ML Supervisado - Obligatorio

De Polsi - del Palacio - González

12/8/2019

Contents

Exploración Inicial

Estructura de los datos y estadísticas descriptivas.

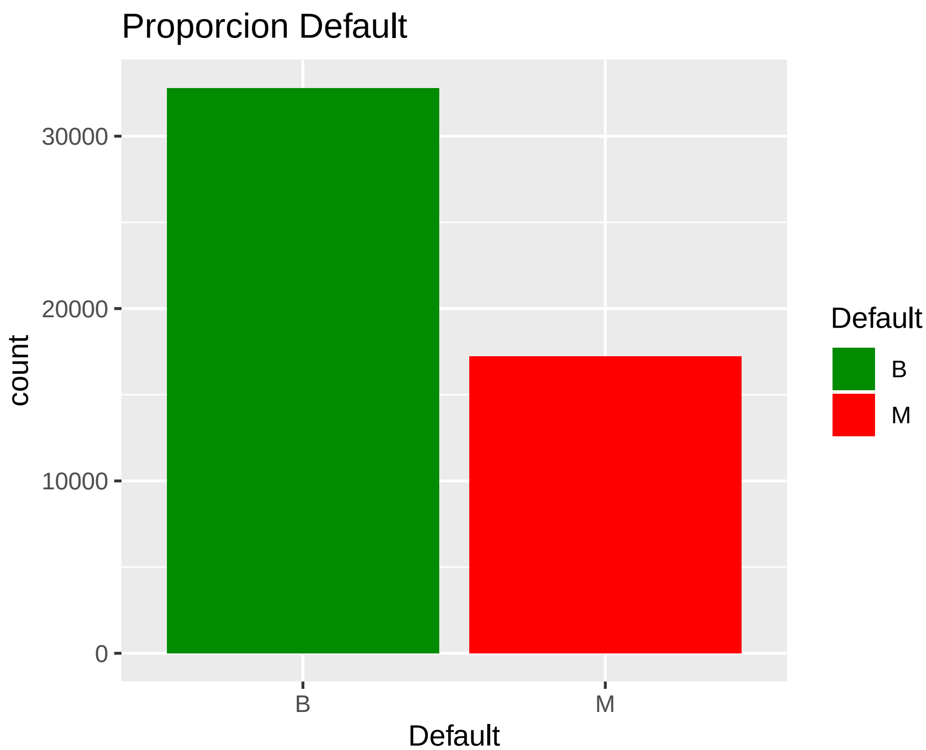
ID	Edad	Sexo	NED	NumBancos	NumFinanc	TDSR	C6M	AumDeu	PI	Cont	Vigen	Ingreso	Default
1	38	F	TT	0	0	21.02	1C	0.0349	0	21648	81429.34	111136.10	B
2	59	F	PNT	0	0	13.27	1C	0.0449	0	185479	152695.98	31017.77	B
3	40	F	ST	1	1	8.28	1C	0.0280	1	127278	88594.60	27935.69	B
4	35	F	TNT	2	0	16.08	1C	0.0702	1	358502	93783.30	49003.78	B
5	36	F	ST	1	1	34.42	2A	0.0036	0	28026	201497.76	45473.32	M
6	21	M	TT	2	0	46.69	4	-0.0510	0	221382	335253.50	51014.09	B

```
## 'data.frame':    50000 obs. of  14 variables:
##  $ ID       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Edad     : int  38 59 40 35 36 21 39 47 32 44 ...
##  $ Sexo     : Factor w/ 2 levels "F","M": 1 1 1 1 1 2 2 2 2 2 ...
##  $ NED      : Factor w/ 6 levels "PNT","PT","SNT",..: 6 1 4 5 4 6 6 5 4 5 ...
##  $ NumBancos: int  0 0 1 2 1 2 1 0 3 2 ...
##  $ NumFinanc: int  0 0 1 0 1 0 2 0 3 1 ...
##  $ TDSR     : num  21.02 13.27 8.28 16.08 34.42 ...
##  $ C6M      : Factor w/ 7 levels "1A","1C","2A",..: 2 2 2 2 3 6 2 2 2 6 ...
##  $ AumDeu   : num  0.0349 0.0449 0.028 0.0702 0.0036 ...
##  $ PI       : int  0 0 1 1 0 0 1 0 0 0 ...
##  $ Cont     : num  21648 185479 127278 358502 28026 ...
##  $ Vigen    : num  81429 152696 88595 93783 201498 ...
##  $ Ingreso  : num  111136 31018 27936 49004 45473 ...
##  $ Default  : Factor w/ 2 levels "B","M": 1 1 1 1 2 1 1 1 1 2 ...
```

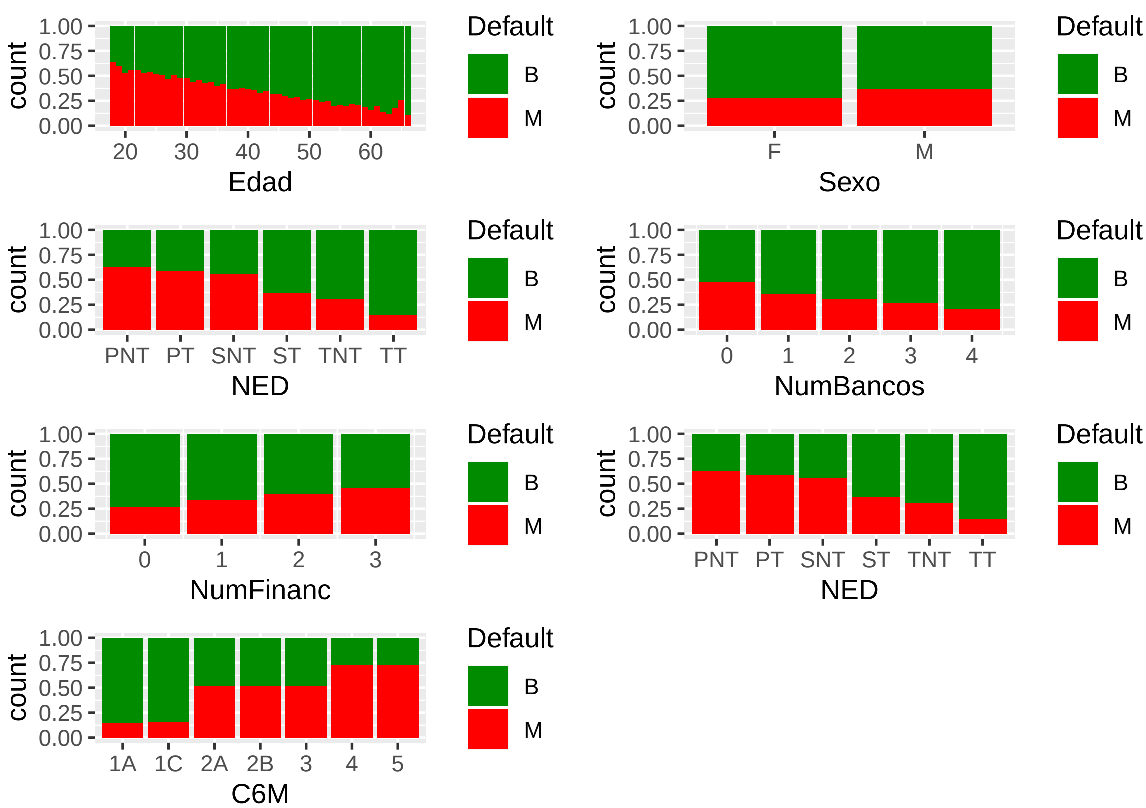
	Edad	Sexo	NED	NumBancos	NumFinanc	TDSR	C6M	AumDeu	PI	Cont	Vigen	Ingreso	Default
	Min. :18.00	F:15037	PNT: 1255	Min. :0.000	Min. :0.000	Min. : 0.47	1A: 4107	Min. :-0.34970	Min. :0.0000	Min. : -6909	Min. : -8999	Min. : 15714	B:32784
	1st Qu.:35.00	M:34963	PT : 2579	1st Qu.:1.000	1st Qu.:0.000	1st Qu.: 10.17	1C:25019	1st Qu.:-0.01810	1st Qu.:0.0000	1st Qu.: 86553	1st Qu.: 117902	1st Qu.: 35542	M:17216
	Median :42.00	NA	SNT: 7652	Median :2.000	Median :1.000	Median : 19.94	2A: 4116	Median : 0.04940	Median :0.0000	Median : 173461	Median : 234772	Median : 56385	NA
	Mean :42.03	NA	ST :12680	Mean :1.626	Mean :1.125	Mean : 33.15	2B: 4098	Mean : 0.04926	Mean :0.1489	Mean : 229303	Mean : 303558	Mean : 78818	NA
	3rd Qu.:49.00	NA	TNT:12876	3rd Qu.:2.000	3rd Qu.:2.000	3rd Qu.: 39.61	3 : 4203	3rd Qu.: 0.11680	3rd Qu.:0.0000	3rd Qu.: 311216	3rd Qu.: 414776	3rd Qu.: 95396	NA
	Max. :66.00	NA	TT :12958	Max. :4.000	Max. :3.000	Max. :2212.35	4 : 4220	Max. : 0.45170	Max. :1.0000	Max. :2125857	Max. :3293045	Max. :1735496	NA
	NA	NA	NA	NA	NA	NA	5 : 4237	NA	NA	NA	NA	NA	NA

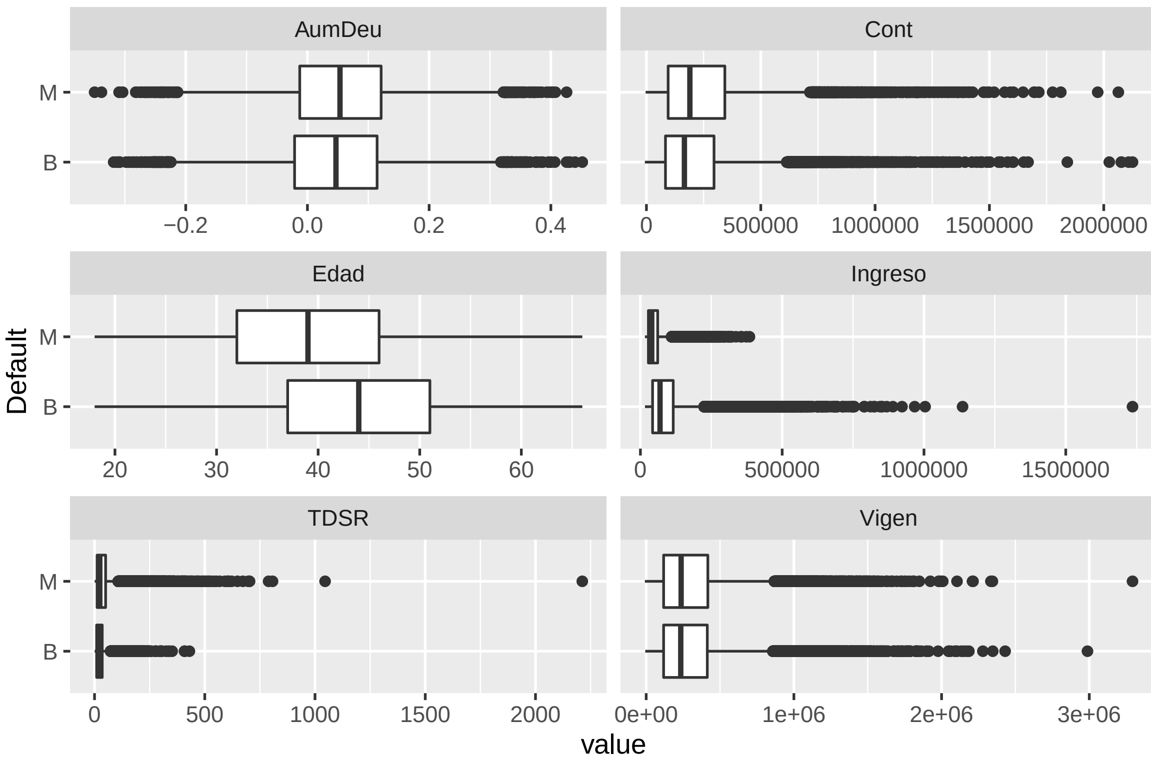
Amálisis Univariado de Variable indpeneidente "Default".

	Count	Percentage
B	32784	65.57
M	17216	34.43



Análsis vibariado de las variables del dataset contra la variable “Default”.





Creación de Variables

A partir de la exploración inicial y la visualización gráfica, se realizaron agrupaciones en variables como *Edad* y *Calificación en el historial de pago*. Se binarizan las variables *Cantidad de Bancos* y *Cantidad de Financieras no Bancarias*. También se realizan operaciones aritmeticas con variables continuas como *Ingreso*, *Contingencias* y *Créditos vigentes*.

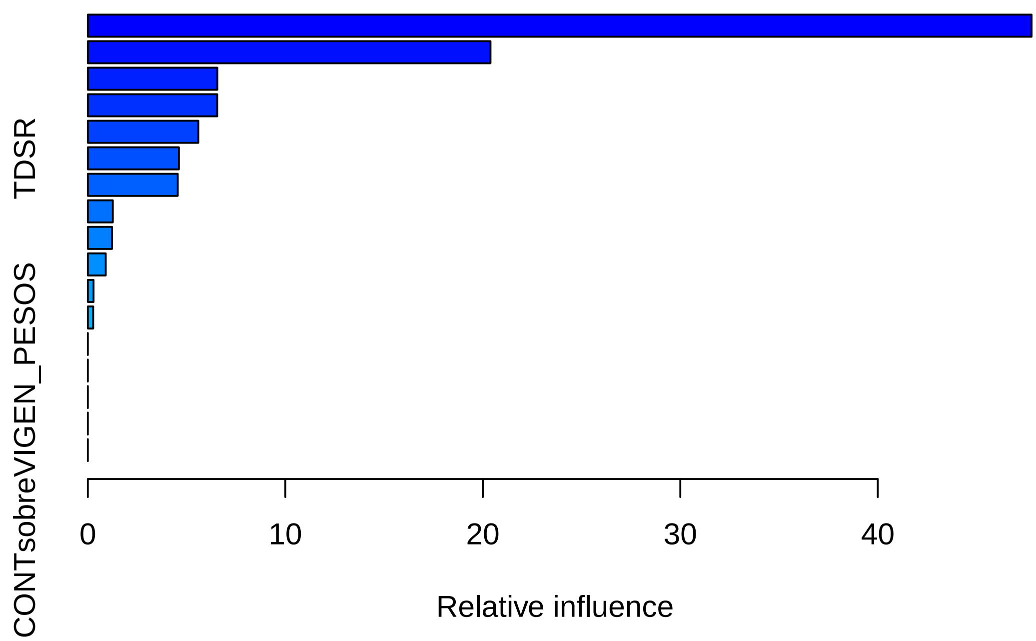
	Default	Rangos_C6M	Rangos_EDAD	SOLO_BANCOS	SOLO_FINANC	CONTsobreINGRESO	VIGEN_PESOS	CONTsobreVIGEN_PESOS
	Min. :0.0000	AL_DIA :29126	MENOR_25 : 1803	0:35798	0:42706	Min. :-0.1454	Min. : -341950	Min. : 0.00000
	1st Qu.:0.0000	ATR_MODERADO:12417	ENTRE_25_34: 9995	1:14202	1: 7294	1st Qu.: 1.1991	1st Qu.: 4480258	1st Qu.: 0.00840
	Median :0.0000	ATR_ALTO : 8457	ENTRE_35_44:18008	NA	NA	Median : 2.8188	Median : 8921332	Median : 0.01966
	Mean :0.3443	NA	ENTRE_45_54:14414	NA	NA	Mean : 4.6665	Mean : 11535207	Mean : 0.05628
	3rd Qu.:1.0000	NA	MAYOR_55 : 5780	NA	NA	3rd Qu.: 5.9605	3rd Qu.: 15761485	3rd Qu.: 0.04580
	Max. :1.0000	NA	NA	NA	NA	Max. :93.2003	Max. :125135710	Max. :42.15631

Importancia de las variables

Para pbservar la importancia de las variables, en primer lugar se analiza la **correlación** entre las variables continuas y la variable dependiente.

	x
Ingreso	-0.2885037
CONTsobreINGRESO	0.2761365
TDSR	0.1686182
NumBancos	-0.1676377
NumFinanc	0.1395705
Cont	0.0788715
AumDeu	0.0362410
CONTsobreVIGEN_PESOS	0.0072729
Vigen	0.0060343
VIGEN_PESOS	0.0060343

En segunda instancias se corre un modelo **Boosting**, sin realizar optimización de hiperparametros, en todo el dataset para chequear la *influencia relativa* de cada predictor en el descenso promedio del error de clasificación.



	var	rel.inf
Rangos_C6M	Rangos_C6M	47.7869175
Ingreso	Ingreso	20.3899028
NED	NED	6.5593173
PI	PI	6.5512208
SOLO_FINANC	SOLO_FINANC	5.5955932
TDSR	TDSR	4.6091505
CONTsobreINGRESO	CONTsobreINGRESO	4.5523575
NumBancos	NumBancos	1.2636216
NumFinanc	NumFinanc	1.2252097
Sexo	Sexo	0.9054791
Rangos_EDAD	Rangos_EDAD	0.2896757
SOLO_BANCOS	SOLO_BANCOS	0.2715544
AumDeu	AumDeu	0.0000000
Cont	Cont	0.0000000
Vigen	Vigen	0.0000000
VIGEN_PESOS	VIGEN_PESOS	0.0000000
CONTsobreVIGEN_PESOS	CONTsobreVIGEN_PESOS	0.0000000

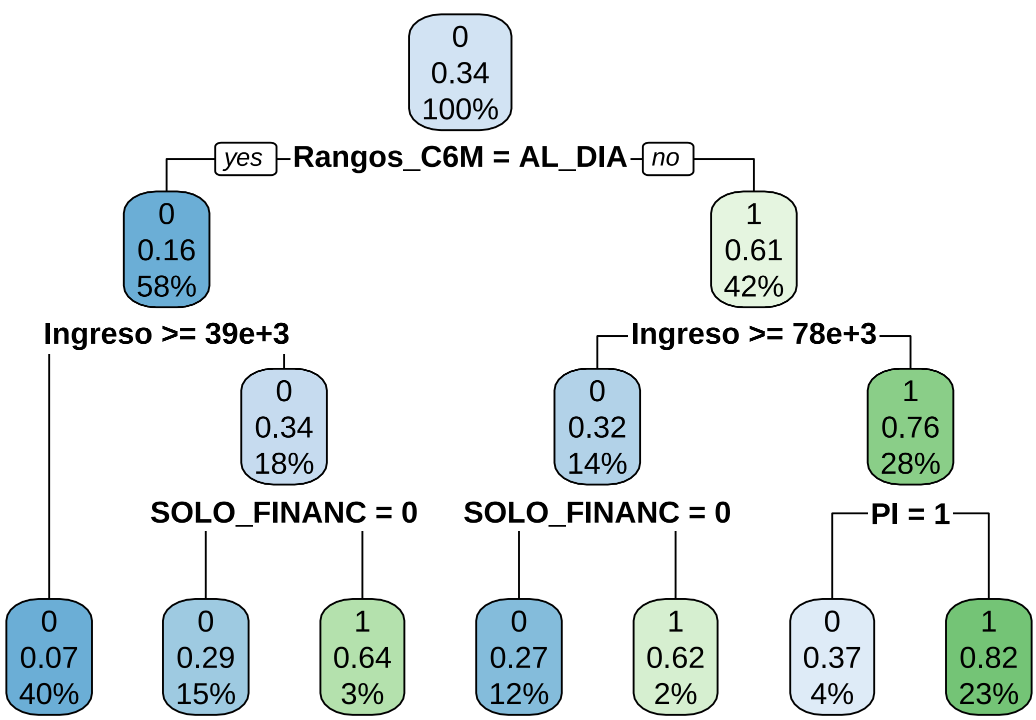
Con los datos obtenidos se pdrían tomar desiciones con las variables a utilizar en la estimación de los modelos. En el caso de este trabajo decidimos mantener todos los predictores.

Estimacion de Modelos

Árbol de clasificación. Modelo Base

Se opta por utilizar un **Árbol de Clasificación** como modelo base. Otra buena opción hubiera sido tomar como base un modelo de *Regresión Loística.* Por la velocidad y simplicidad con la que se logran las estimaciones este modelo es la base de comparción para la elección de modelos más complejos que se estiman posteriormente.

```
##
## Classification tree:
## rpart(formula = y ~ ., data = df.train, method = "class", minbucket = 10,
##     maxdepth = 6)
##
## Variables actually used in tree construction:
## [1] Ingreso     PI          Rangos_C6M  SOLO_FINANC
##
## Root node error: 13785/40000 = 0.34463
##
## n= 40000
##
##         CP nsplit rel error  xerror      xstd
## 1 0.263185      0   1.00000 1.00000 0.0068951
## 2 0.144432      1   0.73682 0.73682 0.0063149
## 3 0.030395      2   0.59238 0.59391 0.0058537
## 4 0.014654      3   0.56199 0.56366 0.0057399
## 5 0.010446      4   0.54733 0.54857 0.0056808
## 6 0.010000      6   0.52644 0.53856 0.0056407
```



Random Forest

El primer modelo estimado para competir con el *Modelo Base* corresponde a un **Random Forest** sin optimización de hiperparamatros. Se estima con **400** *arboles* y la cantidad aleatoria de *predictores* seleccionados para las divisiones igual a **6**.

Se presenta la *importancia de las variables* para el modelo estimado así como los resultados de la matriz de confusión en *train* set.

```
##
## Call:
##  randomForest(formula = y ~ ., data = df.train, mtry = 6, ntree = 400,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 400
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 12.78%
## Confusion matrix:
##       0     1 class.error
## 0 23947  2268  0.08651535
## 1  2845 10940  0.20638375
```

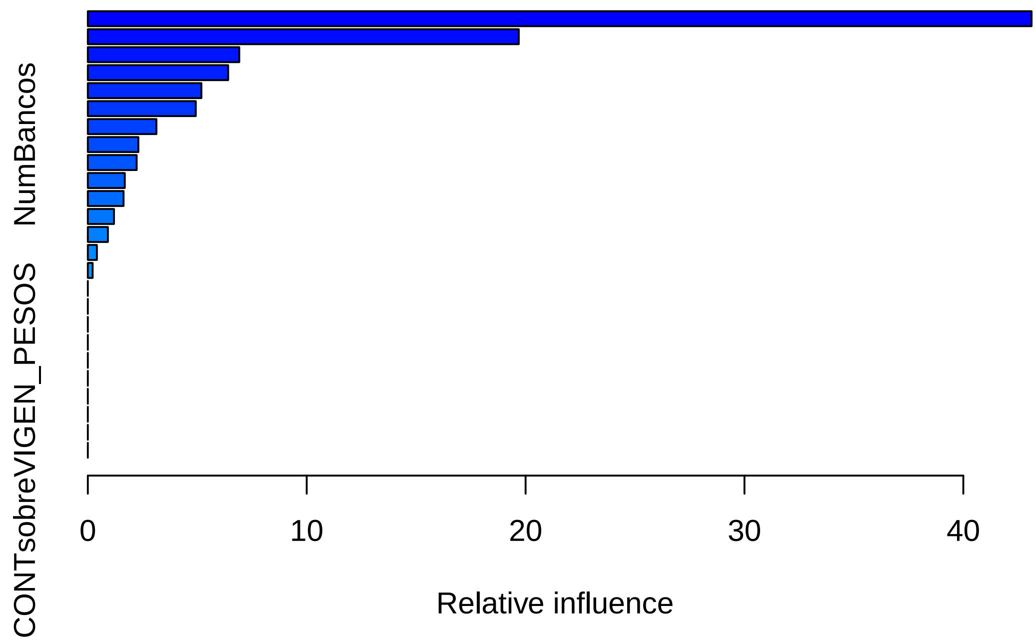
	x
Rangos_C6M	4558.2446
Ingreso	2538.8180
TDSR	1614.3283
CONTsobreINGRESO	1320.1107
NED	1140.5030
AumDeu	897.8067
PI	895.6256
Cont	843.2257
CONTsobreVIGEN_PESOS	728.4842
VIGEN_PESOS	661.6286
Vigen	658.9172
NumBancos	507.1986
SOLO_FINANC	451.2254
Rangos_EDAD	435.0063
NumFinanc	431.5288
Sexo	228.7028
SOLO_BANCOS	152.6711

Boosting

Por último se estima un modelo de **Boosting** sin optimización de hiperparametros. Se utiliza una cantidad de *arboles* igual a **400**, Un *coeficiente de aprendizaje* para cada arbol de **0.1** y una cantidad de *interacciones entre variables de cada arbol* igual a **1**

Se presenta la *importancia de las variables* para el modelo estimado así como los resultados en *train* set.

```
##           Reference
## Prediction     0     1
##          0 24343  2989
##          1  1872 10796
```



	var	rel.inf
Rangos_C6M.L	Rangos_C6M.L	43.1180421
Ingreso	Ingreso	19.6887767
PI1	PI1	6.9186899
TDSR	TDSR	6.4124437
CONTsobreINGRESO	CONTsobreINGRESO	5.1884967
SOLO_FINANC1	SOLO_FINANC1	4.9315407
NEDTT	NEDTT	3.1339209
NumBancos	NumBancos	2.3089036
NumFinanc	NumFinanc	2.2306663
NEDSNT	NEDSNT	1.6896903
SexoM	SexoM	1.6330557
Rangos_EDAD.L	Rangos_EDAD.L	1.1952591
SOLO_BANCOS1	SOLO_BANCOS1	0.9165712
NEDPT	NEDPT	0.4143901
AumDeu	AumDeu	0.2195531
NEDST	NEDST	0.0000000
NEDTNT	NEDTNT	0.0000000
Cont	Cont	0.0000000
Vigen	Vigen	0.0000000
Rangos_C6M.Q	Rangos_C6M.Q	0.0000000
Rangos_EDAD.Q	Rangos_EDAD.Q	0.0000000
Rangos_EDAD.C	Rangos_EDAD.C	0.0000000
Rangos_EDAD^4	Rangos_EDAD^4	0.0000000
VIGEN_PESOS	VIGEN_PESOS	0.0000000
CONTsobreVIGEN_PESOS	CONTsobreVIGEN_PESOS	0.0000000

Principales Hiperparametros a optimizar

Arbol de Clasificación

En Árboles de Clasificación, la importancia de optimizar hiperparametros es no caer en *overfitting* que reduzca la capacidad predictiva. Los principales hiperparametros a optimizar en el la *muestra de test* son el *tamaño del arbol*, en *r maxdepht* o el *proceso de podado*, representado por *cp* en *r*.

Random Forest

El principal hiperparametro a optimizar en Random forest será el *número de predictores* utilizado en cada división (en *r mtry*). Otros hiperparametros del modelo pueden ser el número mínimo de *observaciones* que deben tener los *nodos terminales*, en *r nodesisze* y el *Número de Arboles* que se ajustan en el modelo (en *r ntree*). El número de arboles y el crecimiento que estos pueden tener no son considerados *hiperparametros críticos*. Estos es así ya que en los modelos donde se aplica muestreo con *boostrapping* se elimina el riego de caer en *overfitting*.

Boosting

En boosting identificamos 3 hiperparametros críticos a optimizar: La *cantidad de árboles*, en *r n.trees*. Vale aclarar que boosting no hace uso de muestreo *bootstrapping*, por lo que cada árbol construido depende en gran medida de los árboles previos. Esto genera un riesgo alto *overfitting* para un númro grande de árboles.
Por otro lado se tiene el parámetro de aprendizaje, en *r shrinkage*, el cual controla el ritmo al que aprende el modelo de cada árbol. Con coeficientes de aprendizaje muy bajos será necesario un mayor número de árboles para obtener buenos resultados. Finalmente tenemoe la cantidad de *interacciones* entre variables de cada árbol pérmitidas.

Optimización de hiperparámetros

Por razones de economía de tiempo se presentan los códigos utilizados para la optimización de cada uno de los modelos y se explicitan los resultados obtenidos.

Árbol de clasificación

Hiperparámetros seleccionado a optimizar en Árbol de Clasificación para el proceso de podado es: *cp*
En función de las optimizaciones realizadas, los valores son:
cp : 0.000 - 0.005 - 0.010
Minimo Error_dt : 0.1839

```
n <- nrow(datos)
train.x = datos[1:(n*0.8),-c(1,2,8,14)]
train.y = datos$Default[1:(n*0.8)]

test.x = datos[((n*0.8)+1):n,-c(1,2,8,14)]
test.y = datos$Default[((n*0.8)+1):n]

df.train = data.frame ("y" = train.y, train.x)


#Prune el modelo
modelo_dtp = prune(modelo_dt) # modelo_dt es el modelo base especificado en el punto 2
alfas <- seq(0,0.1, by = 0.005)
```

```r
error_dt = c()
for(i in alfas){
  modelo_dtp = prune(modelo_dt, cp = i)
  y_hat_dt = ifelse (predict(modelo_dtp , as.data.frame(test.x))[,2]>0.5,1,0)
  tabla_dt = table(y_hat_dt, test.y)
  er_dt = 1- sum(diag(tabla_dt))/sum(tabla_dt)
  error_dt <- c(error_dt, er_dt)
}

#[1] 0.1839 0.1839 0.1839 0.1949 0.1949 0.1949 0.1949 0.2028 0.2028 0.2028 0.2028
#   0.2028 0.2028 0.2028
#[15] 0.2028 0.2028 0.2028 0.2028 0.2028 0.2028 0.2028

# Minimo Error_dt : 0.1839
# cp 0.000 0.005 0.010
```

En este caso la mejor *accuracy* obtenida es la misma que el para el *podado* utilizado en el *modelo base* por lo que no se volvió a correr el modelo.

Random Forest

Hiperparámetros a optimizar para el Random Forest son: *ntree* y *mtry*.
En función las optimizaciones realizadas, los valores son:
ntree = 400
mtry = 7
Error rate 0.1251
Accuracy 0.8749

```r
n <- nrow(datos)
train.x = datos[1:(n*0.8),-c(1,2,8,14)]
train.y = datos$Default[1:(n*0.8)]

test.x = datos[((n*0.8)+1):n,-c(1,2,8,14)]
test.y = datos$Default[((n*0.8)+1):n]

df.train = data.frame ("y" = train.y, train.x)

# Hiperparametros a Optimizar mtry y ntree
vectorTrees <- seq(100,400, by = 100)
datosRF <- c("nt"=  NA, "mtry" = NA, "er" = NA, "accuracy" = NA)

for(i in vectorTrees){
  for(mtry in 1:17){
    rf = randomForest( y  ~ . , data = df.train, mtry = mtry, ntree = i, importance = TRUE)
    iter_y_hat_bag = predict(rf, test.x)
    iter_tabla_bag = table(iter_y_hat_bag, test.y)
    iter_er_bag = 1- sum(diag(iter_tabla_bag))/sum(iter_tabla_bag)

    resAux <- c(i, mtry, iter_er_bag, (1 - iter_er_bag))
    datosRF <- rbind(datosRF, resAux)
    cat(mtry," ") #printing the output to the console

  }
}
```

```r
MinDatosRF <- min(datosRF[,3])

OrdenDatosRF <- sort_abs(datosRF[,3])

# Better Acuuracy is:
# Árboles  400  MTRY  7  - Error 0.1251  Accuracy 0.8749
```

Boosting opción 1

Hiperparámetros seleccionado a optimizar en Boosting para el proceso 2 son: *interaction.depth* , *shrinkage*, *n.trees*.
En función de las optimizaciones realizadas, los valores son:
n.trees : 400
interaction.depth : 3
shrinkage : 0.100

```r
grid <- expand.grid(n.trees = seq(100,400, by=100), interaction.depth=c(1:3),
                    shrinkage=c(0.001,0.005,0.01,0.05,0.1), n.minobsinnode = 10)

ctrl <- trainControl(method = "cv",number = 5, allowParallel = FALSE)

set.seed(124) #for reproducability

unwantedoutput <- capture.output(GBMModel <- train(y~.,data = df.train,
                                                   method = "gbm", trControl = ctrl, tuneGrid = grid)

GBMModel$results %>% as.data.frame() %>%
  select(1:2, 4:5) %>%
  filter(Accuracy > 0.85) %>%
  arrange(desc(Accuracy))

#Better  n.trees = 400  , interaction.depth = 3 , shrinkage =  0.100
# Accuracy = 0.882675
```

Boosting opción 2

Hiperparámetros seleccionado a optimizar en Boosting para el proceso 2 son: *interaction.depth* , *shrinkage*, *n.trees*.
En función de las optimizaciones realizadas, los valores son:
n.trees : 900
interaction.depth : 4
shrinkage : 0.100

```r
set.seed(612)
vectorLambda <- c(0.001, 0.005, 0.01, 0.05, 0.1)
vectorArboles <- seq(100,1000, by = 200)
vectorD <- c(1:5)

datosDF <- c("nt"=  NA, "d" = NA, "lambda" = NA, "er" = NA)

for(i in vectorArboles){
  for (j in vectorD){
    for (h in vectorLambda){
```

```
      modgbm <- gbm(Default ~ ., data = datos[,-c(1,2,8)], n.trees = i, interaction.depth = j,
                     shrinkage = h, bag.fraction = 0.5, train.fraction = 0.8, cv.folds = 5 ,
                     n.minobsinnode = 10 ,keep.data = TRUE)

      er <- mean(modgbm$cv.error)
      resAux <- c(i, j, h, er)
      datosDF <- rbind(datosDF, resAux)
    }

  }
  cat(i," ") #printing the output to the console
}

OrdenDatosDF <- sort_abs(datosDF[,4])
MinDatosDF <- min(datosDF[,4])

#Better  n.trees = 900  , interaction.depth = 4 , shrinkage =  0.100
# cv.error = 0.5565731
```

El hecho que el óptimo se encontrara en mayor número de árboles de la iteración, tanto en Random Forest como en Boosting (donde además la optimización del hiperparámetro de aprendizaje es el más alto iterado) indica que se debería haber permitido iterar con mayor número de árboles.

Error de clasificación y Modelo elegido

Árbol de clasificación

```
##         test.y
## y_hat_dt    0    1
##        0 5974 1244
##        1  595 2187

## [1] 0.1839
```

Random Forest

```
##            test.y
## y_hat_rf_op    0    1
##           0 6028  706
##           1  541 2725

## [1] 0.1247
```

Boosting

```
##               test.y
## y_hat_boost_op    0    1
##              0 6058  684
##              1  511 2747

## [1] 0.1195
```

En función del error de clasificación nos quedamos con el modelo de **Boosting**. Es importante señalar que el modelo no solo es el que presenta menor error de clasificación sino que además es el que consigue mejor clasificación en los casos de éxito (**Default**).