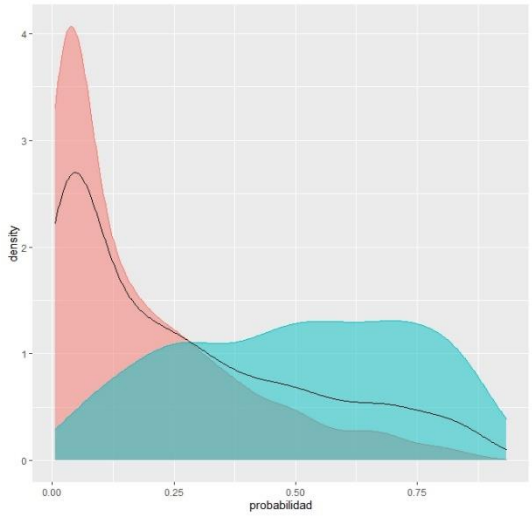
-La probabilidad de fuga con respecto a no-fuga de aquellos clientes que tienen varias líneas contratadas es 1.55 veces superior a aquellos que solo tienen una.

-La probabilidad de fuga con respecto a no-fuga de aquellos clientes que facturan sin papel es 1.46 veces superior a aquellos que sí lo hacen.

7- *Punto de corte y predicción:*

Buscamos ahora el punto de corte óptimo para el modelo, que por defecto viene dado en 0.5. Comprobamos que no hay la misma cantidad de 1's que de 0's, sino que tenemos un 26.5% de los primeros y un 73.5% d e los segundos. Después sacamos gráficamente la distribución de probabilidad:

```
freq(data$varObj)

hist_targetbinaria(predict(modelo_fin,
newdata=data_test,type="response"),data_test$varObj,"probabilidad")
```



La distribución para los 0's (rojo) tiene buena pinta, pues en la mayor parte de casos los detecta con una alta probabilidad. La distribución de los unos sin embargo está mas repartida, lo que quiere decir que al modelo le costará mucho detectarlos. Si el punto de corte lo mantenemos el 0.5, toda la densidad de puntos que quede a la izquierda del corte los clasificará como 0's cuando realmente no lo son (falsos negativos). Probablemente podamos desplazar el corte un poco hacia la izquierda para que detectar más 1's, en detrimento de los 0's que ahora serán catalogados como 1's (falsos positivos).

Probamos dos puntos de cortes diferente:

```
sensEspCorte(modelo_fin,data_test,"varObj",0.5,"1")

sensEspCorte(modelo_fin,data_test,"varObj",0.265,"1")
```

```
 Accuracy    Sensitivity    Specificity Pos Pred Value Neg Pred V
0.8078740      0.5133531      0.9142551      0.6837945      0.838

 Accuracy    Sensitivity    Specificity Pos Pred Value Neg Pred V
0.7464567      0.7804154      0.7341908      0.5146771      0.902
```

Como vemos mejora la capacidad para reconocer los 1's, es decir la sensibilidad. A cambio perdemos un poco en accuracy y en especificidad (capacidad para reconocer 0's). Buscamos ahora puntos de corte de una forma algo más objetiva, es decir, generando una rejilla con estos posibles puntos. Después nos quedamos con aquellos que tengan la accuracy y el índice de Youden más alto.

```
posiblesCortes<-seq(0,1,0.01)

rejilla<-data.frame(t(rbind(posiblesCortes,sapply(posiblesCortes,function(x)
sensEspCorte(modelo_fin,data_test,"varObj",x,"1")))))

rejilla$Youden<-rejilla$Sensitivity+rejilla$Specificity-1

plot(rejilla$posiblesCortes,rejilla$Youden)

plot(rejilla$posiblesCortes,rejilla$Accuracy)

rejilla$posiblesCortes[which.max(rejilla$Youden)]

rejilla$posiblesCortes[which.max(rejilla$Accuracy)]

sensEspCorte(modelo_fin,data_test,"varObj",0.36,"1")

sensEspCorte(modelo_fin,data_test,"varObj",0.46,"1")
```

Los puntos obtenidos son 0.29 y 0.49. Nos quedamos con el primero por equilibrar la sensibilidad y la especificidad a costa de un poco de accuracy. Los valores son muy cercanos a los del punto que habíamos elegido a ojímetro. Obtenemos ahora la matriz de correlación:

```
pred_test<-factor(ifelse(predict(modelo_fin,data_test,type                    =
"response")>0.29,1,0))

confusionMatrix(pred_test,data_test$varObj, positive = '1')
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 713  80
         1 220 257

               Accuracy : 0.7638
                 95% CI : (0.7394, 0.7869)
    No Information Rate : 0.7346
    P-Value [Acc > NIR] : 0.009565

                  Kappa : 0.4651
```

El modelo reconoce 713 de 193 0's y 257 de 447 1's. Intuíamos ya que iba a ser difícil reconocer los casos positivos. Finalmente predecimos las instancias del test usando el punto de corte óptimo encontrado. Para ello tenemos que preparar los datos y generar las mismas variables que utiliza el modelo. Después de esto, y para acabar la práctica, predecimos y guardamos el archivo:

```
pred_test_fin<-factor(ifelse(predict(modelo_fin,datos_test,type               =
"response")>0.29,1,0))

prediccion <- data.frame(datos_test$ID, pred_test_fin)

saveRDS(prediccion,"FugaPredict_AlejandroSierraFernandez.RDS")
```