Tarea ML Clasificación Binaria

Miguel Benayas Penas

10/6/2021

Contents

1 Objetivo

Demostración de los conocimientos adquiridos en el módulo de ML con R, mediante un problema de clasificación binaria.

Se busca un dataset de tamaño mediano/pequeño para cumplir el objetivo disminuyendo el riesgo de tiempos de computación muy altos, en base a los recursos disponibles.

El dataset escogido procede de la web Rdatasets, mencionado en el pdf de la práctica. Dicho dataset cumple con las dos recomendaciones "aproximadamente más de 300 observaciones y 5 variables input posibles, de las cuales al menos una debe ser categórica".

De los Rdatasets de clasificación, HMDA presenta un número de columnas alto para así poder tener más libertad cuando se evalúen distintos subconjutos de variables input. Además presenta un número de columnas alto (todo respecto a ese archivo de datasets) disminuyendo el riesgo de overfitting.

Luego se comprobará el número de observaciones de la clase minoritaria de la variable objetivo,

2 Presentación de datos

En esta sección se realizará la importación del data set, data cleaning y un EDA básico

2.1 Data Loading

En primer lugar, se cargan una serie de paquetes útiles para proyectos de machine learning.

```
# Library loading
rm(list = ls())

suppressPackageStartupMessages({
  library(data.table)
  library(dplyr)
  library(caret)
  library(scales)
  library(ggplot2)
  library(stringi)
  library(stringr)
  library(dataPreparation)
  library(knitr)
  library(kableExtra)
  library(ggpubr)
  library(tictoc)
  library(ggeasy)
  library(lubridate)
  library(inspectdf)
  library(Rcpp)
  library(car) # recode
  library(questionr ) # freq
  library(gbm)
  library(randomForest)
  library(xgboost)
  library(MASS)
  library(dummies)
  library(parallel)
  library(doParallel)
  library(kernlab)
  library(reshape)
})
```

Se importa el dataset y se guarda como data frame.

```
datos <- fread( file = 'HMDA.csv', nThread = 2)
datos <- as.data.frame(datos)
```

2.2 Data cleaning

Se elimina la variable V1 al carecer de potencial predictivo, ya que se trata del ID de las obervaciones.

Se verifica que hay tres observaciones duplicadas. Se procede a su eliminación.

Se comprueba que presenta el mismo número de observaciones y columnas que lo indicado en la descripción, 2380 y 14 respectivamente.

```
datos$V1 <- NULL
cat("Número de observaciones duplicadas:", sum(duplicated(datos)),'\n')
```

```
## Número de observaciones duplicadas: 3
```

```
datos <- datos[!duplicated(datos),]
cat("El número de predictores es de:",ncol(datos),'\n')
```

```
## El número de predictores es de: 14
```

```
cat("El número de filas es:",nrow(datos),'\n')
```

```
## El número de filas es: 2377
```

Se comprueba si los datos han sido correctamente formateados.

```
 str(datos)
```

```
## 'data.frame':    2377 obs. of  14 variables:
##  $ deny     : chr  "no" "no" "no" "no" ...
##  $ pirat    : num  0.221 0.265 0.372 0.32 0.36 ...
##  $ hirat    : num  0.221 0.265 0.248 0.25 0.35 ...
##  $ lvrat    : num  0.8 0.922 0.92 0.86 0.6 ...
##  $ chist    : int  5 2 1 1 1 1 1 2 2 2 ...
##  $ mhist    : int  2 2 2 2 1 1 2 2 2 1 ...
##  $ phist    : chr  "no" "no" "no" "no" ...
##  $ unemp    : num  3.9 3.2 3.2 4.3 3.2 ...
##  $ selfemp  : chr  "no" "no" "no" "no" ...
##  $ insurance: chr  "no" "no" "no" "no" ...
##  $ condomin : chr  "no" "no" "no" "no" ...
##  $ afam     : chr  "no" "no" "no" "no" ...
##  $ single   : chr  "no" "yes" "no" "no" ...
##  $ hschool  : chr  "yes" "yes" "yes" "yes" ...
```

Se efectúa una copia del dataset original (datMod) sobre los que se aplicarán los cambios

Se cambia el tipo de la variable objetivo (deny) por factor.

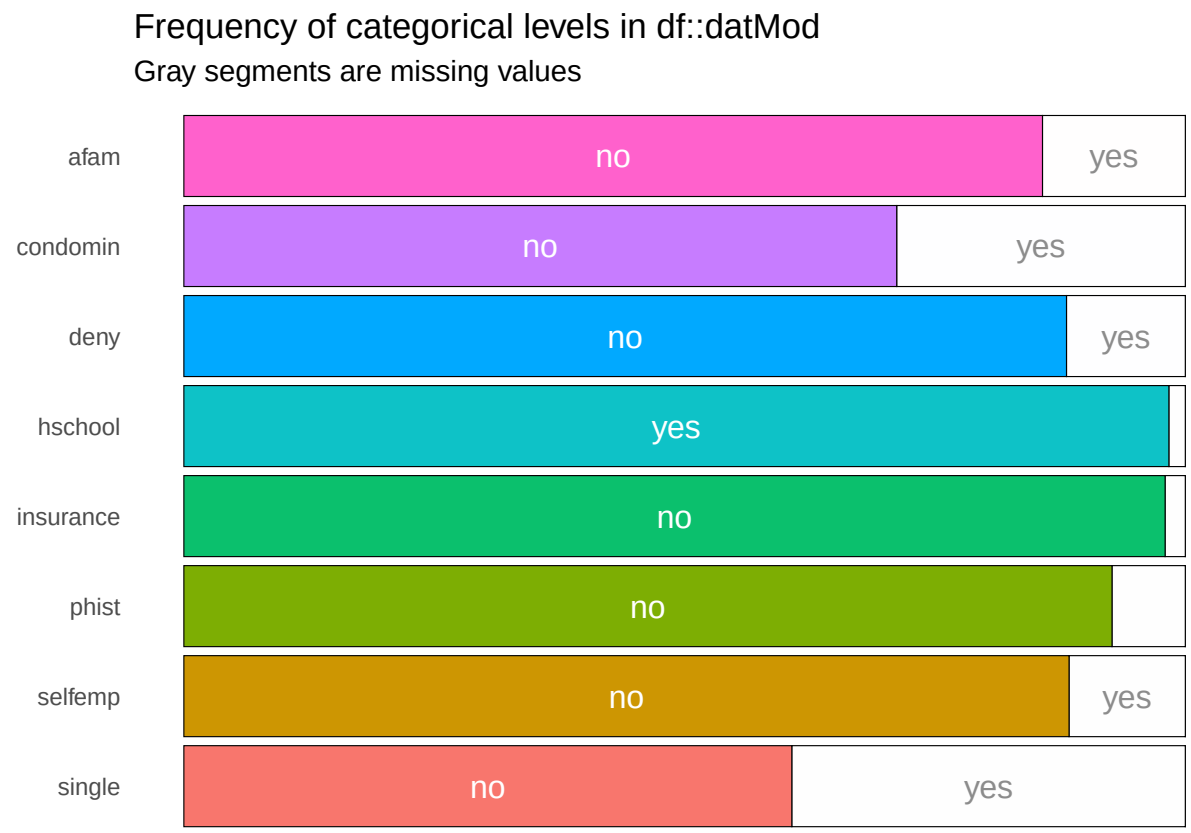
Las variables mhist y chist deben ser categóricas, como se expone en la descripción del dataset. Esto también se puede intuir observando el tipo entero y la muestra de los valores que toman.

```
datMod<- copy(datos)
# Factorizar la variable objetivo
datMod$deny <- as.factor(datMod$deny)
datMod$chist <- as.character(datMod$chist)
datMod$mhist <- as.character(datMod$mhist)
```

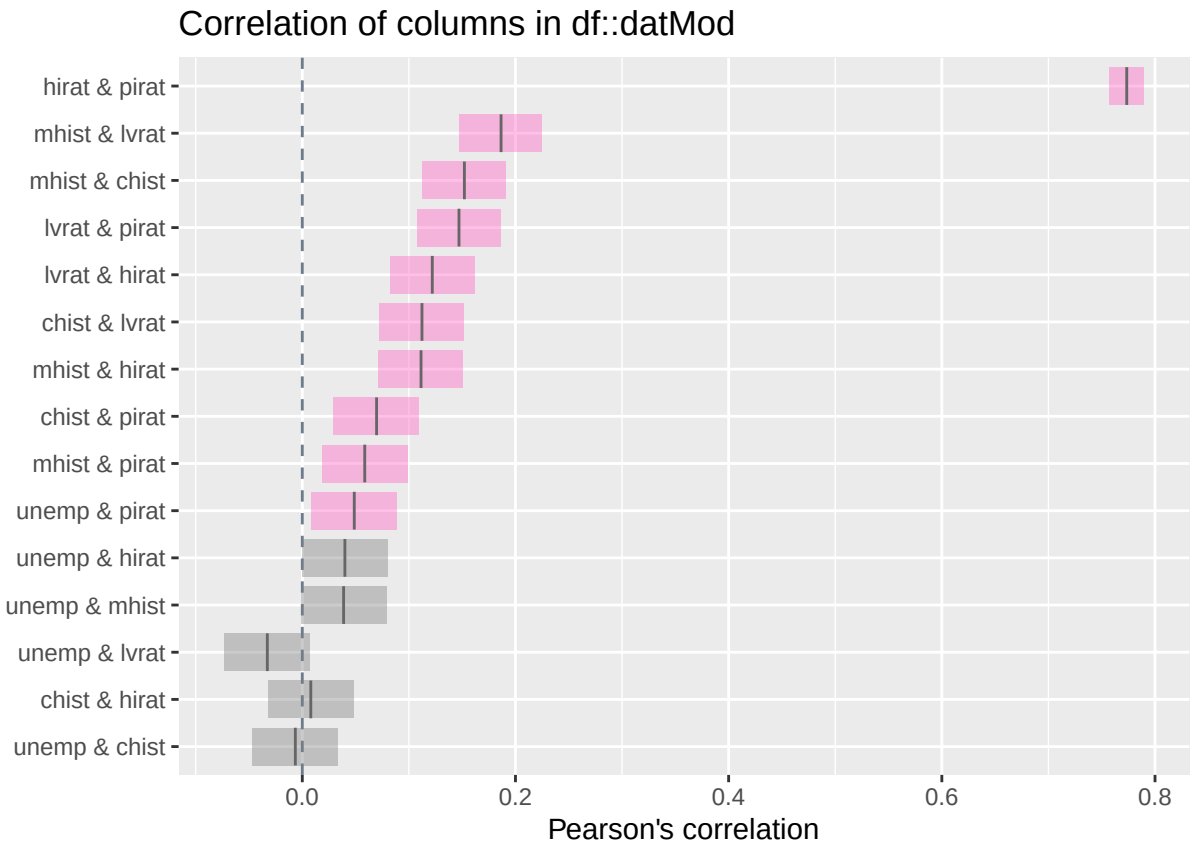
2.3 EDA

Hacemos una exploración del dataset por medio del paquete inspectdf

```
# categorical plot
x <- inspect_cat(datMod)
show_plot(x)
```

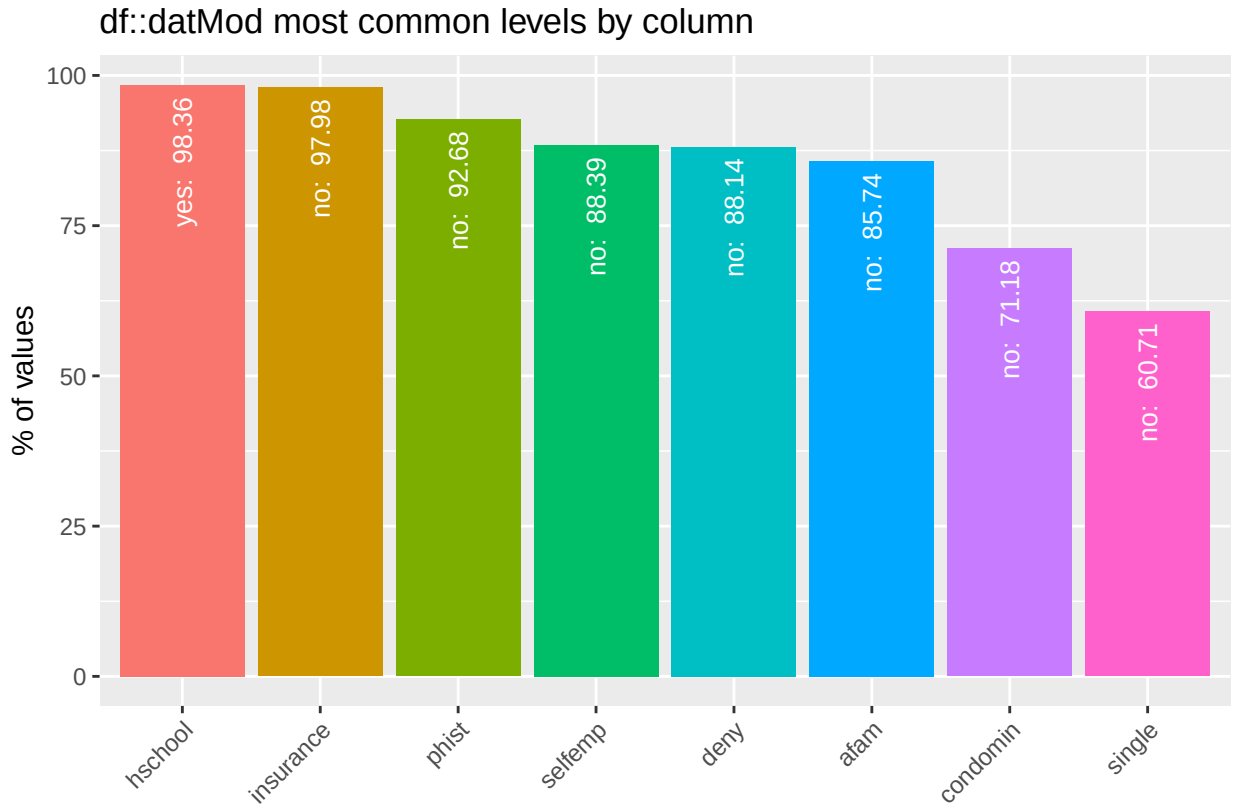



```
# correlations in numeric columns
x <- inspect_cor(datMod)
show_plot(x)
```



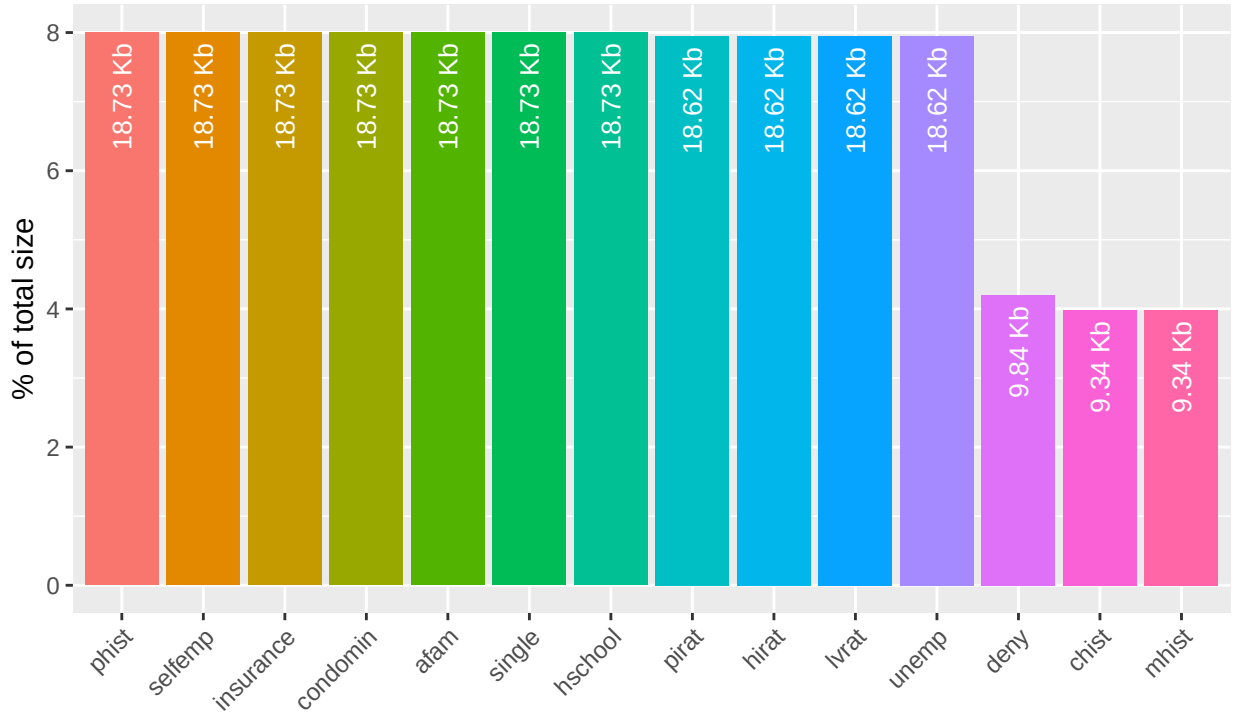

```
# feature imbalance bar plot
x <- inspect_imb(datMod)
show_plot(x)
```

```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```



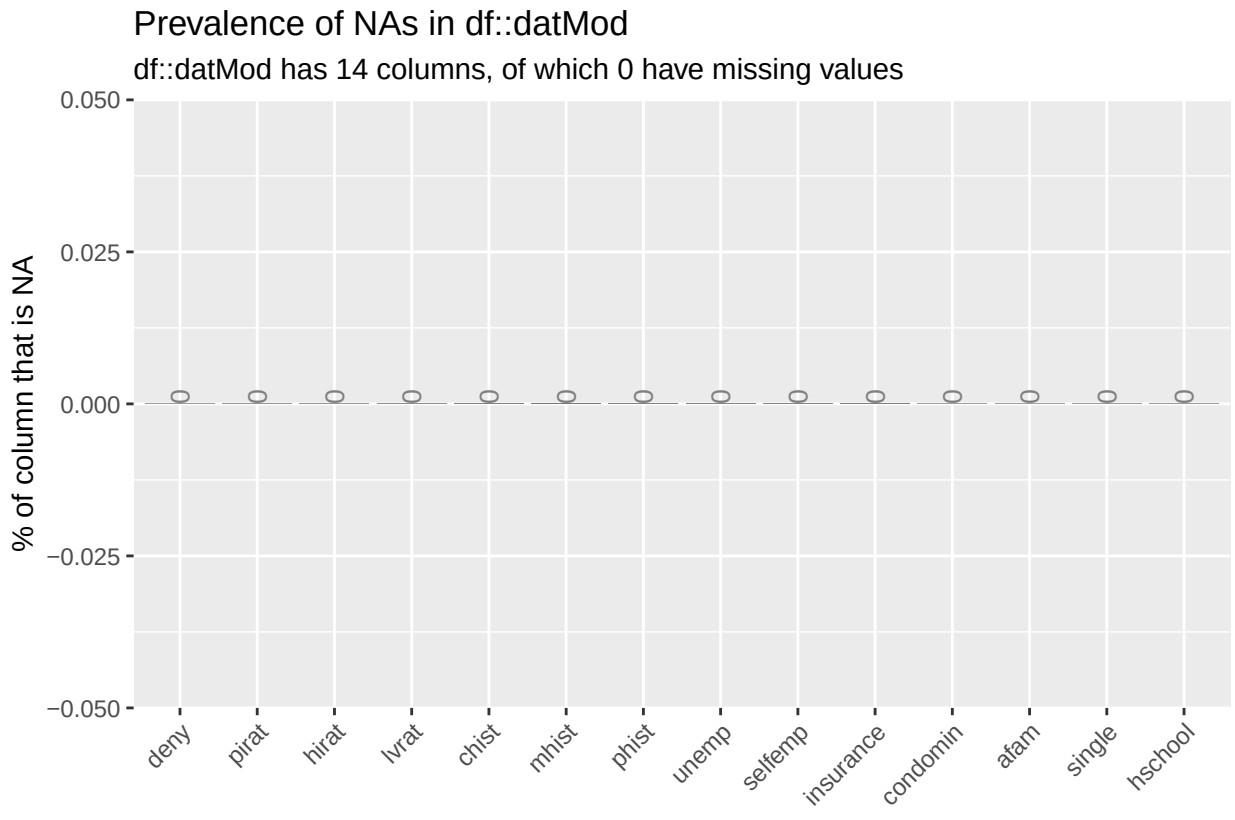

```
# memory usage barplot
x <- inspect_mem(datMod)
show_plot(x)
```

```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```

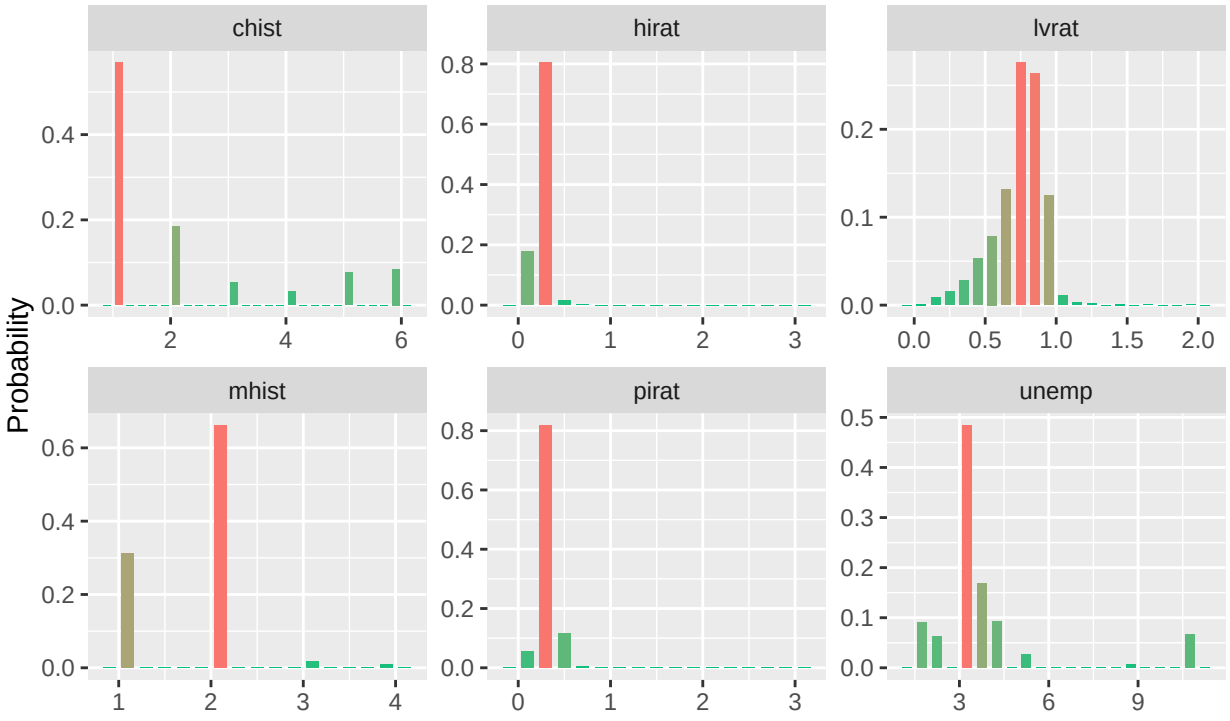



```
# missingness barplot
x <- inspect_na(datMod)
show_plot(x)
```

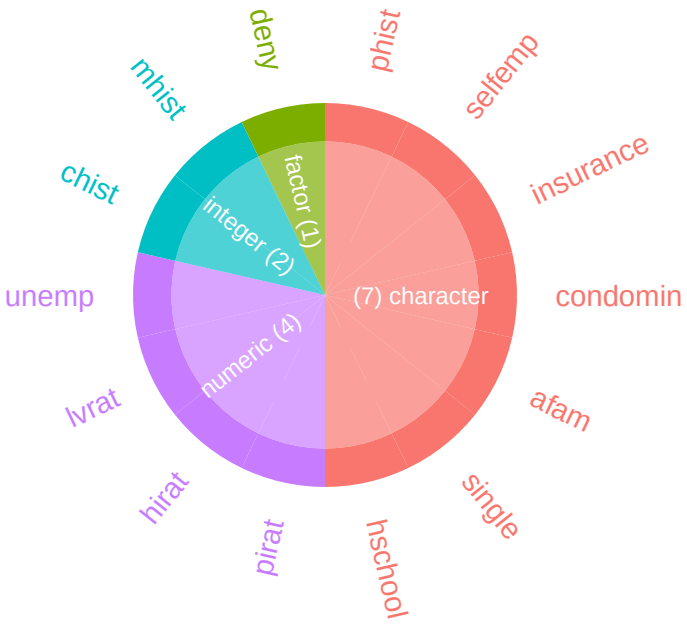
```
## Warning: `guides(<scale> = FALSE)` is deprecated. Please use `guides(<scale> =
## "none")` instead.
```




```
# histograms for numeric columns
x <- inspect_num(datMod)
show_plot(x)
```




```
# barplot of column types
x <- inspect_types(datMod)
show_plot(x)
```



varias conclusiones:

- La mayoría de los predictores son variables categóricas. Por tanto, es posible que modelos basados en árboles proporcionen altos ¿ROC? respecto a otros algoritmos, dado el tipo de superficie de predicción que presentan.
- Algunas variables categóricas tienen frecuencias muy desbalancedas o pequeñas, lo que aumenta el riesgo a padecer overfitting. En concreto, se eliminarán las variables hschool e insurance por su bajo poder predictivo, ya que presentan categorías dominantes con frecuecias mayores al 95% en este dataset con pocas observaciones.
- Variables cuantitativas no tienen unas correlaciones con valores absolutos por encima de 0.95, reduciendo así el riesgo de overfitting. A simple vista, no parecen tener valores fuera de rango. Sin embargo, la presencia de valores estrictamente 0 podrían esconder valores missing. Además algunas variables podrían tener menos de 10 valores distintos.
- No hay presencia explícita de missings. No obstante, se ha de evaluar si hay valores imposibles como cero o outliers los cuales contaría como datos faltantes. Si hubiera missings, habría que recurrir a eliminar, recategorizar o imputar en función de su porcentaje en cada variable.
- La variable objetivo deny tiene categorías desbalanceadas. Usar stratifiedKfolds como técnica de remuestreo.

En primer lugar, se muestran las frecuencias de la variable objetivo para asegurar que, en efecto, hay más de 100 observaciones de la categoría minoritaria.

```
freq(datMod$deny)
```

```
##        n    % val%
## no  2095 88.1 88.1
## yes  282 11.9 11.9
```

Se estudian las frecuencias de las variables cuantitativas para ver si tienen más de 10 valores diferentes

```
sapply(Filter(is.numeric, datMod), function(x) length(unique(x)))
```

```
## pirat hirat lvrat unemp
##   519   500  1537    10
```

La variable unemp tiene solo 10 valores distintos, lo cual aumenta el riesgo de overfitting. Se muestran a continuación las frecuencias de valores y se observa que hay dos con frecuencias inferiores a 5%.

```
freq(datMod$unemp, sort = 'dec')
```

```
##                    n    % val%
## 3.20000004768372 876 36.9 36.9
## 3.09999990463257 274 11.5 11.5
## 3.90000009536743 229  9.6  9.6
## 4.30000019073486 220  9.3  9.3
## 1.79999995231628 217  9.1  9.1
## 3.59999990463257 174  7.3  7.3
## 10.6000003814697 158  6.6  6.6
## 2                148  6.2  6.2
## 5.30000019073486  65  2.7  2.7
## 8.89999961853027  16  0.7  0.7
```

Se hará una codificación tipo lumping a esos dos valores con los más cercanos que presenten frecuencias superiores al límite impuesto de 5% por tratarse de un dataset pequeño.

```
datMod$unemp <-car::recode(datMod$unemp, "'5.30000019073486' = '4.30000019073486'")

datMod$unemp <-car::recode(datMod$unemp, "'8.89999961853027' = '10.6000003814697'")

datMod$unemp <- as.character(datMod$unemp)

freq(datMod$unemp, sort = 'dec')
```

```
##                    n    % val%
## 3.20000004768372 876 36.9 36.9
## 4.30000019073486 285 12.0 12.0
## 3.09999990463257 274 11.5 11.5
## 3.90000009536743 229  9.6  9.6
## 1.79999995231628 217  9.1  9.1
## 10.6000003814697 174  7.3  7.3
## 3.59999990463257 174  7.3  7.3
## 2                148  6.2  6.2
```

2.4 Tratamiento de valores missing y outliers

Se muestran los valores mínimos de las variables numéricas:

```
sapply(Filter(is.numeric, datMod), function(x) min(x , na.rm = TRUE))
```

```
## pirat hirat lvrat
##  0.00  0.00  0.02
```

No tiene sentido que las mensualidades de la hipoteca y gastos totales del hogar sean 0. Se asumen como valores missing. Se muestran las frecuencias de estos valores asumidos como missing.

```
datMod$pirat <- as.numeric( car::recode(datMod$pirat , "'0' = 'NA'")  )
```

```
## Warning: NAs introduced by coercion
```

```
datMod$hirat  <- as.numeric( car::recode(datMod$hirat  , "'0' = 'NA'")  )
```

```
## Warning: NAs introduced by coercion
```

```
sapply(Filter(is.numeric, datMod), function(x) min(x , na.rm = TRUE))
```

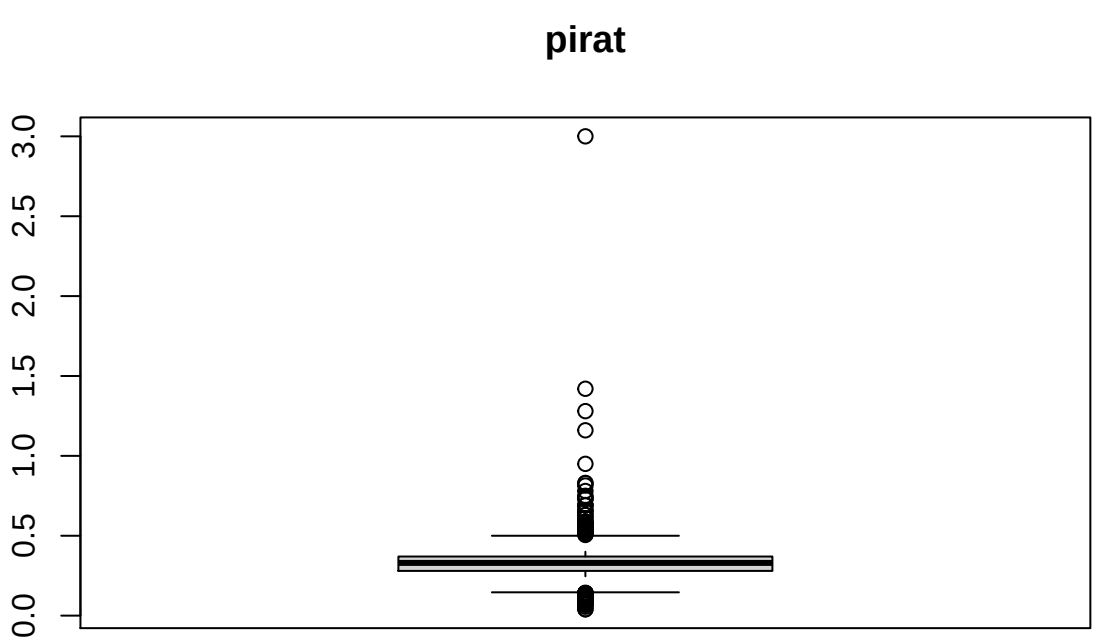
```
##   pirat   hirat   lvrat
## 0.04000 0.00085 0.02000
```

```
sum(is.na(datMod$hirat))
```

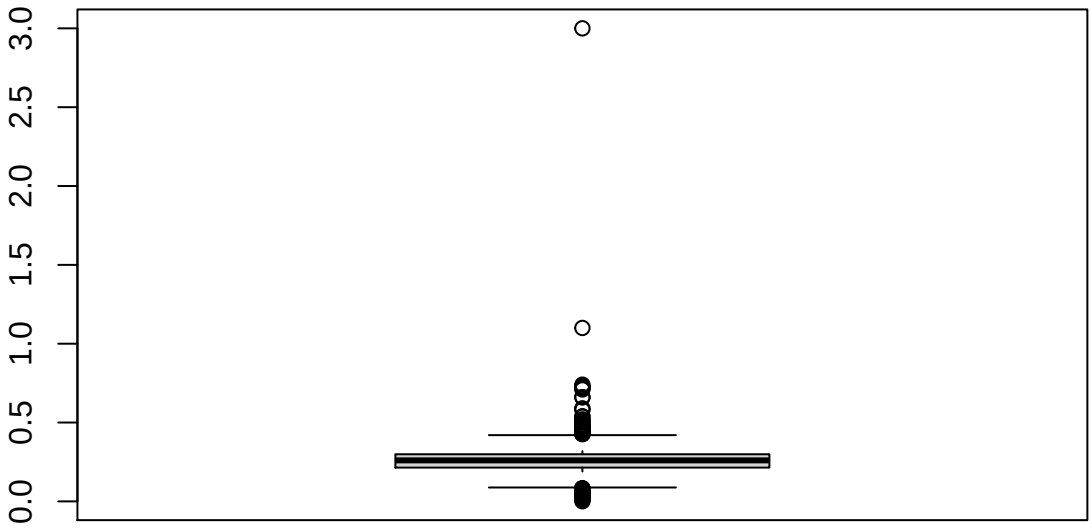
```
## [1] 3
```

Respecto al tratamiento de outliers, se usa el criterio IQR. Aquellos variables con porcentajes menores a 5% se considerá que hay outliers y no 'datos grandes'.

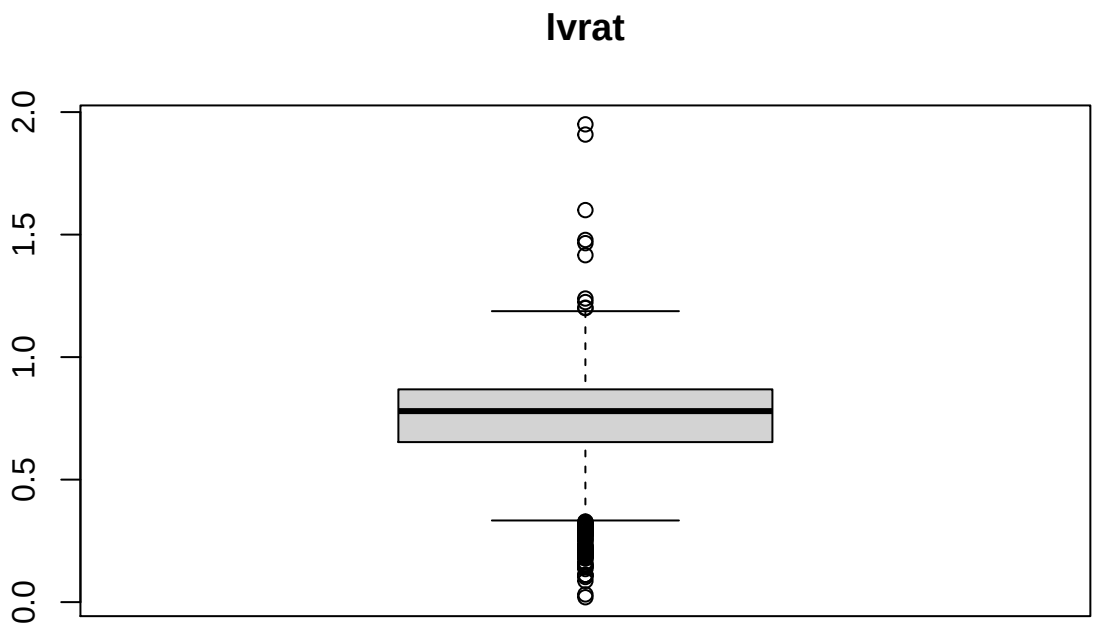
```
listconti <- c("pirat", "hirat", "lvrat")

for (cont in listconti)
{
 boxplot(datMod[,cont], main = cont)
 print(cont)
 bxplot <- boxplot.stats(datMod[,cont])
 print(100*length(bxplot$out)/nrow(datMod)  )

}
```




```
## [1] "pirat"
## [1] 3.996634
```




```
## [1] "hirat"
## [1] 3.744215
```

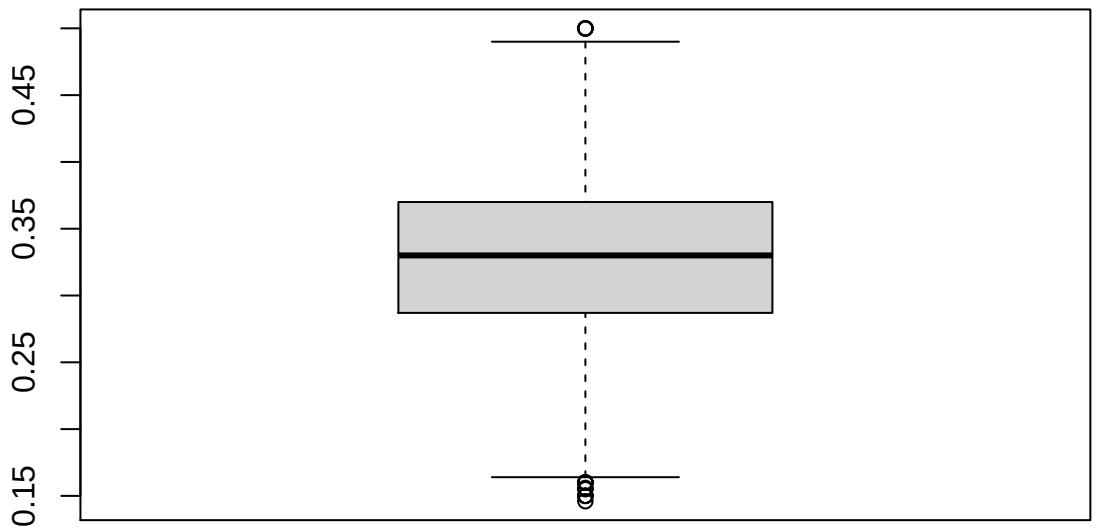



```
## [1] "lvrat"
## [1] 3.828355
```

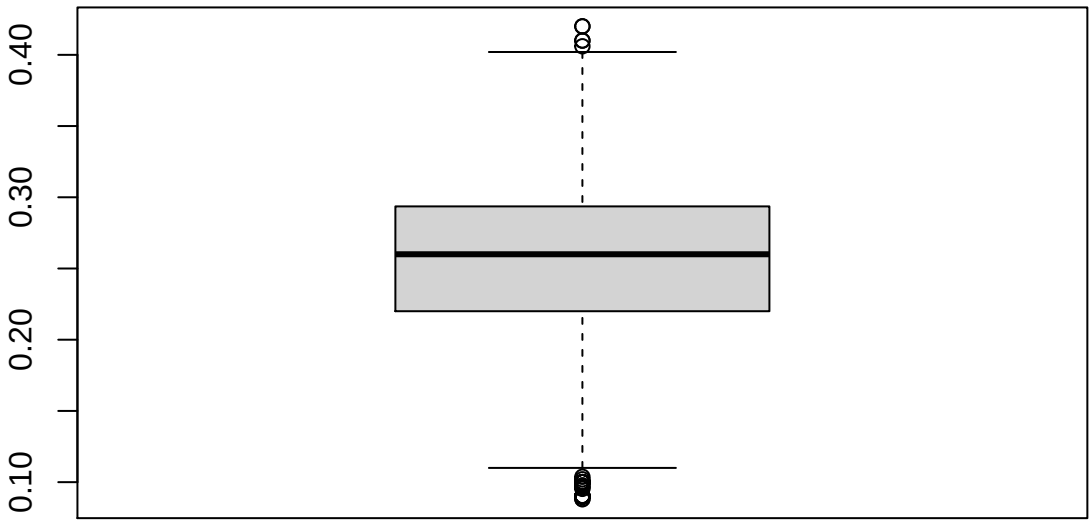
Por tanto, se convierten dichos outliers en missings y se comprueba el nuevo tamaño del dataset

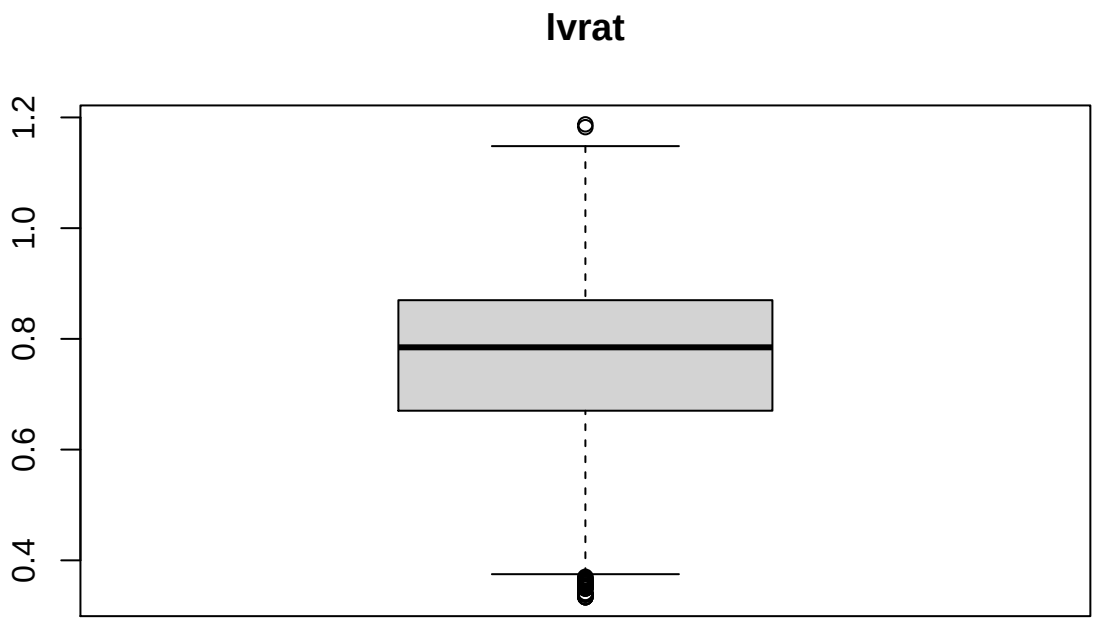
```
for (cont in listconti)
{

 bxplot <- boxplot.stats(datMod[,cont])

 datMod[,cont][datMod[,cont] %in% (bxplot$out )] <- 'NA'
 #datMod[,cont][datMod[,cont] > max(bxplot$out )] <- 'NA'

 datMod[,cont] <- as.numeric(datMod[,cont])

 boxplot(datMod[,cont], main = cont)


}
```

```
## Warning: NAs introduced by coercion

## Warning: NAs introduced by coercion
```




```
## Warning: NAs introduced by coercion
```

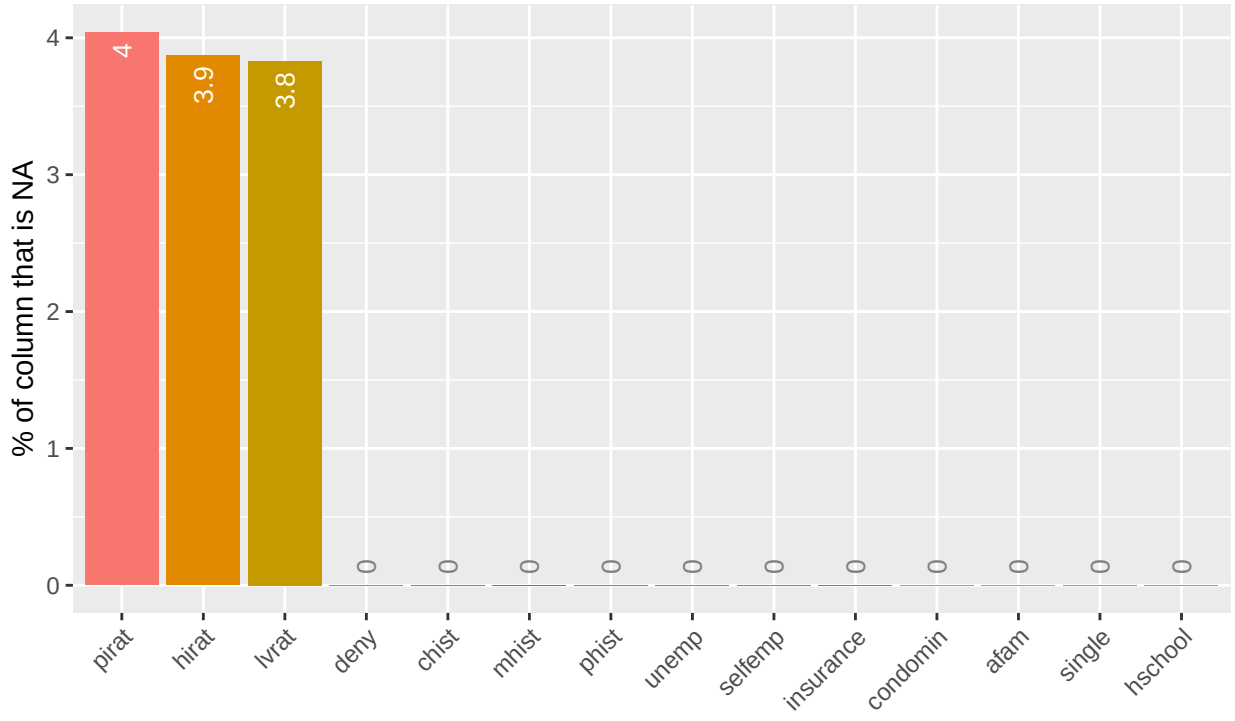

hirat
0.40
0.30
0.20
0.10



Se recalculan los porcentajes, si no superan el 5% se procede a eliminarlos.

```
# missingness barplot
x <- inspect_na(datMod)
show_plot(x)
```

```
## Warning: ‘guides(<scale> = FALSE)‘ is deprecated. Please use ‘guides(<scale> =
## "none")‘ instead.
```




```
for (cont in listconti)
{
 datMod<-subset(datMod, (!is.na(datMod[,cont]))  )
}


cat('Número de observaciones resultante: ', toString(nrow(datMod)), '\n')
```

```
## Número de observaciones resultante:  2159
```

```
cat('Porcentaje sobre el original, sin duplicados: ',toString(100 *nrow(datMod)/ nrow(datos) ), '\n')
```

```
## Porcentaje sobre el original, sin duplicados:  90.8287757677745
```

Se estandarizan variables numéricas. Esta práctica es recomendable para evitar problemas de overflow en algoritmos como redes neuronales o SVM. tras haber comprobado el rango parece aceptable

```
# Estandarizar las variables numéricas
listconti <- c("pirat", "hirat", "lvrat")
means <-apply(datMod[,listconti],2,mean,na.rm=TRUE)
sds<-sapply(datMod[,listconti],sd,na.rm=TRUE)

st.num<-scale(datMod[,listconti], center = means, scale = sds)

datMod<-data.frame(cbind(st.num, datMod[, -which(names(datMod) %in% listconti )] ))
```

Finalmente, se pasan a dummy las variables categóricas, guardamos datMod en datMod.nd por no dummies antes de modificarlo.

```
datMod.nd <- copy(datMod)
listclass <-colnames(Filter(is.character, datMod))

datMod<-dummy.data.frame(datMod, listclass, sep = ".")
```

2.5 Comentarios sobre la variable objetivo

Comprobamos que apenas han cambiado los porcentajes vistos anteriormente en deny. La clase dominante previamente presenta 1924 (89.1%) de las observaciones mientras que la minotaria "yes" tiene 235 observaciones, 10.9%, cumpliendo la condición "mayor de 100" impuesta en la guí de la tarea. De nuevo, la desproporción implica la necesidad de un stratifiedkfold para reducir el riesgo de overfitting.

Además, al ser un dataset pequeño de alrededor de 2159 observaciones, se harán menos folds que el estándar de 10, para reducir el riesgo de overfitting durante el entrenamiento. En este caso, tomando como referencia el ejemplo Ameshousing, se establecerá 4 folds para el remuestreo.

La accuracy base - que consistiría en asumir un modelo nulo con todos los outputs de la clase mayoritaria - es por tanto de 89.1%, siendo la tasa de fallos base de 10.9%.

```
freq(datMod$deny)
```

```
##        n    % val%
## no  1924 89.1 89.1
## yes  235 10.9 10.9
```

No hacemos featuring engineering

3 Modelo benchmark de regresión logística con selección de variables

La selección se hará mediante wrappers, en concreto stepwise. Otras formas de selección como filtros o embebidos se podrían evaluar en otras iteraciones.

Se evaluará el dataset completo tanto con stepwise como stepwise repetido, ambos con los dos criterios diferentes AIC y BIC. Se usa el dataset búsqueda de conjuntos de variables input como en validación cruzada al ser un dataset pequeño y, por tanto, una partición train-validation aumentaría el riesgo de underfitting. El riesgo de overfitting por no reservar una muestra para validación es reducido por la validación cruzada.

Se empiezan declarando el modelo nulo y con todas las variables. Se comprueba que el modelo nulo tiene la misma tasa de acierto que la base accuracy.

```
null<-glm(deny~1,data=datMod,family=binomial)
full <- glm(deny~.,data=datMod,family=binomial)

table(datMod$deny,predict(null)>0)# Punto de corte la predicción: ln(p/1-p) = 0 ó p = 0.5
```

```
##
##       FALSE
##   no   1924
##   yes   235
```

```
cat("La accuracy del modelo es: ", toString(1924/nrow(datMod)) )
```

```
## La accuracy del modelo es:  0.891153311718388
```

Vamos con no repetición. Se evalúa con todo el dataset, se introducen los criterios AIC y BIC para evitar que se escojan todas las variables.

```
set.seed(12345)
select1 <- stepAIC(null, scope=list(lower=null, upper=full), direction="both",trace=F)
set.seed(12345)
select2 <- stepAIC(null, scope=list(lower=null, upper=full),direction="both",k=log(nrow(datMod)), trace=
```

Ahora con repetición, no uso 'funcion steprepetido binaria.R' por tener las clases muy desbalanceadas que podrían dar lugar unas varianzas altas ya que puede ser que algun fold no tenga de la clase minoritaria. De nuevo uso los dos criterios. Pongo 0.75 en coherencia con kfolds = 4.

```
listconti <- colnames(datMod)[! colnames(datMod) %in% "deny"]

source("funcion steprepetido binaria_mb.R")

select3 <- steprepetidobinaria_mb(data=datMod,vardep="deny",
  listconti=listconti,
 sinicio=12345,sfinal=12346,porcen=0.75,criterio="AIC")

select4 <- steprepetidobinaria_mb(data=datMod,vardep="deny",
  listconti=listconti,
 sinicio=12345,sfinal=12346,porcen=0.75,criterio="BIC")
```

Defino las formulas de los stepwise anteriores, cojo las dos más frecuentes de cada repetida, si hay ya que es un dataset pequeño. No parece haber modelos repetidos.

```
rl.1 <- formula(select1)
rl.2 <- formula(select2)
rl.3 <- as.formula( paste('deny ~ ', select3[[1]][1]$modelo[1]))
rl.4 <- as.formula( paste('deny ~ ', select3[[1]][1]$modelo[2]))
rl.5 <- as.formula( paste('deny ~ ', select4[[1]][1]$modelo[1]))
rl.6 <- as.formula( paste('deny ~ ', select4[[1]][1]$modelo[2]))

modelos <- c(rl.1, rl.2, rl.3, rl.4, rl.5,rl.6)
print(modelos)
```

```
## [[1]]
## deny ~ insurance.no + chist.1 + phist.no + lvrat + afam.no +
##     pirat + chist.2 + selfemp.no + hschool.no + single.no + unemp.3.59999990463257 +
##     chist.6 + unemp.4.30000019073486 + unemp.10.6000003814697
##
## [[2]]
## deny ~ insurance.no + chist.1 + phist.no + lvrat + afam.no +
##     pirat
##
## [[3]]
```

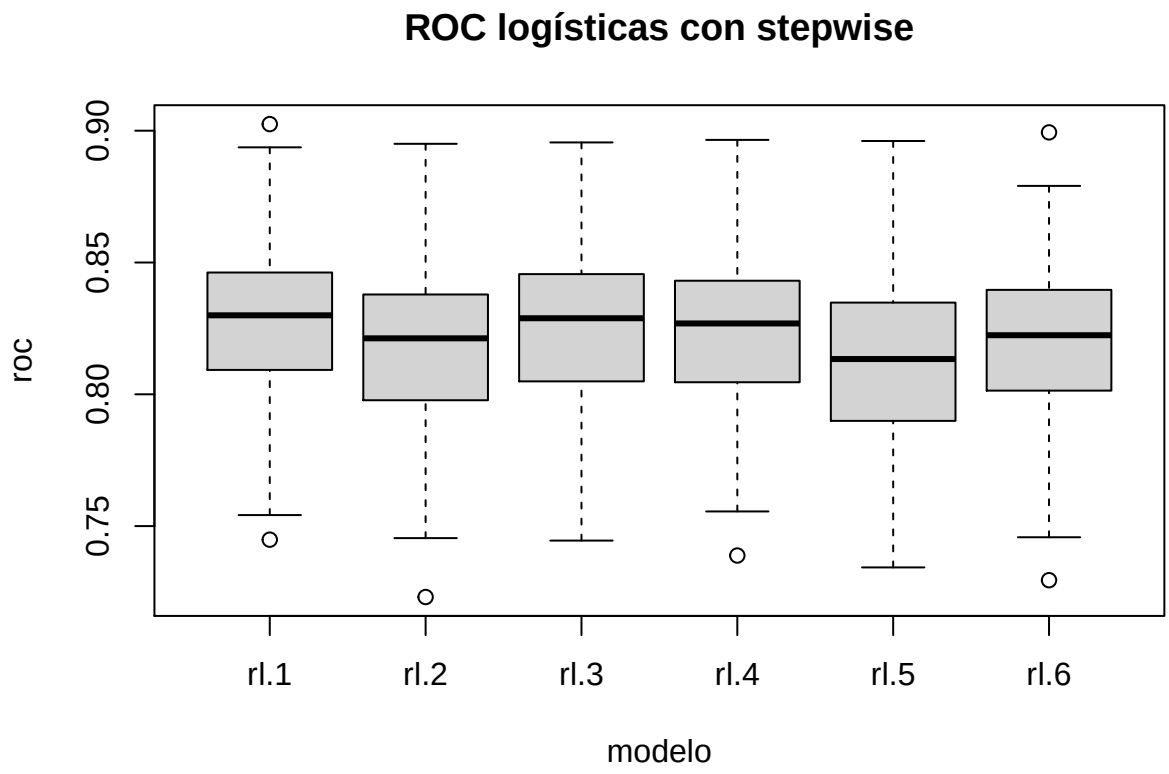
```
## deny ~ chist.1 + lvrat + afam.no + pirat + chist.2 + selfemp.no +
##     hschool.no + single.no + unemp.3.59999990463257 + insurance.yes +
##     phist.no + unemp.10.6000003814697
##
## [[4]]
## deny ~ insurance.no + phist.yes + chist.1 + lvrat + afam.no +
##     pirat + chist.2 + selfemp.no + hschool.no + single.no + unemp.3.59999990463257 +
##     chist.3
##
## [[5]]
## deny ~ chist.1 + lvrat + chist.2 + insurance.yes + phist.no
##
## [[6]]
## deny ~ insurance.no + phist.yes + chist.1 + lvrat + afam.no +
##     pirat + chist.2
```

Se hace validación cruzada repetida, uso de una semilla distinta para reducir la posibilidad de sesgo. Uso todo el datMod sin apartar un validation set al ser pequeño.

```
total<-c()
n.folds <- 4



for (i in 1:length(modelos)){
set.seed(1234)

cvIndex <- createMultiFolds(factor(datMod$deny), k = n.folds, times = 20)

#createFolds(factor(datMod$deny), n.folds, returnTrain = T)
# index = cvIndex,

trainControl <- trainControl(index = cvIndex,
                             method = "cv",
                             #number = n.folds,
                             #repeats = 20,
                             classProbs = TRUE,
                             summaryFunction=twoClassSummary,
                             returnResamp="all"
                            )


vcr<-train(modelos[[i]], data = datMod,
method = "glm", family="binomial", metric = "ROC",
trControl = trainControl
)

total<-rbind(total,data.frame(roc=vcr$resample[,1],
modelo=rep(paste("rl.", ifelse(i<10,paste0(i),i), sep=""),
nrow(vcr$resample))))

}
```

Se muestran los datos obtenidos en forma de diagrama de cajas y tabular.

```
boxplot(roc~modelo,data=total,main="ROC logísticas con stepwise")
```




```
n.var <- c()



for (modelo in modelos)

{
  n.var <- append(n.var, length(attr(terms(modelo), "term.labels")))
}



results <- cbind(aggregate(total[, 1], list(total$modelo), mean),
                 aggregate(total[, 1], list(total$modelo), sd))

results[,3] <- NULL

results <- cbind(results,n.var)

colnames(results) <- c('modelo','mean', 'sd', '# variables')

results
```

```
##   modelo      mean         sd # variables
## 1   rl.1 0.8280904 0.03124426          14
## 2   rl.2 0.8177701 0.03254752           6
## 3   rl.3 0.8260656 0.03150629          12
## 4   rl.4 0.8247110 0.03190663          12
## 5   rl.5 0.8117793 0.03336646           5
## 6   rl.6 0.8189515 0.03204852           7
```

Los 6 modelos evaluados dan lugar a una reducción de predictores de 35 a menos de 14, llegando incluso a 5 variables input. Los ROC tanto en media como desviación son parecidos. Sin embargo hay que recordar que la accuracy base es de un 89.1%. Por tanto, en este datasetm, pequeñas variaciones del ROC pueden dar lugar a significativas ganancias respecto a la accuracy base.

Además, hay que señalar que estos resultados NO contemplan el comportamiento frente a nuevas observaciones, son simplemente un indicio de qué resultados pueden ser. Es por tanto necesario emplear un esquema de remuestreo para estar seguros del sesgo y varianza de estos dos modelos tentativos más prometedores.

A continuación mostramos las accuracies de rl.1 y rl.5, los conjuntos con más y menos variables y con más y menos ROC medio, respectivamente

```
n.folds <- 4

set.seed(1234)

cvIndex <- createMultiFolds(factor(datMod$deny), k = n.folds, times = 20)

trainControl <- trainControl(index = cvIndex,
                             method = "cv",
                             classProbs = TRUE,
                             summaryFunction=twoClassSummary,
                             returnResamp="all"
                            )


fit.rl.1<-train(rl.1, data = datMod,
method = "glm", family="binomial", metric = "ROC",
trControl = trainControl
)




set.seed(1234)

cvIndex <- createMultiFolds(factor(datMod$deny), k = n.folds, times = 20)

trainControl <- trainControl(index = cvIndex,
                             method = "cv",
                             classProbs = TRUE,
                             summaryFunction=twoClassSummary,
                             returnResamp="all"
                            )

fit.rl.5<-train(rl.5, data = datMod,
method = "glm", family="binomial", metric = "ROC",
```

```
trControl = trainControl
)

predict.rl.1 <- predict(fit.rl.1, newdata = datMod)
confusionMatrix.rl.1 <- confusionMatrix( datMod$deny, predict.rl.1)

predict.rl.5 <- predict(fit.rl.5, newdata = datMod)
confusionMatrix.rl.5 <- confusionMatrix( datMod$deny, predict.rl.5)

cat('Accuracy de rl.1: ', toString(confusionMatrix.rl.1$overall[1]),'\n')
```

```
## Accuracy de rl.1:  0.914775358962483
```

```
cat('Accuracy de rl.5: ', toString(confusionMatrix.rl.5$overall[1]),'\n')
```

```
## Accuracy de rl.5:  0.911533117183881
```

Se observa accuracies muy similares, ambas por encima de 0.891. Se asumirá que la ganancia de apenas 0.3% no compensa y, por tanto, se escogerá el conjunto de variables input rl.5 como referencia al tener menos predictores. Además se tomará como estándar a comparar la regresión logística para rl.5 al ser este un modelo modelo simple y descriptivo respecto al significado de los coeficientes.

El tener menos predictores presenta varias ventajas. En primer lugar, se necesita menos tiempo de computación aunque esto no es tan relevante en datasets pequeños. En segundo lugar, siempre que los modelos tengacapacidades predictivas similares (tanto en media como en variabilidad) para no incurrir en underfitting, reducir el número de predictores supone una protección frente a sobreajuste de cara a su despliegue en aplicaciones reales. Finalmente, en casos como este dataset, menos predictores pueden propiciar a centrarse más en unas pocas variables fundamentales, el uso de "pueden" es porque otros modelos con más variables pueden ser dummies de una original.

De todos modos, si las limitaciones de tiempo/capacidad de computación lo permiten, otros conjuntos de variables convendría evaluar ya que un algoritmo puede dar mejores/peores resultados en función del conjunto de variables input. Del mismo modo, además de regresión logística con stepwise, otros tipos de selección como embedded o árboles también conviene evaluar.

Se muestra un resumen del modelo escogido como referencia. Se observa que el ROC medio y su varianza son similares a los valores obtenidos en stepwise repetidos. Por lo que, en este caso, los wrappers stepwise (con o sin repetición) han resultado ser un buen indicativo del sesgo y varianza de los modelos.

```
cat('ROC de rl.5: ', toString(fit.rl.5$results$ROC),'\n')
```

```
## ROC de rl.5:  0.811779316997391
```

```
cat('ROC SD de rl.5: ', toString(fit.rl.5$results$ROCSD),'\n')
```

```
## ROC SD de rl.5:  0.0333664597122185
```

```
cat('Accuracy de rl.5: ', toString(confusionMatrix.rl.5$overall[1]),'\n')
```

```
## Accuracy de rl.5:  0.911533117183881
```

```
cat('Número de variables rl.5: ', toString(length(attr(terms(rl.5), "term.labels"))),'\n')
```

```
## Número de variables rl.5:  5
```

```
formula(rl.5)
```

```
## deny ~ chist.1 + lvrat + chist.2 + insurance.yes + phist.no
```

4 Tuneado de algoritmos

Un primer grid search no fino de los hiperparámetros con cv sin repetición para agilizar la computación. Tras ello, se realiza con el modelo más prometedor una cv con repetición para intentar reducir el error y, por ende, mejorar la varianza como se vió en la justificación teórica de los ensamblados.

Los algoritmos de la guía de la tarea, así como discusión de los parámetros más importantes y su efecto.

Un grid search fino alrededor del modelo ganador en cada algoritmo, a modo de optimización local, en una siguiente iteración.

De nuevo, se recuerda que el dataset cuenta con pocas observaciones con solo 5 predictores a evaluar y, por tanto, propenso a overfitting.

4.1 Redes

Los párametros más importantes son el número de nodos, learning rate y el número de iteraciones.

- Número de nodos. Es el parámetro más relevante. El rango de valores a evaluar en el grid search dependerá del tamaño del número de observaciones o de la complejidad de las variables input. Si el número de observaciones es pequeño y la dependencia de la variable objetivo con las input es lineal, el rango deberá ser menor para evitar overfitting. En cambio si hay muchas observaciones y el problema presenta muchas variables categóricas o relacones no lineales entre variables numéricas habrá que aumentar el rango de valors para evitar underfitting.
- Learning rate. Controla la velocidad del cambio de pesos en cada iteración. Un valor muy pequeño podría llegar a converger o converger a un mínimo local. En cambio, un valor alto daría lugar a oscilaciones alrededor del óptimo ocasionando problemas de inestabilidad.
- maxit. pocas iteraciones puede dar lugar a no convergencia o a underfitting. Demasiadas iteraciones podría ocasionar sobreajuste.

A todo esto hay que añadir la varianza presentada por la iniciación aleatoria de los pesos de la red. Para intentar reducirlo, se recurrirá a un ensamblado llamado "avNNet". avNNet consiste en inicializar varias veces una misma red. Se fijará un valor de 5, en consonancia con los códigos de clase.

Tras estas consideraciones generales se discute a continuación los valores a evaluar en el grid search.

Número de nodos. Siguiendo las reglas generales como referencia inicial y teniendo en cuenta que es un dataset pequeño y con pocas variables, se evalúa el número de parámetros para la ratio 20 obs./parámetro. La fórmula para calcular el número de nodos ocultos (h) es h(k+1)+h+1 donde k es el número de variables input en este caso, cuyo valor es 7 para el modelo rl.5, obtenido del apartado anterior. Considerando que nrow(datMod) es 2159, esta referencia inicial daría aproximadamente 15 nodos.

Teniendo en cuenta esta referencia inicial y limitaciones en cuanto a capacidad de computación, el mallado será c(30,25,20,15,10). Con h = 30 solo hay 10 observaciones por nodo, por lo que se considerá que el

overfitting puede ser significativo y, por tanto, carece de sentido seguir aumentando el valor de h. Con h = 10 se obtiene 30 observaciones por nodo lo cual, teniendo en cuenta que se trata de un dataset pequeño , podría incrementarse el riesgo de underfitting.

Learning rate. Se recurre a valores tipo usados en la literatura c(0.01,0.1,0.001).

maxit. Se usarán los valores c(100, 300, 500), al ser 100 y 500 valores referenciados en los scripts. Aunque se mencionó en las clases que 100 es un número bajo, en este problema propenso a overfitting por el pequeño tamaño del dataset podría dar mejores resultados que usar valores mayores.

Se realiza el grid search con los valores escogidos con cv stratified sin repetición, por motivos computacionales.

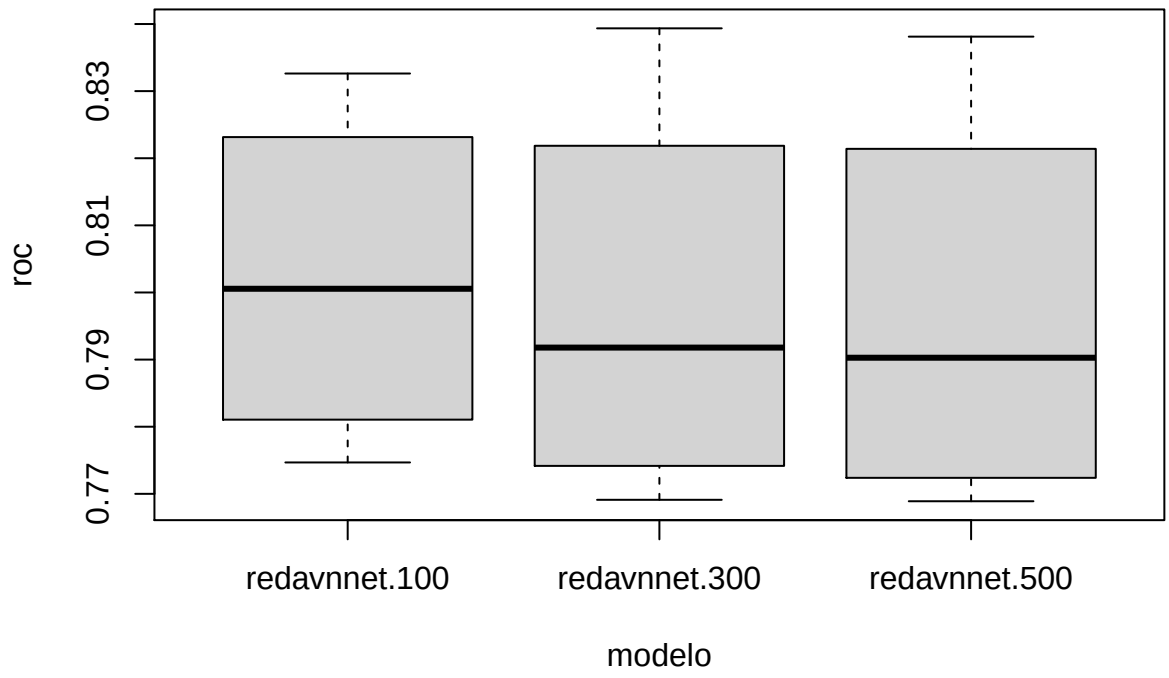
```
total.avvnet <- c()

tic()

avnnet.repeats <- 5


set.seed(1234)

cvIndex <- createFolds(factor(datMod$deny), n.folds, returnTrain = T)


trainControl <- trainControl(index = cvIndex,
                             method = "cv",
                             number = n.folds,
                             classProbs = TRUE,
                             summaryFunction=twoClassSummary
                            )



avnnetgrid <-expand.grid(size=c(30,25,20,15,10),
                         decay=c(0.01,0.1,0.001),bag=FALSE)


redavnnet.100<- train(rl.5,
                  data=datMod,method="avNNet",linout = TRUE,maxit=100,
                  trControl=trainControl,tuneGrid=avnnetgrid,
                  repeats=avnnet.repeats,
                  metric = "ROC")

redavnnet.300<- train(rl.5,
                  data=datMod,method="avNNet",linout = TRUE,maxit=300,
                  trControl=trainControl,tuneGrid=avnnetgrid,
                  repeats=avnnet.repeats,
                  metric = "ROC")

redavnnet.500<- train(rl.5,
                  data=datMod,method="avNNet",linout = TRUE,maxit=500,
                  trControl=trainControl,tuneGrid=avnnetgrid,
                  repeats=avnnet.repeats,
                  metric = "ROC")
```

```
toc()

i <- 100
total.avvnet<-rbind(total.avvnet,data.frame(roc=redavnnet.100$resample[,1],
modelo=rep(paste("redavnnet.", ifelse(i<10,paste0(i),i), sep=""),
nrow(redavnnet.100$resample))))

i <- 300
total.avvnet<-rbind(total.avvnet,data.frame(roc=redavnnet.300$resample[,1],
modelo=rep(paste("redavnnet.", ifelse(i<10,paste0(i),i), sep=""),
nrow(redavnnet.300$resample))))

i <- 500
total.avvnet<-rbind(total.avvnet,data.frame(roc=redavnnet.500$resample[,1],
modelo=rep(paste("redavnnet.", ifelse(i<10,paste0(i),i), sep=""),
nrow(redavnnet.500$resample))))
```

Se analizan todos los modelos con las tablas de resultados para cada configuración y por un boxplot con el modelo propuesto por caret. Es decir, no se escoge automáticamente el mejor ajuste dado por el train, ya que puede ser algo mejor en términos de ROC pero tener una varianza mayor que otros modelos.

Del boxplot se intuye que la configuración maxit = 100 puede ser la mejor al presentar menor sesgo y varianza. Esto se confirma en las tablas, donde las dos mejores configuraciones de size y decay se producen con maxit = 100. En este caso, el mejor modelo en maxit= 100 es el propuesto propuesto por caret, al tener menor sesgo y varianza.

```
boxplot(roc~modelo,data=total.avvnet,main="AUC avvnet maxit = 100, 300, 500")
```




```
redavnnet.100$results[,c('size','decay','ROC','ROCSD')]
```

```
##    size decay       ROC      ROCSD
## 1    10 0.001 0.7921940 0.03281387
## 2    10 0.010 0.7900879 0.02887181
## 3    10 0.100 0.7986384 0.02992255
## 4    15 0.001 0.7893273 0.03756130
## 5    15 0.010 0.7939052 0.02869318
## 6    15 0.100 0.7977724 0.02989147
## 7    20 0.001 0.7970728 0.03423949
## 8    20 0.010 0.7992085 0.02840543
## 9    20 0.100 0.7976378 0.03214312
## 10   25 0.001 0.7932957 0.03749113
## 11   25 0.010 0.7947588 0.03059722
## 12   25 0.100 0.8021017 0.02603212
## 13   30 0.001 0.8012278 0.02789150
## 14   30 0.010 0.7978776 0.02568729
## 15   30 0.100 0.7983145 0.03366352
```

```
redavnnet.300$results[,c('size','decay','ROC','ROCSD')]
```

```
##    size decay       ROC      ROCSD
## 1    10 0.001 0.7892340 0.02184096
## 2    10 0.010 0.7872360 0.03415864
```

```
## 3    10 0.100 0.7962601 0.03061333
## 4    15 0.001 0.7814220 0.02868591
## 5    15 0.010 0.7806548 0.02962364
## 6    15 0.100 0.7959167 0.03238787
## 7    20 0.001 0.7789306 0.02728832
## 8    20 0.010 0.7889843 0.02897644
## 9    20 0.100 0.7969347 0.03071935
## 10   25 0.001 0.7687464 0.02765873
## 11   25 0.010 0.7835317 0.03276131
## 12   25 0.100 0.7980099 0.03129406
## 13   30 0.001 0.7733154 0.02874174
## 14   30 0.010 0.7828438 0.02981412
## 15   30 0.100 0.7973608 0.03293537
```

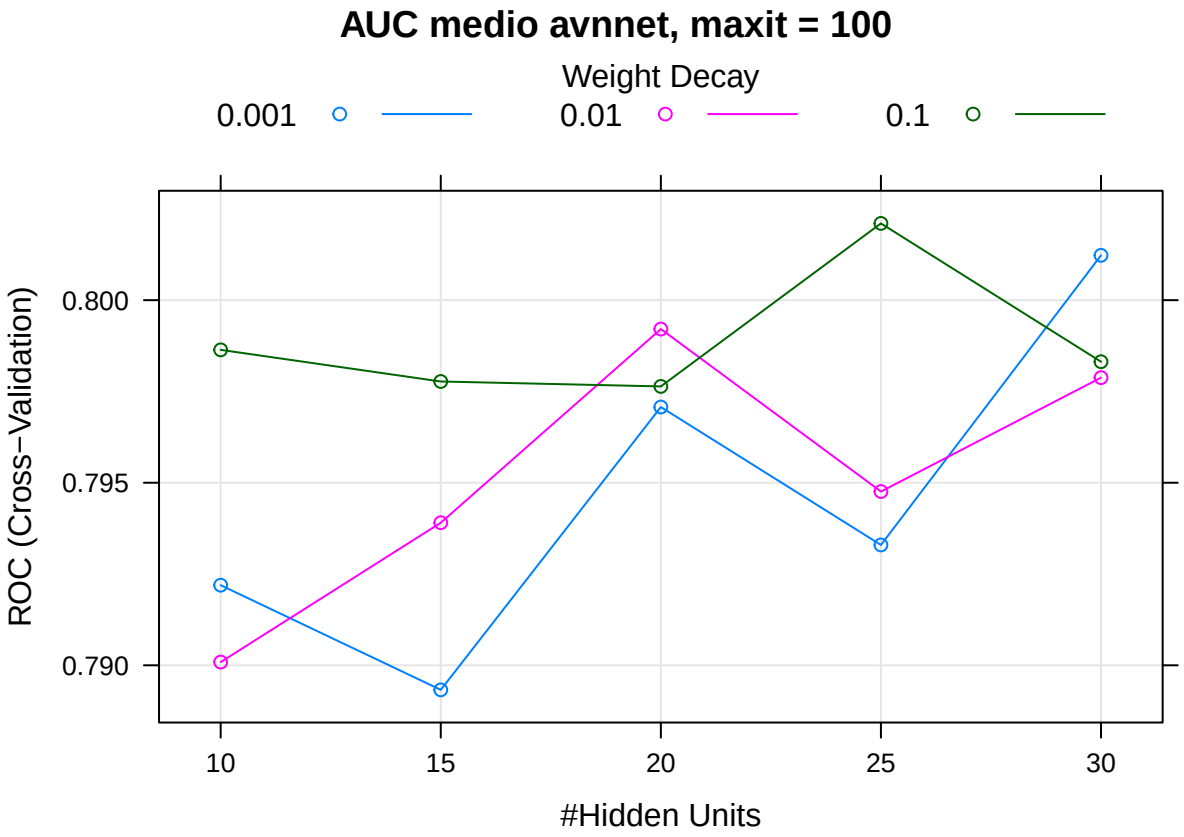
```
redavnnet.500$results[,c('size','decay','ROC','ROCSD')]
```

```
##    size decay       ROC      ROCSD
## 1    10 0.001 0.7794660 0.02317944
## 2    10 0.010 0.7839000 0.02678202
## 3    10 0.100 0.7960237 0.03036708
## 4    15 0.001 0.7677687 0.03224752
## 5    15 0.010 0.7765073 0.02294747
## 6    15 0.100 0.7968102 0.03130652
## 7    20 0.001 0.7473949 0.02779625
## 8    20 0.010 0.7783246 0.03204766
## 9    20 0.100 0.7966443 0.03161999
## 10   25 0.001 0.7448989 0.03642502
## 11   25 0.010 0.7727323 0.02655219
## 12   25 0.100 0.7962917 0.03228614
## 13   30 0.001 0.7518598 0.03031074
## 14   30 0.010 0.7763346 0.02594042
## 15   30 0.100 0.7968925 0.03154876
```

De los tres mallados se escoge los a priori dos mejores ajustes obtenido para maxit = 100 para cv repetida. Los dos mejores modelos son las parejas de (size, decay) (25, 0.1) y (30, 0.001) - como se puede apreciar en las tablas.

Se aprecia que el AUC tiende a subir con size y con el decay.

```
plot(redavnnet.100, main = 'AUC medio avnnet, maxit = 100')
```



Con los dos moelos escogidos (con el mallado realizado) se realiza una validación cruzada con repetición para intentar mejorar el roc y reducir la varianza respecto al cv simple. Escogemos los dos modelos tentativos más prometedores.

Genero boxplots

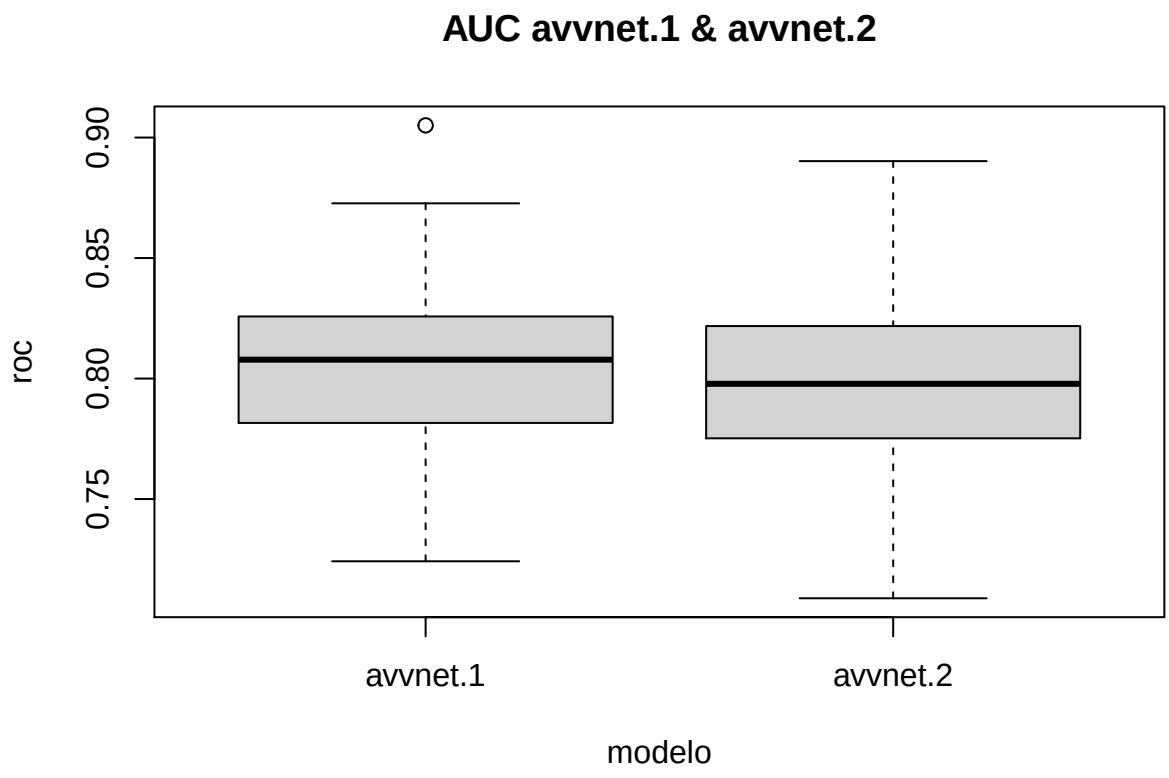
```
set.seed(1234)

cvIndex <- createMultiFolds(factor(datMod$deny), k = n.folds, times = 20)

#createFolds(factor(datMod$deny), n.folds, returnTrain = T)
# index = cvIndex,

trainControl <- trainControl(index = cvIndex,
                             method = "cv",
                             #number = n.folds,
                             #repeats = 20,
                             classProbs = TRUE,
                             summaryFunction=twoClassSummary,
                             returnResamp="all"
                            )


avnnetgrid <-expand.grid(size=c( 25 ),
                         decay=c( 0.1),bag=FALSE)
```

```
avvnet.1<- train(rl.5,
                 data=datMod,method="avNNet",linout = TRUE,maxit=100,
                 trControl=trainControl,tuneGrid=avnnetgrid,
                 repeats=avnnet.repeats,
                 metric = "ROC")



set.seed(1234)

cvIndex <- createMultiFolds(factor(datMod$deny), k = n.folds, times = 20)

#createFolds(factor(datMod$deny), n.folds, returnTrain = T)
# index = cvIndex,

trainControl <- trainControl(index = cvIndex,
                             method = "cv",
                             #number = n.folds,
                             #repeats = 20,
                             classProbs = TRUE,
                             summaryFunction=twoClassSummary,
                             returnResamp="all"
                            )


avnnetgrid <-expand.grid(size=c( 30 ),
                         decay=c( 0.001),bag=FALSE)


avvnet.2<- train(rl.5,
                 data=datMod,method="avNNet",linout = TRUE,maxit=100,
                 trControl=trainControl,tuneGrid=avnnetgrid,
                 repeats=avnnet.repeats,
                 metric = "ROC")
```

```
i <-1
total.avvnet <- data.frame(roc=avvnet.1$resample[,1],
modelo=rep(paste("avvnet.", ifelse(i<10,paste0(i),i), sep=""),
nrow(vcr$resample)))

i <-2
total.avvnet <-rbind(total.avvnet, data.frame(roc=avvnet.2$resample[,1],
modelo=rep(paste("avvnet.", ifelse(i<10,paste0(i),i), sep=""),
nrow(vcr$resample))))


results.avvnet <- c("avvnet.1",mean(avvnet.1$resample[,'ROC']), sd(avvnet.1$resample[,'ROC']))
results.avvnet <- transpose(as.data.frame(results.avvnet))

colnames(results.avvnet) <- c("model", "mean", "sd")
results.avvnet <- rbind(results.avvnet, c("avvnet.2",mean(avvnet.2$resample[,'ROC']), sd(avvnet.2$resamp

results.avvnet
```

```
##      model              mean                 sd
## 1 avvnet.1 0.802907652544195  0.035108557464155
## 2 avvnet.2 0.797550151672376 0.0359498178866205
```

```
boxplot(roc~modelo,data=total.avvnet,main="AUC avvnet.1 & avvnet.2")
```



El modelo ganador para este algoritmo es avvnet.1 al tener menor sesgo y menor varianza que avvnet.2.

4.2 Bagging y Random Forest

Se analizan los dos algoritmos en una misma sección al ser bagging un caso particular de random forest donde el mtry es igual al número de variables input. Ambos algoritmos son ensamblados de árboles usados para bajar la varianza y por tanto el error de los modelos. Además el mtry de random forest permite crear árboles muy diferentes, aumentando la probabilidad de captar más sutilezas del dataset, reduciendo el riesgo de sobreajuste.

Los hiperparámetros más importantes son en caret:

mtry. El número de variables input a evaluar para la partición en cada nodo. Un mayor mtry tiende a reducir el sesgo, permitiendo captar más sutilezas del train, aumentando también el riesgo de overfitting. Un bajo mtry tiende a reducir la varianza, si es muy bajo el modelo podría no estar captando suficientes sutilezas del train e incurrir en underfitting.

nodesize. Tamaño mínimo de nodos finales, un tamaño mayor tiende a reducir la varianza mientras que un tamaño menor propicia reducir el sesgo.

ntree. Número de árboles, a partir de un cierto número el modelo "converge", es decir, la inclusión de más árboles ni mejora ni empeora los modelos. Conviene hacer un estudio previo antes de las técnicas de

remuestreo sobre el número de árboles para ahorrar tiempo para evitar malos resultados de forma sistemática (demasiado pocos) o cálculos innecesarios. En ambos casos se reduce el tiempo computacional.

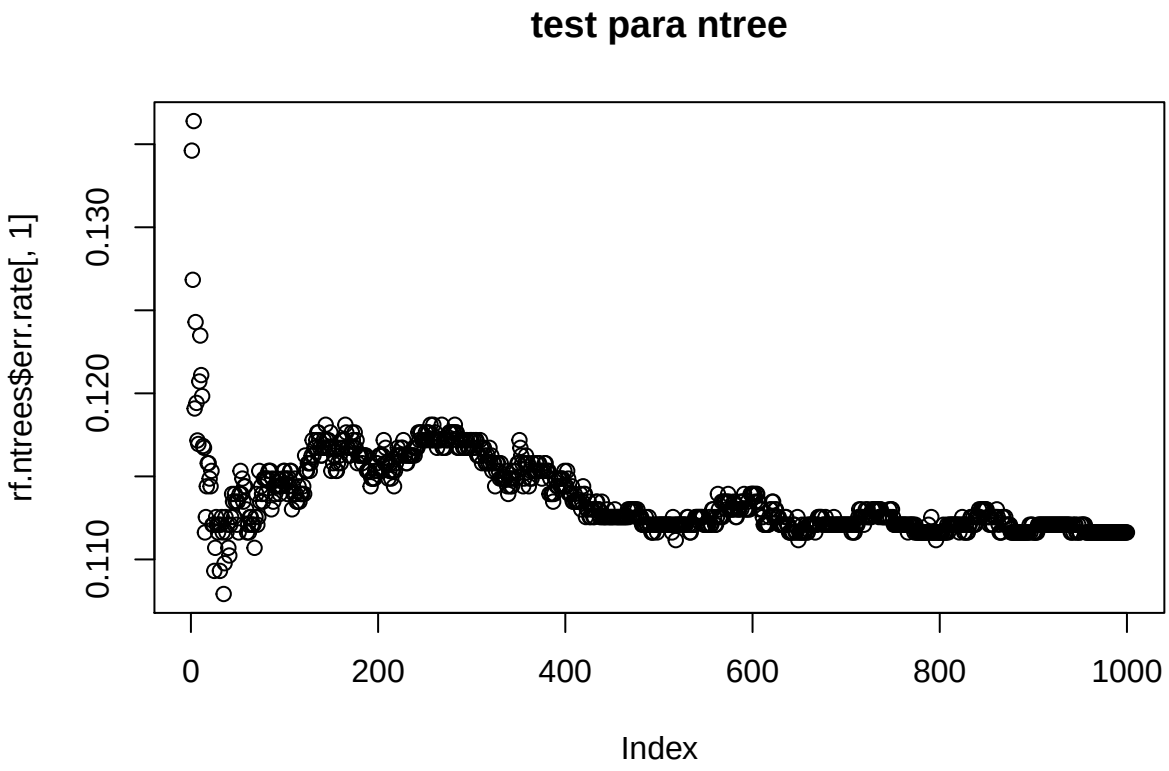
sampsize. Número de observaciones a evaluar. Aumentar el tamaño muestra reduce el sesgo mientas que disminuirlo hace a los modelos más robustos al reducir la varianza. Esto se puede hacer con o sin reemplazo. En teoría convendría evaluar tanto con como sin reemplazo, de cara a obtener un pool más grande de modelos candidatos. Por motivos de tiempo solo se utilizará la opción clásica de estos algoritmos, con reemplazamiento.

En primer lugar, se procede a fijar el valor del hiperparámetro ntree.

4.2.1 minimo numero de ntree

Para ver el número de árboles a seleccionar se recurre a la técnica OOB. El ntree a escoger será el mínimo valor- para así ahorrar tiempo computacional - en el cual se estabiliza el valor de OOB, que viene a cumplir funciones de test set. Para ello se establece un random forest con parámetros que propicien el sobreajuste, es decir, que simulen un caso adverso de alta varianza que cueste en converger el OOB.

```
set.seed(1234)

rf.ntrees <-randomForest(rl.5, data=datMod, mtry=5,ntree=1000,sampsize=nrow(datMod),nodesize=10,replace=

plot(rf.ntrees$err.rate[,1], main = 'test para ntree')
```



A la vista del gráfico, se asume que la convergencia se produce con ntree = 500. Con el valor de ntree seleccionado, se procede al grid search con cv sin repetición.**cv y no repeated en aras a la generalizacion** En primer lugar se presentan los valores de los hiperparámetros a estudiar.

```
ntree <- 500
mtry <- c(3,4,5) # 5 para simular bagging

# hiperparámetros para el bucle
n.min.class <- freq(datMod$deny)[2,'n']
sampsizes <- c(as.integer(0.6*n.min.class), as.integer(0.8*n.min.class), as.integer(0.9*n.min.class))
nodesizes <- c(20, 30, 40)

#iris = read.table("http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data", sep = ","
#names(iris) = c("sepal.length", "sepal.width", "petal.length", "petal.width", "iris.type")
```

Además del mtry se evalúan distintos valores de sampsize y nodesize atendiendo a las referencias aportadas en clase, las muestras se toman de forma estratificada por el desequilibrio de clases en la variable objetivo.

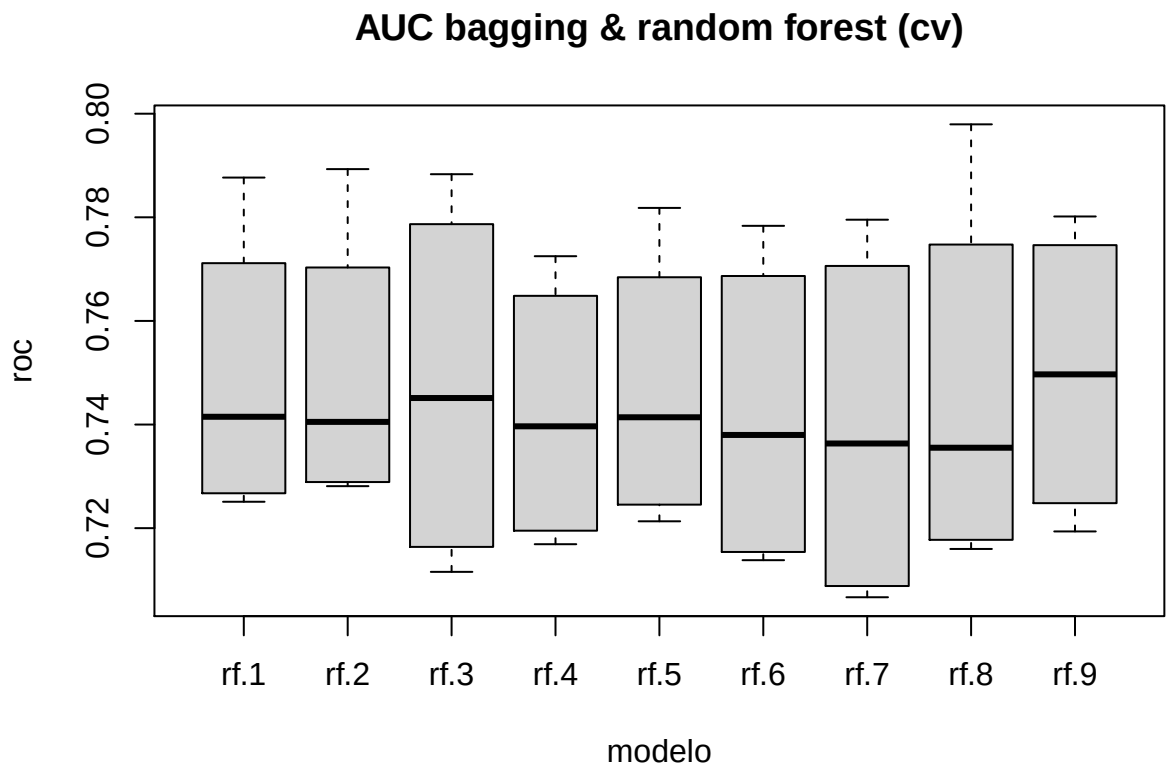
```
total.rf <- c()
i <- 1

for (sampsize in sampsizes)

{
  for (nodesize in nodesizes)

  {

    trainControl <- trainControl(method="cv",
                         summaryFunction=twoClassSummary,
                         number = n.folds,
                         classProbs=T,
                         savePredictions = "all")

    rf.grid <- expand.grid(mtry=mtry)


    set.seed(1234)

    rfFit <- train(rl.5, data=datMod,
                   method="rf", ntree= 500, nodesize = nodesize, tuneGrid = rf.grid,
                   trControl=trainControl, metric="ROC",
                   sampsize=c(9*sampsize, sampsize), strata=datMod$deny, replace = TRUE)


    print(rfFit$results[,c('mtry','ROC','ROCSD')])

    total.rf<-rbind(total.rf,data.frame(roc=rfFit$resample[,1],
    modelo=rep(paste("rf.", ifelse(i<10,paste0(i),i), sep=""),
    nrow(rfFit$resample))))

    i <- i + 1

  }

}
```

```
##   mtry       ROC      ROCSD
```

```
## 1    3 0.7378842 0.04095023
## 2    4 0.7489417 0.02900628
## 3    5 0.7417196 0.03335039
##   mtry       ROC      ROCSD
## 1    3 0.7495932 0.02501479
## 2    4 0.7496094 0.02850434
## 3    5 0.7399258 0.02455177
##   mtry       ROC      ROCSD
## 1    3 0.7466318 0.03324167
## 2    4 0.7475306 0.03702639
## 3    5 0.7474843 0.02799880
##   mtry       ROC      ROCSD
## 1    3 0.7421123 0.04105746
## 2    4 0.7421779 0.02699227
## 3    5 0.7404170 0.03041399
##   mtry       ROC      ROCSD
## 1    3 0.7388289 0.03773130
## 2    4 0.7464855 0.02771938
## 3    5 0.7422055 0.02844389
##   mtry       ROC      ROCSD
## 1    3 0.7420398 0.03177996
## 2    4 0.7380078 0.03695089
## 3    5 0.7390300 0.03179832
##   mtry       ROC      ROCSD
## 1    3 0.7365506 0.03965895
## 2    4 0.7397235 0.03644763
## 3    5 0.7381576 0.03082006
##   mtry       ROC      ROCSD
## 1    3 0.7462456 0.03799533
## 2    4 0.7446214 0.02621811
## 3    5 0.7379158 0.02224716
##   mtry       ROC      ROCSD
## 1    3 0.7295838 0.02722816
## 2    4 0.7497248 0.02944595
## 3    5 0.7448090 0.03344490
```

Las tablas muestran poca variación en el AUC medio y su SD, por lo que se puede considerar las recomendadas en caret como óptimas para en la mayoría de los casos. A continuación se muestra un boxplot de la cv.

```
boxplot(roc~modelo,data=total.rf,main="AUC bagging & random forest (cv)")
```



Se escogen rf.2, rf.3 y rf.9 para validación cruzada repetida. rf.3 tiene más varianza pero menos sesgo que los otros modelos descartados.rf.2 tiene sesgo y varianza similar a estos pero con valores mínimos más altos. rf.9 tiene el que menos sesgo de todos. Las configuraciones de caret, viendo las tablas anteriores, se pueden tomar como el óptimo dada la poca variación. El mtry en ambos se fijará en 4 de cara a la validación repetida.

```
trainControl <- trainControl(method="repeatedcv",
                      summaryFunction=twoClassSummary,
                      number = n.folds,
                      classProbs=T,
                      savePredictions = "all",
                      repeats=20)

rf.grid <- expand.grid(mtry=4)


set.seed(1234)

rf.2 <- train(rl.5, data=datMod,
               method="rf", ntree= 500, nodesize = nodesizes[2], tuneGrid = rf.grid,
               trControl=trainControl, metric="ROC",
               sampsize=c(9*sampsizes[1], sampsizes[1]), strata=datMod$deny, replace = TRUE)


set.seed(1234)

rf.3 <- train(rl.5, data=datMod,
```

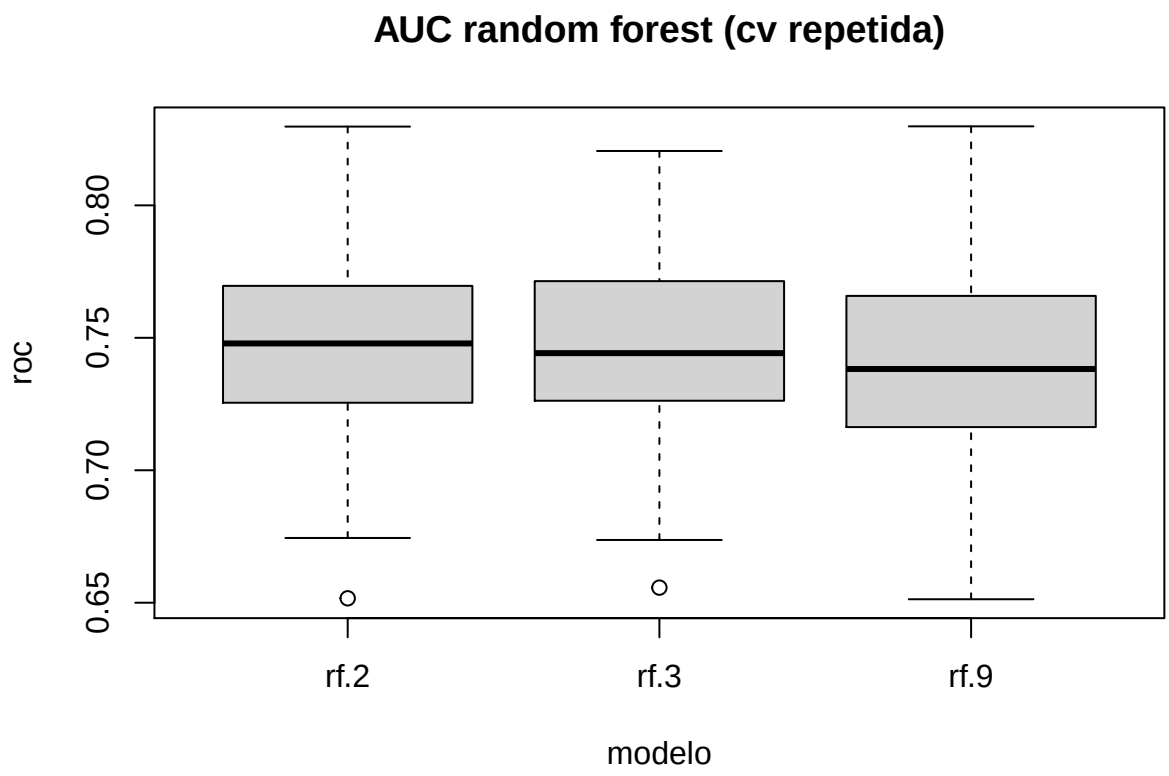
```
                method="rf", ntree= 500, nodesize = nodesizes[3], tuneGrid = rf.grid,
                trControl=trainControl, metric="ROC",
                sampsize=c(9*sampsizes[1], sampsizes[1]), strata=datMod$deny, replace = TRUE)


  set.seed(1234)

  rf.9 <- train(rl.5, data=datMod,
                method="rf", ntree= 500, nodesize = nodesizes[3], tuneGrid = rf.grid,
                trControl=trainControl, metric="ROC",
                sampsize=c(9*sampsizes[3], sampsizes[3]), strata=datMod$deny, replace = TRUE)
```

boxplot del repeated cv para las escogidas de cv

```
total.rf <- c()

i <-2
total.rf <- data.frame(roc=rf.2$resample[,1],
modelo=rep(paste("rf.", ifelse(i<10,paste0(i),i), sep=""),
nrow(rf.2$resample)))

i <-3
total.rf <-rbind(total.rf, data.frame(roc=rf.3$resample[,1],
modelo=rep(paste("rf.", ifelse(i<10,paste0(i),i), sep=""),
nrow(rf.3$resample))))

i <-9
total.rf <-rbind(total.rf, data.frame(roc=rf.9$resample[,1],
modelo=rep(paste("rf.", ifelse(i<10,paste0(i),i), sep=""),
nrow(rf.9$resample))))


results.rf <- c("rf.2",mean(rf.2$resample[,1]), sd(rf.2$resample[,1]))
results.rf <- transpose(as.data.frame(results.rf))
colnames(results.rf) <- c("model", "mean", "sd")
results.rf <- rbind(results.rf, c("rf.3",mean(rf.3$resample[,1]), sd(rf.3$resample[,1])))
results.rf <- rbind(results.rf, c("rf.9",mean(rf.9$resample[,1]), sd(rf.9$resample[,1])))



results.rf
```

```
##   model              mean                 sd
## 1  rf.2 0.747979739450371 0.0366041509059199
## 2  rf.3 0.747746049167002 0.0374889251477342
## 3  rf.9 0.740691947117283 0.0382086388331634
```

```
boxplot(roc~modelo,data=total.rf,main="AUC random forest (cv repetida)")
```



Se toma como modelo el rf.2 al tener menor sesgo y varianza que rf.3 (se osbserva mejor en la tabla) y r.9, es posible que haya sobreajuste al incrementar sd con mayour sampsize. El mtry escogido para rf.3 es el que da caret, al tener menor sesgo y varianza que los otros valores.

4.3 Gradient boosting

Los algoritmos de boosting están también basados en árboles por lo que tienen todas sus ventajas, pero son más agresivos a la hora de reducir el error. Esto es debido a que se centran en cada iteración en aquellas partes del train donde hay resíduos, explotando más toda la información presente en ese conjunto. Obviamente, el objetivo no es lograr residuos cero en el train ya que daría lugar a sobreajuste y malos resultados a la hora de generalizar con otras muestras.

Los hiperparámetros más importantes son la constante de regularización (shrinkage), número de árboles, n.trees, el mínimo de observaciones por nodo (n.minobsinnode) y el tamaño de la muestra, bag.fraction, el sampsize de la pasada sección. Tres comentarios:

- En cada iteración se evalúa un árbol. n.trees se asemeja al maxit de las redes y no al de random forest. Es decir, no hay un número de árboles en el que el error del train se estabiliza, tiende siempre a descender.
- El parámetro shrinkage es análogo al learning rate de las redes
- Los parámetros son iterdependientes.Por ejemplo, a menor shrinkage mayor n.trees se deberían tomar para compensar y conseguir niveles parecidos de AUC en este caso.

Los valores de shrinkage y n.trees se establecen de acuerdo a las recomendaciones de la teoría. Es decir, shrinkage=c(0.2,0.1,0.05,0.03,0.01,0.001) y n.trees=c(100,500,1000,5000). En consonancia con lo dicho en

random forest/bagging respecto al sobreajuste, n.minobsinnode tomará los valores c(20, 30, 40) y bag.fraction c(0.6, 0.8, 0.9) .

Finalmente, se dejará la profundidad de la iteración en dos para evaluar posibles interación más complejas que el valor por defecto.

Uso class.stratify.cv = TRUE para estratificar, pero no ha sido posible ni siquiera cuando uso solamente gbm, sin el wrapper de caret. Solución aprox para este cv y que haya bag.fraction? stratifiedkfolds normal y el conjunto train con 5 repeats como avvnet

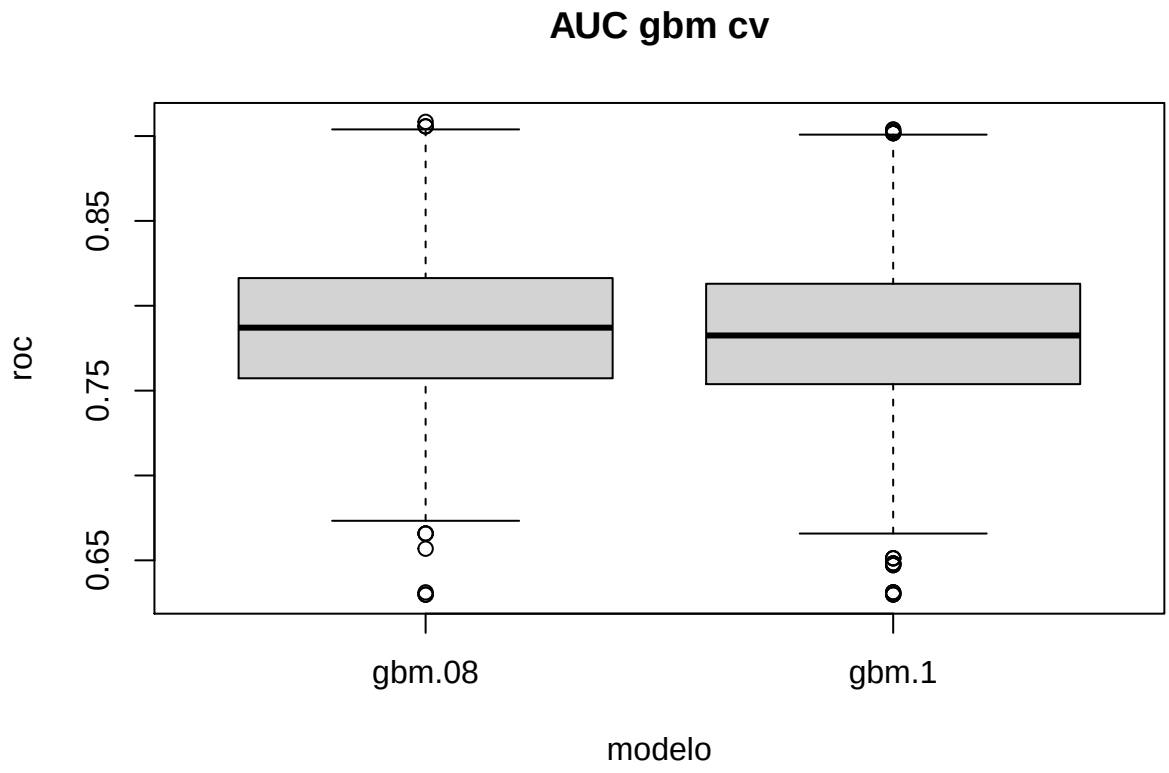
Cogeremos las mejores y haremos otro repeats con 20. En este caso bag.file será mas grande para compensar la perdida de randomness de no iterar en cada arbol con conjunto distinto unused argument (class.stratify.cv = TRUE)

```
set.seed(1234)
cvIndex <- createMultiFolds(factor(datMod$deny), k = n.folds, times = 5)

gbmgrid<-expand.grid(shrinkage=c(0.2,0.1,0.05,0.03,0.01,0.001),
 n.minobsinnode=c(5,10,20),
 n.trees=c(100,500,1000,5000),
 interaction.depth=c(2))

trainControl <- trainControl(index = cvIndex,
                             method = "cv",
                             #number = n.folds,
                             #repeats = 20,
                             classProbs = TRUE,
                             summaryFunction=twoClassSummary,
                             returnResamp="all"
                            )



set.seed(1234)
gbm.08<- train(rl.5,
               data=datMod,
 method="gbm",trControl=trainControl,tuneGrid=gbmgrid,
 distribution="bernoulli",verbose=FALSE,
 bag.fraction = 0.8, metric = "ROC")


set.seed(1234)
gbm.1<- train(rl.5,
               data=datMod,
 method="gbm",trControl=trainControl,tuneGrid=gbmgrid,
 distribution="bernoulli",verbose=FALSE,
 bag.fraction = 1, metric = "ROC")
```

Los resultados. Debido a la gran cantidad de modelos, ordeno por ROC decreciente y analizo ROCSD.

```
total.gbm <- c()

i <- 8
total.gbm<-rbind(total.gbm,data.frame(roc=gbm.08$resample[,'ROC'],
modelo=rep(paste("gbm.0", ifelse(i<10,paste0(i),i), sep=""),
```

```
nrow(gbm.08$resample))))

i <- 1
total.gbm<-rbind(total.gbm,data.frame(roc=gbm.1$resample[,'ROC'],
modelo=rep(paste("gbm.", ifelse(i<10,paste0(i),i), sep=""),
nrow(gbm.1$resample))))

boxplot(roc~modelo,data=total.gbm,main="AUC gbm cv")
```




```
gbm.08.res <- as.data.frame( gbm.08$results[,c('shrinkage', 'n.minobsinnode', 'n.trees','ROC', 'ROCSD')]

gbm.08.res <- gbm.08.res[order(gbm.08.res$ROC,decreasing = TRUE),]


gbm.1.res <- as.data.frame( gbm.1$results[,c('shrinkage', 'n.minobsinnode', 'n.trees','ROC', 'ROCSD')] )

gbm.1.res <- gbm.1.res[order(gbm.1.res$ROC,decreasing = TRUE),]


gbm.08.res[1:10,]
```

```
##    shrinkage n.minobsinnode n.trees       ROC      ROCSD
## 22     0.010             20     500 0.8131020 0.03866192
## 12     0.001             20    5000 0.8130486 0.03772944
## 41     0.050             10     100 0.8127144 0.03801464
```

```
## 18     0.010              10     500 0.8121632 0.03725102
## 45     0.050              20     100 0.8121560 0.03876843
## 8      0.001              10    5000 0.8121112 0.03678711
## 4      0.001               5    5000 0.8116107 0.03709323
## 14     0.010               5     500 0.8114969 0.03741707
## 23     0.010              20    1000 0.8113568 0.03705861
## 19     0.010              10    1000 0.8109634 0.03673810
```

```
gbm.1.res[1:10,]
```

```
##    shrinkage n.minobsinnode n.trees       ROC      ROCSD
## 41     0.050              10     100 0.8114345 0.03786318
## 8      0.001              10    5000 0.8113289 0.03792292
## 45     0.050              20     100 0.8110862 0.03768932
## 18     0.010              10     500 0.8110789 0.03809657
## 4      0.001               5    5000 0.8109204 0.03840008
## 12     0.001              20    5000 0.8107523 0.03779113
## 22     0.010              20     500 0.8107260 0.03771005
## 37     0.050               5     100 0.8105040 0.03825908
## 14     0.010               5     500 0.8103659 0.03825139
## 23     0.010              20    1000 0.8096045 0.03681425
```

Se asumen que las pequeñas mejoras del ROCSD no compensa para elegir otra alternativa al mejor ajuste aportado por caret. Usamos el mejor modelo de cada valor de bag.fraction para la cv con 20 repeticiones

```
set.seed(1234)
cvIndex <- createMultiFolds(factor(datMod$deny), k = n.folds, times = 20)


trainControl <- trainControl(index = cvIndex,
                             method = "cv",
                             #number = n.folds,
                             #repeats = 20,
                             classProbs = TRUE,
                             summaryFunction=twoClassSummary,
                             returnResamp="all"
                            )


set.seed(1234)

gbmgrid<-expand.grid(shrinkage=c(0.010),
 n.minobsinnode=c(20),
 n.trees=c(500),
 interaction.depth=c(2))

gbm.1<- train(rl.5,
               data=datMod,
 method="gbm",trControl=trainControl,tuneGrid=gbmgrid,
 distribution="bernoulli",verbose=FALSE,
 bag.fraction = 0.8, metric = "ROC"
 )
```

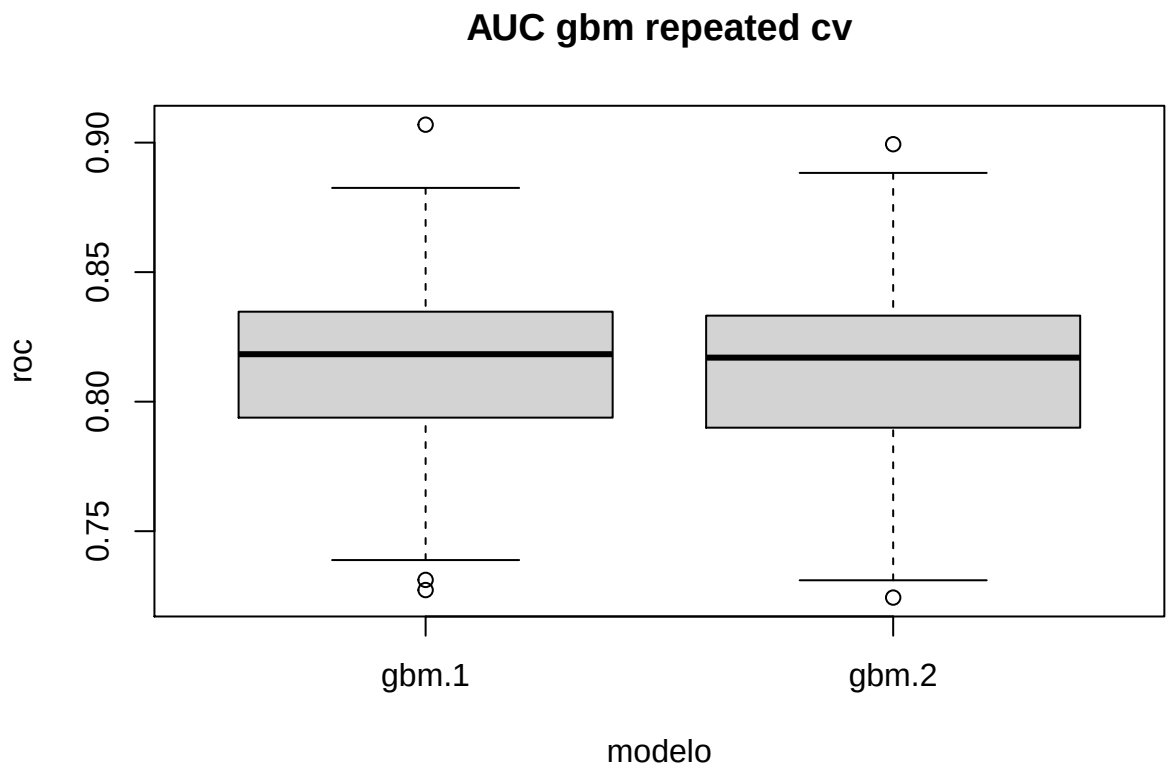
```
set.seed(1234)

gbmgrid<-expand.grid(shrinkage=c(0.050),
 n.minobsinnode=c(10),
 n.trees=c(100),
 interaction.depth=c(2))

gbm.2<- train(rl.5,
               data=datMod,
 method="gbm",trControl=trainControl,tuneGrid=gbmgrid,
 distribution="bernoulli",verbose=FALSE,
 bag.fraction = 1, metric = "ROC"
 )
```

Boxplot del repeated cv para las escogidas de cv

```
total.gbm <- c()

i <- 1
total.gbm<-rbind(total.gbm,data.frame(roc=gbm.1$resample[,'ROC'],
modelo=rep(paste("gbm.", ifelse(i<10,paste0(i),i), sep=""),
nrow(gbm.1$resample))))

i <- 2
total.gbm<-rbind(total.gbm,data.frame(roc=gbm.2$resample[,'ROC'],
modelo=rep(paste("gbm.", ifelse(i<10,paste0(i),i), sep=""),
nrow(gbm.2$resample))))



results.gbm <- c("gbm.1",mean(gbm.1$resample[,'ROC']), sd(gbm.1$resample[,'ROC']))
results.gbm <- transpose(as.data.frame(results.gbm))

colnames(results.gbm) <- c("model", "mean", "sd")
results.gbm <- rbind(results.gbm, c("gbm.2",mean(gbm.2$resample[,'ROC']), sd(gbm.2$resample[,'ROC'])))




results.gbm
```

```
##   model              mean                 sd
## 1 gbm.1 0.812313451028019 0.0341838974256471
## 2 gbm.2 0.810932121371923 0.0343420912451332
```

```
boxplot(roc~modelo,data=total.gbm,main="AUC gbm repeated cv")
```



gbm.1 menor sesgo y varianza, como se aprecia en la tabla y el boxplot anteriores.

4.4 Support Vector Machines

flexible con varios tipos de kernel a elegir, pocos hiperparámetros a optimizar por lo que es fácil ver la influencia o interacciones de estos. Además es un competidor natural de la regresión lineal/logística para problemas próximos a la linealidad entre los predictores y la variable objetivo.

Sin embargo, puede ser lento en converger, pudiendo estancarse en un mínimo local o presentar problemas de overflow como en las redes neuronales. Sufre como los otros algoritmos clásicos , a diferencia de los modelos basados en árboles, de problemas con el tratamiento de missings así como mayor sensibilidad frente a variables irrelevantes o con alta correlación, categorías poco representadas. Por tanto, son menos robustos frente a la depuración de datos y EDA.

Hay dos tipos de SVM, lineales y no lineales:

- Lineales. Presentan un único parámetro de penalización, C, para ajustar cuán agresivo el SVM ha de ser a la hora de aceptar observaciones dentro del margen de separación. Un C alto intentar reducir el sesgo (menos underfitting, pero más propenso a undefitting) mientras que, al contrario, un C bajo busca reducir la varianza y, por tanto, hacer el algortimo más robusto.
- No lineales. Amplían la dimensionalidad del problema mediante transformaciones a partir de las variables input con el objetivo de encotrar un hiperplano eficaz a la hora de separar las clases de la variable objetivo. Las transformaciones se realizan a través de funciones Kernel establecidas para reducir el tiempo computacional, bajando el número de productos escalares. Se tratarán dos tipos:

- Polinomiales. Tienen dos parámetros a optimizar el C del lineal más un parámetro d, para el grado de polinomio. A mayor C y de menor sesgo y a menores C y d menor varianza.
- RBF. Tienen dos parámetros a optimizar el C del lineal más un parámetro sigma, que controla la dimensionalidad del espacio input transformado, como el d en los polinomiales. A mayor C y de menor sesgo y a menores C y d menor varianza.

Hay interdependencia entre los parámetros C y los del kernel.

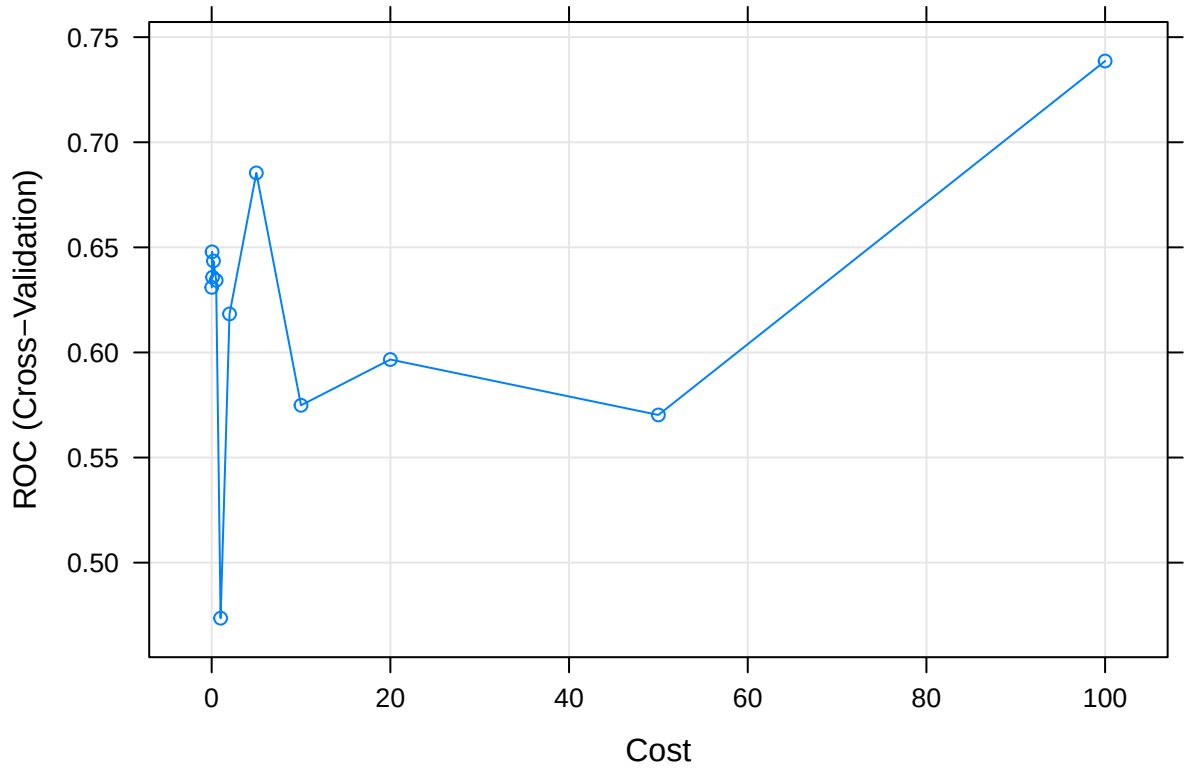
Como SVM tiende a ser lento en cuanto a tiempo CPU recurrimos a la computación en paralelo. Se prueban una serie de valores de C vistos en la referencia y se añade el 20, 50 y 100 para ver el comportamiento a valores más altos.

```
set.seed(1234)
c.grid<-expand.grid(C=c(0.01,0.05,0.1,0.2,0.5,1,2,5,10, 20,50,100))

cvIndex <- createFolds(datMod$deny, n.folds, returnTrain = T)


control<-trainControl(   index = cvIndex,
                         method = "cv",
                         summaryFunction=twoClassSummary,
                         #number = n.folds,
                         classProbs=T,
                         savePredictions = "all")




set.seed(1234)

GS_T0 <- Sys.time()
cluster <- makeCluster(detectCores() - 2) # number of cores
registerDoParallel(cluster) # register the parallel processing

SVM.lineal <- train(rl.5, data=datMod,
  method="svmLinear",trControl=control,
 tuneGrid=c.grid,verbose=FALSE, metric = 'ROC')


stopCluster(cluster) # shut down the cluster
registerDoSEQ(); #  force R to return to single threaded processing
GS_T1 <- Sys.time()
#GS_T1-GS_T0



SVM.lineal$results[,c("C",'ROC', 'ROCSD')]
```

```
##        C       ROC      ROCSD
## 1  1e-02 0.6309539 0.12823072
## 2  5e-02 0.6478727 0.16377952
## 3  1e-01 0.6357993 0.16492408
## 4  2e-01 0.6435398 0.16923171
## 5  5e-01 0.6343276 0.12785372
```

```
## 6  1e+00 0.4735693 0.09282426
## 7  2e+00 0.6183253 0.13765216
## 8  5e+00 0.6854273 0.13612031
## 9  1e+01 0.5748511 0.20071289
## 10 2e+01 0.5966382 0.13027767
## 11 5e+01 0.5702661 0.08336494
## 12 1e+02 0.7386548 0.08858576
```

```
plot(SVM.lineal)
```



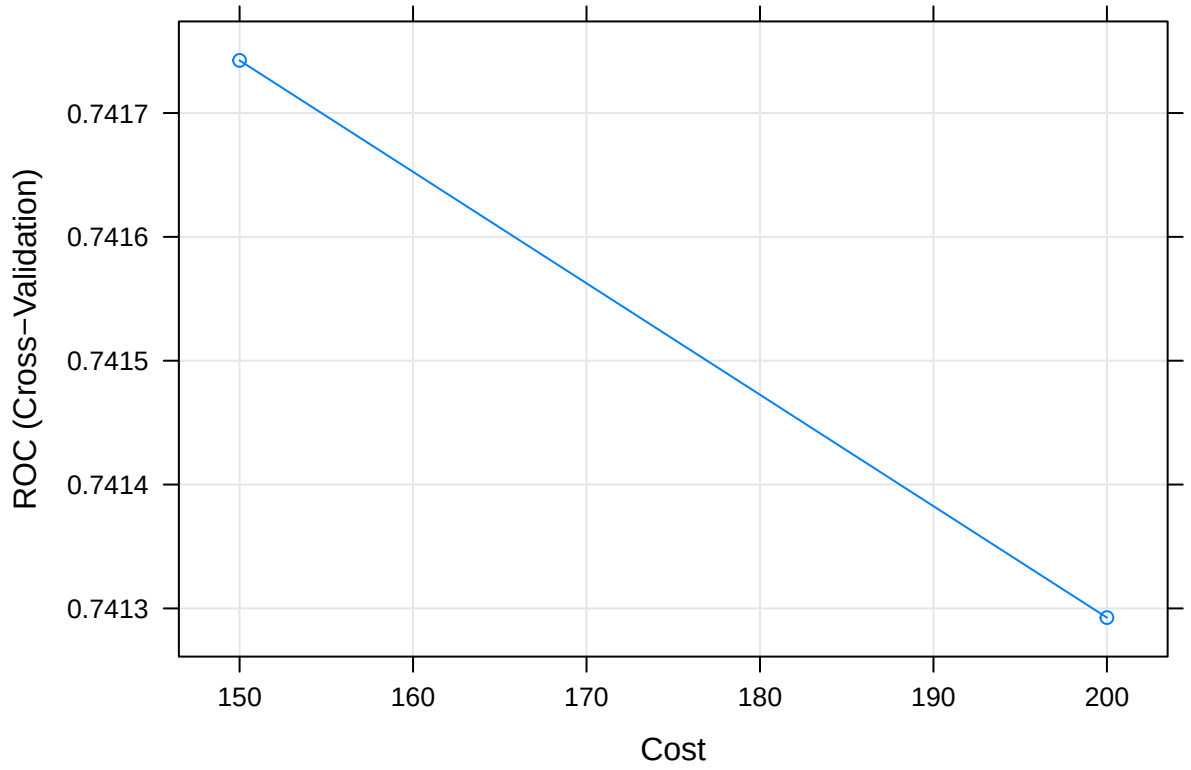
Se observa que el ROC con C=100 da buenos resultados tanto en ROC como en ROCSD por lo que se escoge como

```
c.grid<-expand.grid(C=c(150, 200))

control<-trainControl(   index = cvIndex,
                         method = "cv",
                         summaryFunction=twoClassSummary,
                         #number = n.folds,
                         classProbs=T,
                         savePredictions = "all")

set.seed(1234)

GS_T0 <- Sys.time()
cluster <- makeCluster(detectCores() - 2) # number of cores
```

```
registerDoParallel(cluster) # register the parallel processing

SVM.lineal <- train(rl.5, data=datMod,
  method="svmLinear",trControl=control,
 tuneGrid=c.grid,verbose=FALSE, metric = 'ROC')


stopCluster(cluster) # shut down the cluster
registerDoSEQ(); #  force R to return to single threaded processing
GS_T1 <- Sys.time()
#GS_T1-GS_T0

SVM.lineal$results[,c("C",'ROC', 'ROCSD')]
```

```
##     C       ROC      ROCSD
## 1 150 0.7417425 0.05249592
## 2 200 0.7412926 0.10464121
```

```
plot(SVM.lineal)
```



C = 150 ofrece un mejor ROC/ROCSD que C = 100. Sin embargo, el tiempo de computación llega a 22 minutos en vez de 8 con el primer mallado, contando con la computación en paralelo. Esto es debido a que se está forzando a ser más agresivo.Por tanto, se deja aquí.

tuneado de C, degree, SVM polinomial. Se reduce el mallado por el tiempo computacional usualmente

requerido. la razón es que muy pocas veces SVM polinomial es la mejor opción: si la separación es lineal, es mejor SVM lineal; si no es lineal, SVM RBF se suele adaptar mejor que el SVM polinomial.

```
SVM.poly.grid<-expand.grid(C=c(0.01,0.1,1,10),
degree=c(2,3),scale=c(0.1,1,5))




control<-trainControl(   index = cvIndex,
                         method = "cv",
                         summaryFunction=twoClassSummary,
                         #number = n.folds,
                         classProbs=T,
                         savePredictions = "all")

set.seed(1234)

GS_T0 <- Sys.time()
cluster <- makeCluster(detectCores() - 2) # number of cores
registerDoParallel(cluster) # register the parallel processing


SVM.poly<- train( rl.5, data=datMod,
  method="svmPoly",trControl=control,
 tuneGrid=SVM.poly.grid,verbose=FALSE, metric = 'ROC')


stopCluster(cluster) # shut down the cluster
registerDoSEQ(); #  force R to return to single threaded processing
GS_T1 <- Sys.time()
```

grado dos mejor que el tres, el mejor el bestfit de caret

```
SVM.poly$bestTune
```

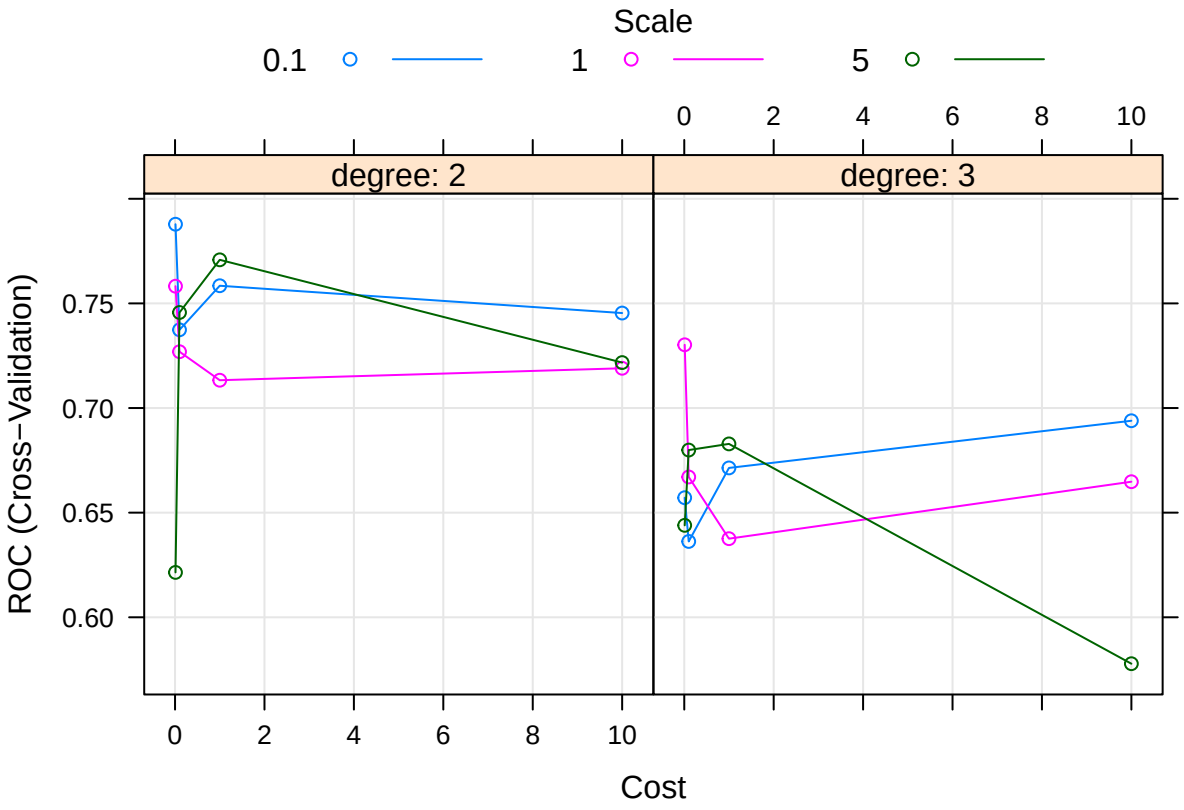
```
##   degree scale    C
## 1      2   0.1 0.01
```

```
SVM.poly.results <- as.data.frame( SVM.poly$results[,c("C","degree", "scale", 'ROC', 'ROCSD')] )

SVM.poly.results <- SVM.poly.results[order(SVM.poly.results$ROC,decreasing = TRUE),]

SVM.poly.results[1:10,]
```

```
##        C degree scale       ROC      ROCSD
## 1   0.01      2   0.1 0.7878393 0.02162412
## 15  1.00      2   5.0 0.7708266 0.06562337
## 13  1.00      2   0.1 0.7584481 0.02799619
## 2   0.01      2   1.0 0.7582264 0.02242267
## 9   0.10      2   5.0 0.7456575 0.03702796
## 19 10.00      2   0.1 0.7453971 0.03730672
```

```
## 7   0.10      2   0.1 0.7373797 0.05529735
## 5   0.01      3   1.0 0.7302276 0.05085647
## 8   0.10      2   1.0 0.7269180 0.03580895
## 21 10.00      2   5.0 0.7217284 0.06863361
```

```
plot(SVM.poly)
```



tuneado de C, degree, SVM RBF. Se dejan los sigmas del grid ejemplo, los C como el caso lineal.

```
SVM.rbf.grid<-expand.grid(C=c(0.01,0.05,0.1,0.2,0.5,1,2,5,10, 20,50,100),
 sigma=c(0.0001,0.005,0.01,0.05))

control<-trainControl(   index = cvIndex,
                         method = "cv",
                         summaryFunction=twoClassSummary,
                         #number = n.folds,
                         classProbs=T,
                         savePredictions = "all")

set.seed(1234)

GS_T0 <- Sys.time()
cluster <- makeCluster(detectCores() - 2) # number of cores
registerDoParallel(cluster) # register the parallel processing
```

```
SVM.rbf<- train(rl.5, data=datMod, method="svmRadial",trControl=control,
 tuneGrid=SVM.rbf.grid,verbose=FALSE, metric = 'ROC')
```

```
## maximum number of iterations reached 0.004976795 0.005456482
```

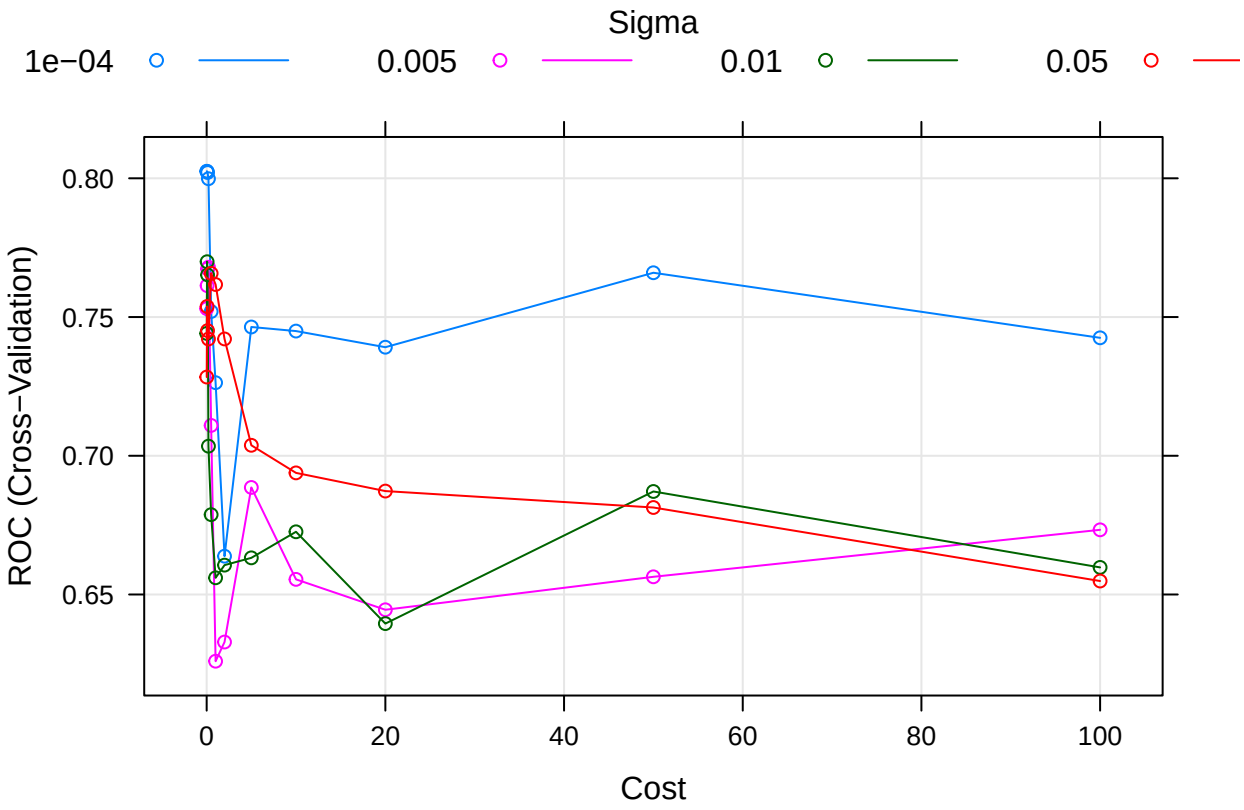
```
stopCluster(cluster) # shut down the cluster
registerDoSEQ(); #  force R to return to single threaded processing
GS_T1 <- Sys.time()
```

Los resultados, tendencia no clara. Parece que podría haber un mínimo local con C y sigma bajos.

```
SVM.rbf.results <- as.data.frame(SVM.rbf$results[,c("C","sigma", 'ROC', 'ROCSD')])

SVM.rbf.results <- SVM.rbf.results[order(SVM.rbf.results$ROC,decreasing = TRUE),]

SVM.rbf.results[1:10,]
```

```
##         C sigma       ROC      ROCSD
## 1    0.01 1e-04 0.8025430 0.03636413
## 5    0.05 1e-04 0.8024902 0.03640647
## 9    0.10 1e-04 0.8020922 0.03590381
## 13   0.20 1e-04 0.7998506 0.03556375
## 7    0.05 1e-02 0.7699369 0.01370544
## 14   0.20 5e-03 0.7679511 0.01876594
## 10   0.10 5e-03 0.7674144 0.01913360
## 41 50.00 1e-04 0.7659593 0.03377917
## 20   0.50 5e-02 0.7657438 0.03065665
## 11   0.10 1e-02 0.7651984 0.02263299
```

```
plot(SVM.rbf)
```



Vamos a coger los modelos polinomial y RBF con ROC más altos para la validación cruzada.

```
set.seed(1234)


SVM.rbf.grid<-expand.grid(C=c(0.01),
 sigma=c(0.0001))


SVM.poly.grid<-expand.grid(C=c(0.01),
degree=c(2),scale=c(0.1))

cvIndex <- createMultiFolds(factor(datMod$deny), k = n.folds, times = 20)

trainControl <- trainControl(index = cvIndex,
                             method = "cv",
                             classProbs = TRUE,
                             summaryFunction=twoClassSummary,
                             returnResamp="all"
                            )


set.seed(1234)

GS_T0 <- Sys.time()
cluster <- makeCluster(detectCores() - 2) # number of cores
```

```
registerDoParallel(cluster) # register the parallel processing

SVM.1<- train( rl.5, data=datMod,
  method="svmPoly",trControl=trainControl,
 tuneGrid=SVM.poly.grid,verbose=FALSE, metric = 'ROC')

stopCluster(cluster) # shut down the cluster
registerDoSEQ(); #  force R to return to single threaded processing
GS_T1 <- Sys.time()



set.seed(1234)

GS_T0 <- Sys.time()
cluster <- makeCluster(detectCores() - 2) # number of cores
registerDoParallel(cluster) # register the parallel processing

SVM.2<- train(rl.5, data=datMod, method="svmRadial",trControl=trainControl,
 tuneGrid=SVM.rbf.grid,verbose=FALSE, metric = 'ROC')
```

```
## maximum number of iterations reached 0.0049773 0.005478093
```

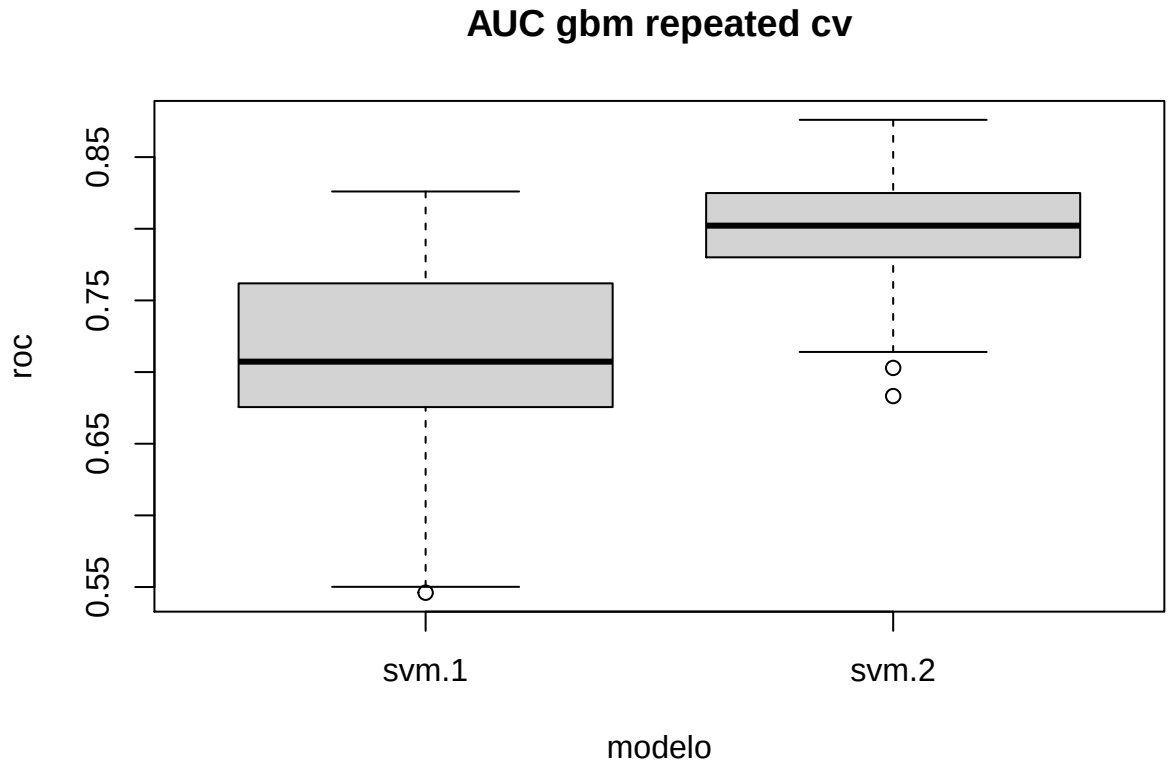
```
stopCluster(cluster) # shut down the cluster
registerDoSEQ(); #  force R to return to single threaded processing
GS_T1 <- Sys.time()
```

Resultados repeated cv

```
total.svm <- c()

i <- 1
total.svm<-rbind(total.svm,data.frame(roc=SVM.1$resample[,'ROC'],
modelo=rep(paste("svm.", ifelse(i<10,paste0(i),i), sep=""),
nrow(SVM.1$resample))))

i <- 2
total.svm<-rbind(total.svm,data.frame(roc=SVM.2$resample[,'ROC'],
modelo=rep(paste("svm.", ifelse(i<10,paste0(i),i), sep=""),
nrow(SVM.2$resample))))



results.svm <- c("svm.1",mean(SVM.1$resample[,'ROC']), sd(SVM.1$resample[,'ROC']))
results.svm <- transpose(as.data.frame(results.svm))

colnames(results.svm) <- c("model", "mean", "sd")
results.svm <- rbind(results.svm, c("svm.2",mean(SVM.2$resample[,'ROC']), sd(SVM.2$resample[,'ROC'])))




results.svm
```

```
##   model              mean                 sd
## 1 svm.1 0.707935426237954 0.0613045660288655
## 2 svm.2 0.798233429193029 0.0367641549273579
```

```
boxplot(roc~modelo,data=total.svm,main="AUC gbm repeated cv")
```



svm.2, el RBF, claramente superior

4.5 Ensamblado

El mejor modelo de cada algoritmo en este apartado, más probabilidades a priori de dar mejor performance.

```
source("cruzadas ensamblado binaria fuente_mb.R")

datMod1 <- copy(datMod)
levels(datMod1$deny) <- c("No", "Yes")

vardep<-"deny"
listconti<- c("chist.1", "lvrat" , "chist.2" , "insurance.yes" , "phist.no" )
listclass<-c("")
grupos<- n.folds
sinicio<-2234
repe<-50
```

```
medias1<-cruzadalogistica(data=datMod1,
                          vardep=vardep,listconti=listconti,
                          listclass=listclass,grupos=grupos,sinicio=sinicio,repe=repe)

medias1bis<-as.data.frame(medias1[1])
medias1bis$modelo<-"Logistica"
predi1<-as.data.frame(medias1[2])
predi1$logi<-predi1$Yes
```

```
# avvnet.1$bestTune
medias2<-cruzadaavnnetbin(data=datMod1,
                          vardep=vardep,listconti=listconti,
                          listclass=listclass,grupos=grupos,sinicio=sinicio,repe=repe,
                          size=c(25),decay=c(0.1),repeticiones=5,itera=100)
```

```
##   size decay   bag  Accuracy     Kappa  AccuracySD    KappaSD
## 1   25   0.1 FALSE 0.9097087 0.3619727 0.007987575 0.06391367
```

```
medias2bis<-as.data.frame(medias2[1])
medias2bis$modelo<-"avnnet"
predi2<-as.data.frame(medias2[2])
predi2$avnnet<-predi2$Yes
```

introduzco strata para hacerlo estratificada

```
#   rf.2 <- train(rl.5, data=datMod,
#                 method="rf", ntree= 500, nodesize = nodesizes[2], tuneGrid = rf.grid,
#                 trControl=trainControl, metric="ROC",
#                 sampsize=c(9*sampsizes[1], sampsizes[1]), strata=datMod$deny, replace = TRUE)
# rf.2$bestTune

  medias3<-cruzadarfbin_mb(data=datMod1,
                     vardep=vardep,listconti=listconti,
                     listclass=listclass,grupos=grupos,sinicio=sinicio,repe=repe,
                     mtry=4,ntree=500,nodesize=nodesizes[2],replace=TRUE, sampsize=c(9*sampsizes[1],
```

```
## Warning in if (sampsize == 1) {: the condition has length > 1 and only the first
## element will be used

## Warning in if (sampsize != 1) {: the condition has length > 1 and only the first
## element will be used

##   mtry  Accuracy     Kappa  AccuracySD    KappaSD
## 1    4 0.9100795 0.3943787 0.008377595 0.06522353
```

```
medias3bis<-as.data.frame(medias3[1])
medias3bis$modelo<-"rf"
predi3<-as.data.frame(medias3[2])
predi3$rf<-predi3$Yes
```

incluyo bag.fraction, para tunear el valor

```
medias4<-cruzadagbmbin_mb(data=datMod1,
                        vardep=vardep,listconti=listconti,
                        listclass=listclass,grupos=grupos,sinicio=sinicio,repe=repe,
                        n.minobsinnode=20,shrinkage=0.010,n.trees=500,interaction.depth=2, bag.fraction =
```

```
##   n.minobsinnode shrinkage n.trees interaction.depth  Accuracy     Kappa
## 1             20      0.01     500                 2 0.9082265 0.3384812
##    AccuracySD    KappaSD
## 1 0.007085206 0.06723261
```

```
medias4bis<-as.data.frame(medias4[1])
medias4bis$modelo<-"gbm"
predi4<-as.data.frame(medias4[2])
predi4$gbm<-predi4$Yes
```

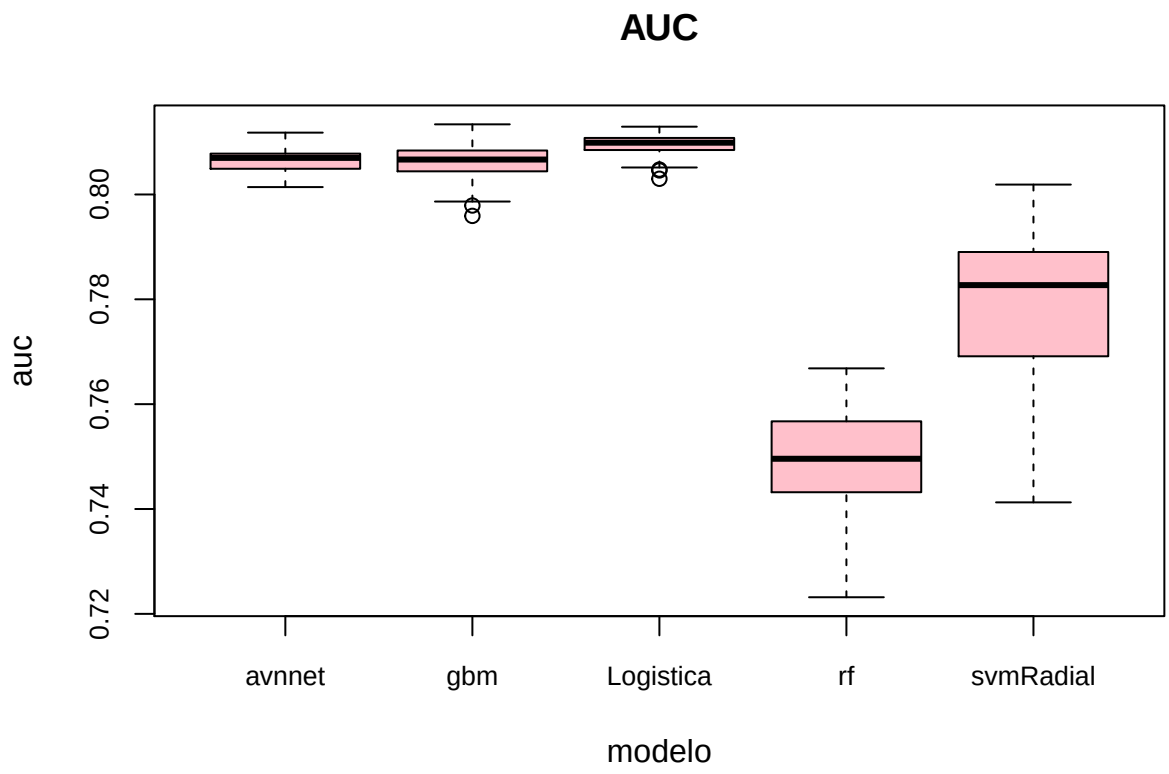
```
medias5<-cruzadaSVMbinRBF(data=datMod1,
                          vardep=vardep,listconti=listconti,
                          listclass=listclass,grupos=grupos,
                          sinicio=sinicio,repe=repe,
                          C=0.01,sigma=0.0001)
```

```
## maximum number of iterations reached 0.00906994 0.009100601maximum number of iterations reached 0.008
## 1 0.01 1e-04 0.9082913 0.2611561 0.004842267 0.06021596
```

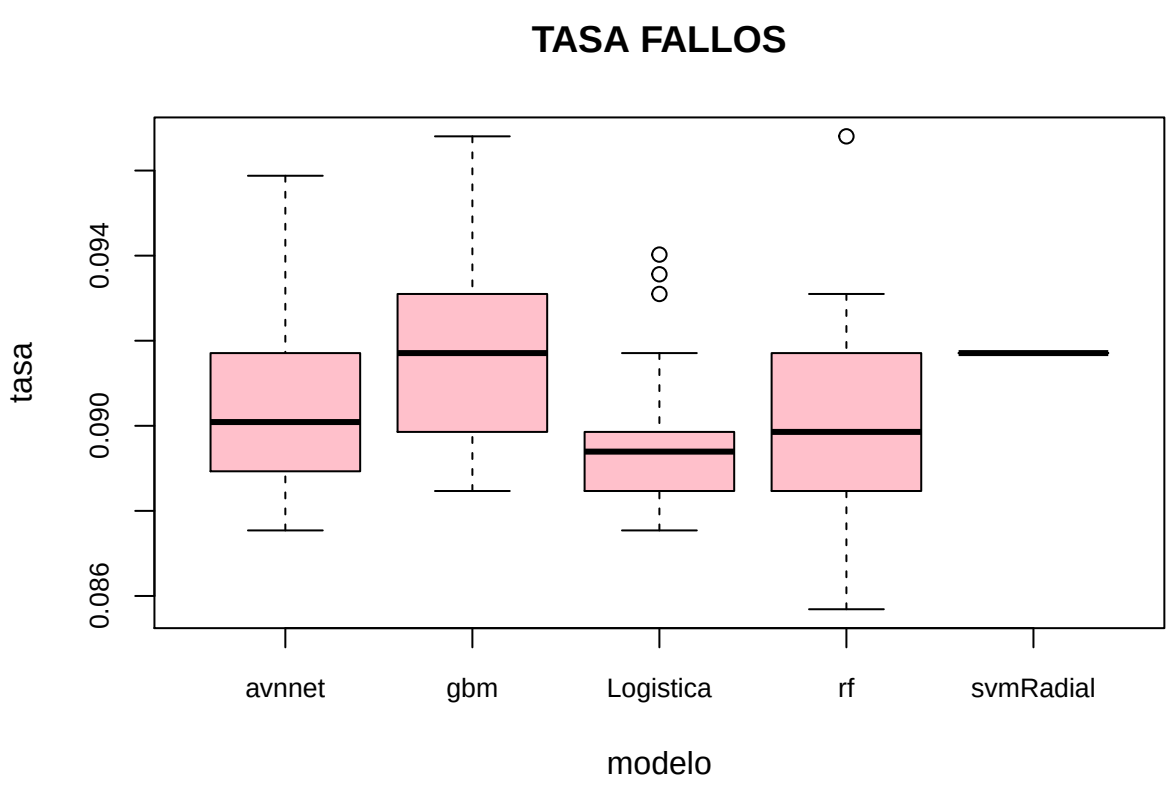
```
medias5bis<-as.data.frame(medias5[1])
medias5bis$modelo<-"svmRadial"
predi5<-as.data.frame(medias5[2])
predi5$svmRadial<-predi5$Yes
```

XX

```
union1<-rbind(medias1bis,medias2bis,
              medias3bis,medias4bis,medias5bis)


par(cex.axis=0.8)
boxplot(data=union1,auc~modelo,col="pink",main='AUC')
```




```
boxplot(data=union1,tasa~modelo,col="pink",main='TASA FALLOS')
```



La mejor la logística al presentar el menor sesgo y varianza en el AUC. Hay que recordar que el AUC es una métrica independiente del punto de corte, lo que supone una ventaja frente a la tasa de fallos. Esto hace que el AUC sea un métrica para evaluar más significativa que la tasa de fallos. Sin embargo, se deberá estudiar el punto de corte para maximizar la performance del modelo con mayor AUC en cuanto al objetivo en cada problema en concreto. El punto de corte tomado por ahora ha sido 0.5.

En la tasa de fallos, la logística a logrado el menor sesgo y la segunda varianza más baja. También esta segunda métrica da a logística como modelo ganador.

Hay otros dos modelos con un AUC parecido a logística, avvnet y gbm, ambos con un menor sesgo y varianzas relativos. No obstante estos, modelos presentan una tasa fallo similar o incluso peor (gbm) en cuanto sesgo-varianza respecto a los otros tres algoritmos. Bajo esta métrica, gbm presenta un sesgo y varianza altos respecto a otros algoritmos.

En este punto, se realizan unos ensamblados para intentar bajar la varianza y hacer modelos más resistentes al sobreajuste. Esto se realizan ponderando las predicciones de los 5 modelos base.

En primer lugar, se analizan las correlaciones en las predicciones. Los tres modelos con mayores AUC - logística, avnnet y gbm - tienen un valor de correlación muy alto, indicando que no mucha ganancia se puede sacar haciendo ensamblados. En cambio, con rf y svmRadial, los valores son distintos pero sus AUC son mucho menores. Se realizará, a modo de test, un ensamblado de logística y svmRadial, al presentar este último un mayor AUC que rf.

```
unipredi<-cbind(predi1,predi2,predi3,predi4,predi5)

unigraf<-unipredi[unipredi$Rep=="Rep01",]
# Correlaciones entre predicciones de cada algoritmo individual
solos<- c("logi", "avnnet","rf" , "gbm", "svmRadial" )
```

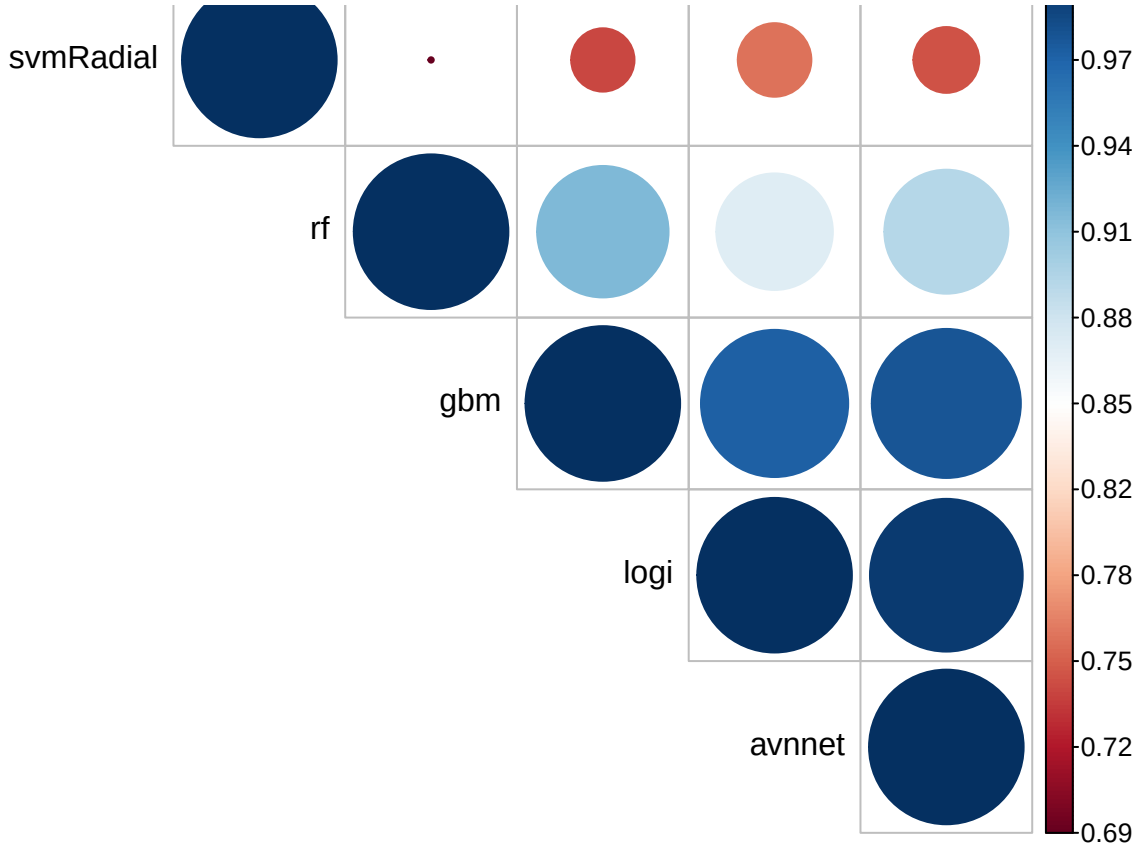
```
mat<-unigraf[,solos]
matrizcorr<-cor(mat)
matrizcorr
```

```
##                logi    avnnet        rf       gbm svmRadial
## logi      1.0000000 0.9926460 0.8669065 0.9711279 0.7617054
## avnnet    0.9926460 1.0000000 0.8901050 0.9781955 0.7478299
## rf        0.8669065 0.8901050 1.0000000 0.9146288 0.6919467
## gbm       0.9711279 0.9781955 0.9146288 1.0000000 0.7436175
## svmRadial 0.7617054 0.7478299 0.6919467 0.7436175 1.0000000
```

```
library(corrplot)
```

```
## corrplot 0.88 loaded
```

```
corrplot(matrizcorr, type = "upper", order = "hclust",
         tl.col = "black", tl.srt = 45,is.corr=FALSE)
```

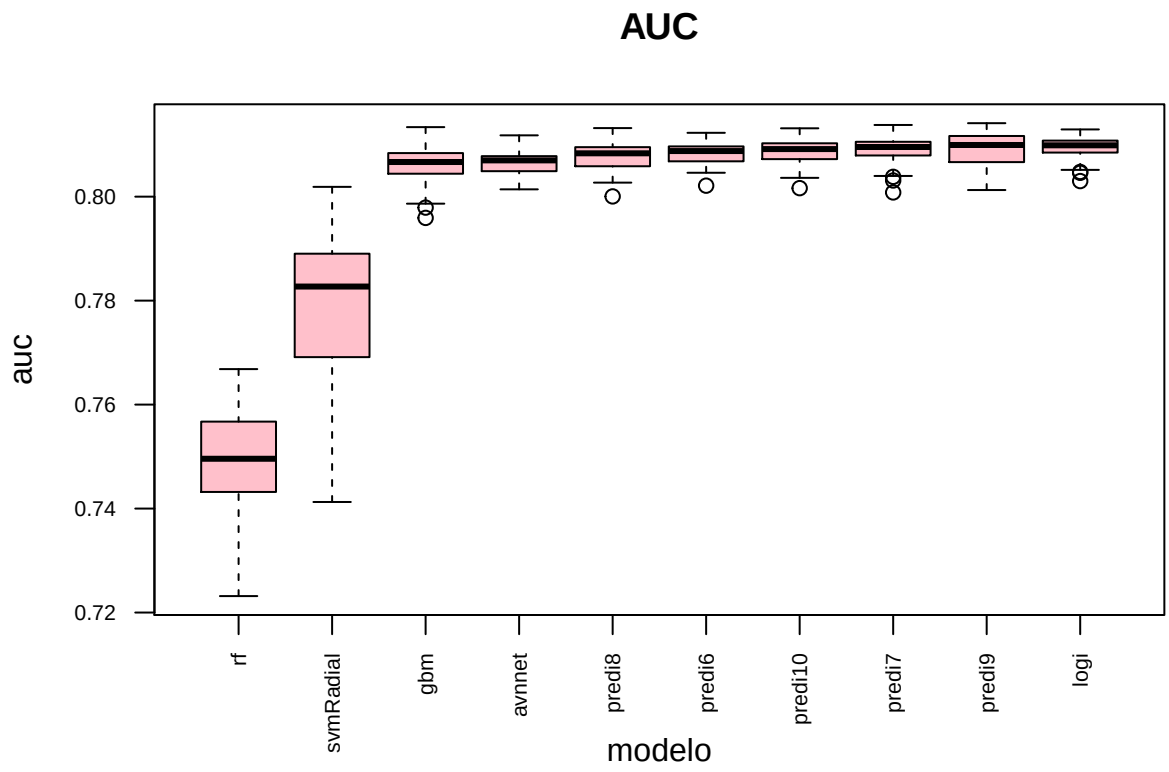


Se prueban las combinaciones de los tres algoritmos con mejor sesgo-varianza de AUC junto con logística y svmRadial.

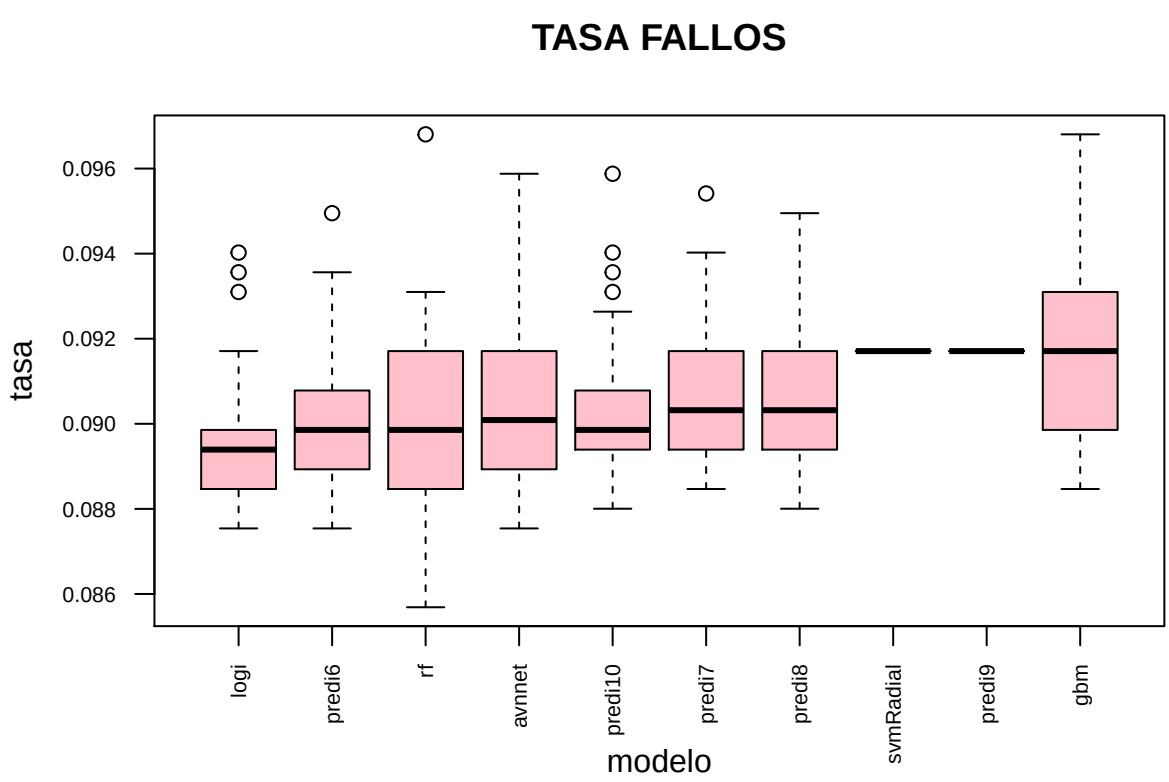
```
# Esto es para eliminar columnas duplicadas
unipredi<- unipredi[, !duplicated(colnames(unipredi))]
```

```
unipredi$predi6<-(unipredi$logi+unipredi$avnnet)/2
unipredi$predi7<-(unipredi$logi+unipredi$gbm)/2
unipredi$predi8<-(unipredi$avnnet+unipredi$gbm)/2
unipredi$predi9<-(unipredi$logi+unipredi$svmRadial)/2
unipredi$predi10<-(unipredi$logi+unipredi$avnnet+unipredi$gbm)/3


listado<-c("logi", "avnnet", "rf","gbm", "svmRadial","predi6", "predi7",
           "predi8","predi9","predi10")


tasafallos<-function(x,y) {
  confu<-confusionMatrix(x,y)
  tasa<-confu[[3]][1]
  return(tasa)
}

auc<-function(x,y) {
  curvaroc<-roc(response=x,predictor=y)
  auc<-curvaroc$auc
  return(auc)
}

# Se obtiene el numero de repeticiones CV y se calculan las medias por repe en
# el data frame medias0

repeticiones<-nlevels(factor(unipredi$Rep))
unipredi$Rep<-as.factor(unipredi$Rep)
unipredi$Rep<-as.numeric(unipredi$Rep)


medias0<-data.frame(c())
for (prediccion in listado)
{
  unipredi$proba<-unipredi[,prediccion]
  unipredi[,prediccion]<-ifelse(unipredi[,prediccion]>0.5,"Yes","No")
  for (repe in 1:repeticiones)
  {
    paso <- unipredi[(unipredi$Rep==repe),]
    pre<-factor(paso[,prediccion])
    archi<-paso[,c("proba","obs")]
    archi<-archi[order(archi$proba),]
    obs<-paso[,c("obs")]
    tasa=1-tasafallos(pre,obs)
    t<-as.data.frame(tasa)
    t$modelo<-prediccion
    auc<-suppressMessages(auc(archi$obs,archi$proba))
    t$auc<-auc
    medias0<-rbind(medias0,t)
  }
}

# Finalmente boxplot
```

```
medias0$modelo <- with(medias0,
                       reorder(modelo,auc, mean))
par(cex.axis=0.7,las=2)
boxplot(data=medias0,auc~modelo,col="pink", main='AUC')
```




```
medias0$modelo <- with(medias0,
                       reorder(modelo,tasa, mean))
par(cex.axis=0.7,las=2)
boxplot(data=medias0,tasa~modelo,col="pink", main='TASA FALLOS')
```



Los ensamblados producen una mejora, sobre todo en sesgo de AUC, frente a otros algoritmos que no sea la logística. Un claro ejemplo es predi9 respecto svmRadial o predi7 respecto a gbm. En ambos casos se aprecia una mejora en sesgo y también en varianza.

Sin embargo, logística sigue teniendo el mejor sesgo-varianza en AUC. Predi6 parece tener una varianza algo menor, aunque un mayor sesgo. Además la regresión logística es muy descriptivo con coeficientess fácilmente interpretables. Por tanto, se asume que esa pequeño descenso en varianza no compensa como para nombrar predi6 modelo ganador.

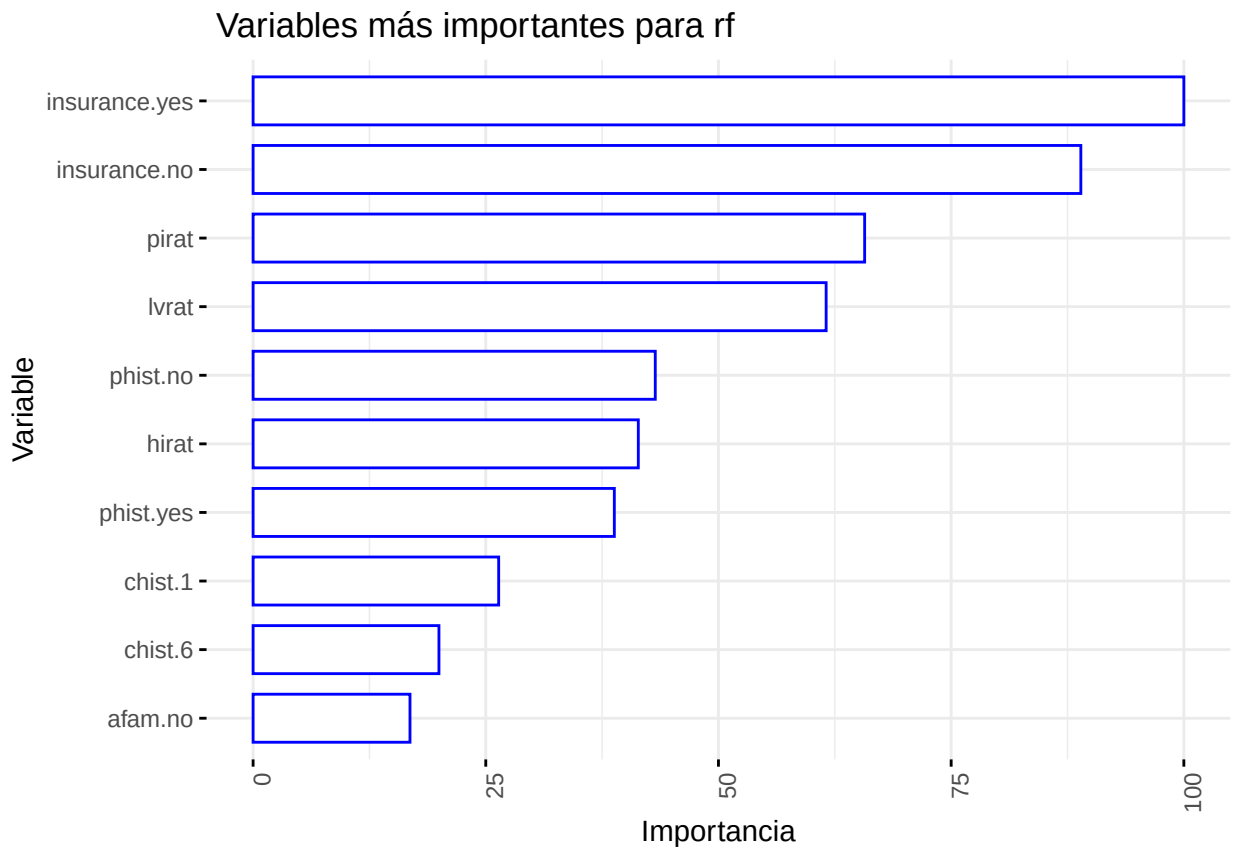
4.6 Análisis, decisiones y conclusiones

El modelo a escoger para el problema teniendo en cuenta la selección de variables, grid search y remuestreo usados es la regresión logística. La regresión logística aporta el menor sesgo y la segunda menor varianza de los modelos evaluados. Además, tiene la ventaja de ser un modelo más simple y descriptivo respecto a otros algoritmos base u ensamblados como Predi6.

Por supuesto, convendría evaluar modelos con otros conjuntos de variables y combinaciones de estos. A modo de ejemplo, se expone aquí el código a seguir para buscar conjuntos candidatos con random forest.

```
    trainControl <- trainControl(method="cv",
                         summaryFunction=twoClassSummary,
                         number = n.folds,
                         classProbs=T,
                         savePredictions = "all")

    rf.grid <- expand.grid(mtry=5)
```

```
    set.seed(1234)

rfFit.example <- train(as.formula(deny  ~ .), data=datMod,
                   method="rf", ntree= 100, nodesize = nodesize, tuneGrid = rf.grid, trControl=trainCon


vars_imp <-  varImp(rfFit.example)$importance

vars_imp <- as.data.frame(vars_imp)

vars_imp$myvar <- rownames(vars_imp)

vars_imp <- vars_imp[order(vars_imp$Overall,decreasing = TRUE),]

vars_imp <- vars_imp[1:10,]

colnames(vars_imp) <- c('Importance','myvar')

library(ggpubr) # aunque ya estaba cargada al principio.
ggbarplot(vars_imp,
          x = "myvar", y = "Importance",
          #fill  = 'myvar',
          color = "blue",             # Set bar border colors to white
          palette = "jco",            # jco journal color palett. see ?ggpar
          sort.val = "asc",          # Sort the value in descending order
          sort.by.groups = FALSE,     # Don't sort inside each group
          x.text.angle = 90,          # Rotate vertically x axis texts
          ylab = "Importancia",
          xlab = 'Variable',
          #legend.title = "MPG Group",
          rotate = TRUE,
          ggtheme = theme_minimal(),
          main = " Variables más importantes para rf"
          )
```



Como se puede ver en el barplot, hay varias variables que coinciden con rl.5 (deny ~ chist.1 + lvrat + chist.2 + insurance.yes + phist.no) pero otras no, las cuales podría aportar información extra a rl.5 para mejorar el performance, a expensas de complicar el modelo. También podrían ser evaluadas, por ejemplo, como conjunto aparte de rl.5 las cuatro primeras variables (insurance.yes, insurance.no, pirat y lvrat) al haber sido claramente las que más han participado en la división de los nodos. En ese caso, se quitaría insurance.no al ser redundante con insurance.yes.

4.6.1 Matriz de confusión, sensitvidad, especifidad y precisión

La precisión de la regresión logística es 0.9105 , algo superior que la accuracy base del modelo nulo de 0.891. La sensitividad es 0.26655 mientras que la especifidad es 0.98915 Se recuerda que la sensitividad es la capacidad detectar Yes para aquellas obersvaciones Yes, mientras que la especificidad es la capacidad de detectar No aquellas observaciones No.

Por tanto, bajo el punto de corte actual de 0.5, el modelo detecta mejor los No que los Yes. Esto es esperable ya que la clase dominante es No y ha tenido más oportunidades para aprender cuando se da esa categoría.

La elección del punto de corte depende del objetivo del problema. En esta caso, el goal es, como asesoría financiera, detectar qué clientes podrían tener problemas a la hora de concederles una hipoteca para concentrar la campaña publicitaria. Por tanto, se trata de un problema de clasificación dura que requiere de una sensitividad lo más alta posible intentando no perjudicar demasiado la especificidad para no gastar muchos recursos para no potenciales clientes. Es decir, hay que bajar el punto de corte para recategorizar más observaciones como Yes.

```
vardep<-"deny"
listconti<- c("chist.1", "lvrat" , "chist.2" , "insurance.yes" , "phist.no" )
listclass<-c("")
```

```
grupos<- n.folds
sinicio<-2234
repe<-50


# ********************************
# CRUZADA LOGISTICA
# ********************************

cruzadalogistica_mb <- function(data=data,vardep=NULL,
 listconti=NULL,listclass=NULL,grupos=4,sinicio=1234,repe=5)
{

  if (any(listclass==c(""))==FALSE)
  {
   for (i in 1:dim(array(listclass))) {
    numindi<-which(names(data)==listclass[[i]])
    data[,numindi]<-as.character(data[,numindi])
    data[,numindi]<-as.factor(data[,numindi])
   }
  }

  data[,vardep]<-as.factor(data[,vardep])

  # Creo la formula para la logistica

if (any(listclass==c(""))==FALSE)
  {
   koko<-c(listconti,listclass)
  }  else   {
   koko<-c(listconti)
  }

  modelo<-paste(koko,sep="",collapse="+")
  formu<-formula(paste(vardep,"~",modelo,sep=""))

  formu
  # Preparo caret

  set.seed(sinicio)
  control<-trainControl(method = "repeatedcv",number=grupos,repeats=repe,
   savePredictions = "final",classProbs=TRUE)

  # Aplico caret y construyo modelo

  regresion <- train(formu,data=data,
   trControl=control,method="glm",family = binomial(link="logit"))

 return(regresion)

}
```

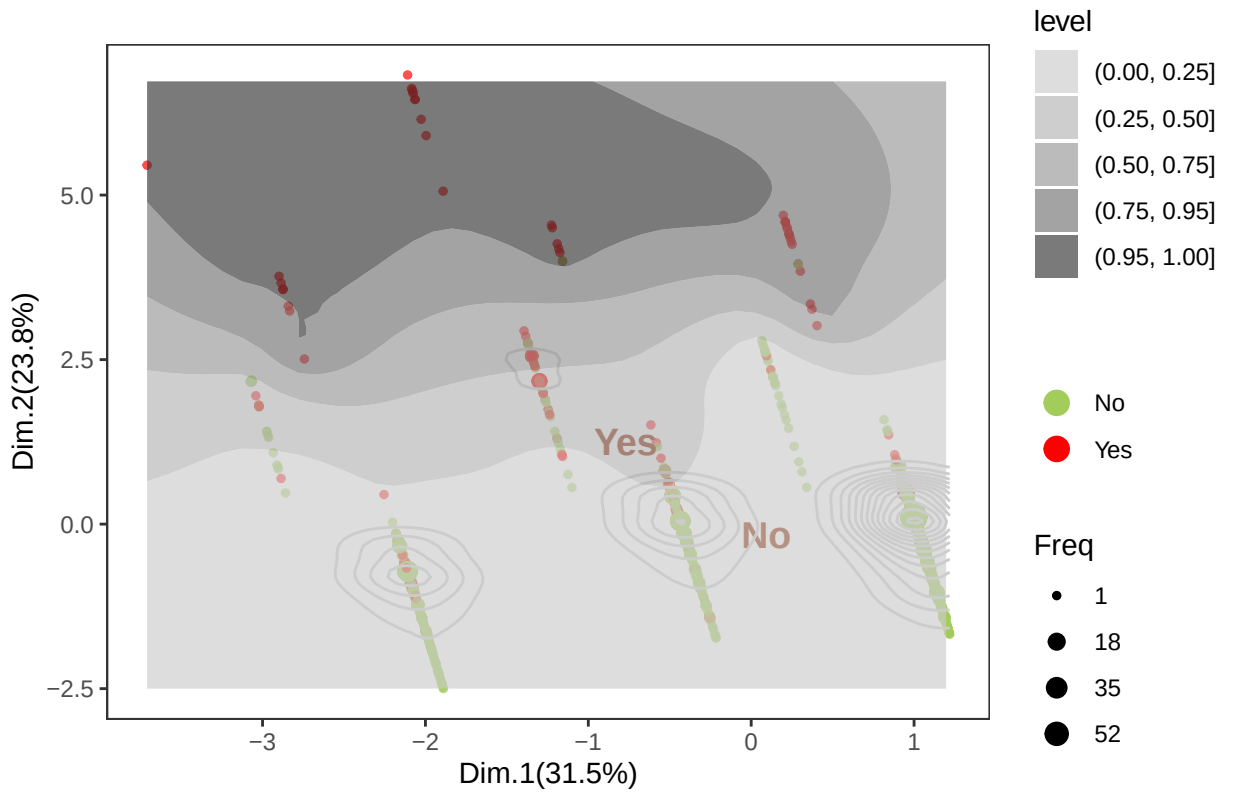
```
logit.final<-cruzadalogistica_mb(data=datMod1,
                          vardep=vardep,listconti=listconti,
                          listclass=listclass,grupos=grupos,sinicio=sinicio,repe=repe)
```

```
sal<-logit.final$pred

# MEDIDAS CON PUNTO DE CORTE 0.5 (valor por defecto)

confusionMatrix(reference=sal$obs,data=sal$pred, positive="Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    No   Yes
##        No  95156  8618
##        Yes  1044  3132
##
##                Accuracy : 0.9105
##                  95% CI : (0.9088, 0.9122)
##     No Information Rate : 0.8912
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3566
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.26655
##             Specificity : 0.98915
##          Pos Pred Value : 0.75000
##          Neg Pred Value : 0.91695
##              Prevalence : 0.10885
##          Detection Rate : 0.02901
##    Detection Prevalence : 0.03868
##       Balanced Accuracy : 0.62785
##
##        'Positive' Class : Yes
##
```

En primer lugar se muestra un mapa de contorno para las dos dimensiones que aportan más varianza. Los tonos reflejan la probabilidad conferida, la clase minoriatria (Yes) aparece en rojo. Se observa que muchos puntos rojos caen en el segundo gris más claro, con lo que se decide bajar el punto de corte al límite inferior de este contorno 0.25

```
library(visualpred)

listconti <- c("chist.1" , "lvrat" , "chist.2" , "insurance.yes" , "phist.no")
listclass <- c()

result.vp <- famdcontour(dataf = datMod1, listconti = listconti, listclass = listclass, vardep = vardep



result.vp[[2]]
```



Se observa que con corte = 0.25 la sensitividad pasa de 0.27 a 0.41 con valores aún altos para la especificad (0.96) y accuracy, 0.90. El punto de corte 0.25 se ajusta más al objetivo del docuento que el valor por defecto de 0.5.

Un valor de 0.15 también podría ser interesante al dar una sensitividad mayor del 60% manteniendo accuracy y especificidad por encima del 80%. En este caso, convendría presentar varios valores de cut-off points al departamento de márketing para coordinar con ellos los recursos disponibles para fijar un valor óptimo.

```
corte<-0.25

sal$predcorte<-ifelse(sal$Yes>corte,"Yes","No")
sal$predcorte<-as.factor(sal$predcorte)

confusionMatrix(reference=sal$obs,data=sal$predcorte, positive="Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    No   Yes
##        No  92656  6892
##        Yes  3544  4858
##
##                Accuracy : 0.9033
##                  95% CI : (0.9015, 0.9051)
```

```
##     No Information Rate : 0.8912
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4304
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.41345
##             Specificity : 0.96316
##          Pos Pred Value : 0.57820
##          Neg Pred Value : 0.93077
##              Prevalence : 0.10885
##          Detection Rate : 0.04500
##    Detection Prevalence : 0.07783
##       Balanced Accuracy : 0.68830
##
##        'Positive' Class : Yes
##
```

```
corte<-0.15

sal$predcorte<-ifelse(sal$Yes>corte,"Yes","No")
sal$predcorte<-as.factor(sal$predcorte)

confusionMatrix(reference=sal$obs,data=sal$predcorte, positive="Yes")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    No   Yes
##        No  81707  4472
##        Yes 14493  7278
##
##                Accuracy : 0.8243
##                  95% CI : (0.822, 0.8266)
##     No Information Rate : 0.8912
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3411
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.61940
##             Specificity : 0.84935
##          Pos Pred Value : 0.33430
##          Neg Pred Value : 0.94811
##              Prevalence : 0.10885
##          Detection Rate : 0.06742
##    Detection Prevalence : 0.20168
##       Balanced Accuracy : 0.73437
##
##        'Positive' Class : Yes
##
```

4.6.2 Descripción del modelo final

Un resumen de la regresión logística se muestra a continuación.

```
summary(logit.final)
```

```
##
## Call:
## NULL
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6642  -0.4383  -0.2858  -0.2174   3.1352
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -0.32275    0.19123  -1.688  0.09147 .
## chist.1       -1.66031    0.19403  -8.557  < 2e-16 ***
## lvrat          0.51349    0.09558   5.372 7.77e-08 ***
## chist.2       -0.66729    0.20926  -3.189  0.00143 **
## insurance.yes  5.08461    0.63363   8.025 1.02e-15 ***
## phist.no      -1.44207    0.20777  -6.941 3.90e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1485.8  on 2158  degrees of freedom
## Residual deviance: 1111.4  on 2153  degrees of freedom
## AIC: 1123.4
##
## Number of Fisher Scoring iterations: 6
```

Todas las 5 variables son significativas, siendo la menos chist.2. Los coeficientes negativos indican odds-ratios (sus exponenciales) inferiores a 1. A continuación se muestran dos ejemplos, variable numérica y categórica, sobre cómo interpretar estos resultados.

Variable input cuantitativa. La variable objetivo escogida "Se le deniega al sujeto la hipoteca" presenta un incremento de las posibilidades en un 67% en el caso de un incremento unitario del ratio prestamo/precio hipoteca, lo cual tiene sentido ya que aumentea el apalancamiento.Esto es debido a que el odds-ratio vale 1.67.

Variable input cualitativa. La variable input insurance significa si se le denegó al sujeto seguro para la hipoteca. Los sujetos a los que se les negó dicho seguro presentan 161.52 veces más de posibilidades de denegarles la hipoteca frente a los que sí le concedieron el seguro, al ser el odds-ratio del insurance.yes 161.52

```
cat('Odds-ratios de los parámetros :\n')
```

```
## Odds-ratios de los parámetros :
```

```
exp(logit.final$finalModel$coefficients)
```

```
##   (Intercept)       chist.1         lvrat       chist.2 insurance.yes
##     0.7241551     0.1900806     1.6711124     0.5130984   161.5176825
##      phist.no
##     0.2364385
```

El conjunto datMod presentaba 235 valores de la clase minoritaria, esto da una ratio de $235/6 \sim 40$ parámetros por coeficiente. Tomando como aproximación la referencia de los ratios para redes, 40 obs/parámetro no tendría que presentar problemas serios de sobreajuste en principio. Esto apoya a la intuición (no robusta) de que no tendría que haber problemas serios de overfitting al solor contar con 5 variables input.