Machine Learning

Miguel Ángel Castaño Ibáñez

5/17/2021

Librerías

Cargo todas las librerías utlilizadas para para este ejercicio

```
# LIBRERIES

suppressPackageStartupMessages({
# install
# install.packages("reshape")
# install.packages('doParallel')

# load
library(ggplot2)
library(inspectdf) # EDAs automaticos
library(plotly)
library(dummies)
library(MASS)
library(caret)
library(plyr)
library(reshape)
library(randomForest)
library(tinytex)
library(doParallel)
library(evaluate)
})
```

Ejercicios

A continuacion, los ejercicios propuestos a resolver en este modulo

a) Se deben realizar pruebas suficientes para obtener una buena selección de variables, obteniendo uno o varios conjuntos de variables tentativos

b) Se requiere la comparación entre los mejores algoritmos y regresión logística

c) Se comprobará el efecto de la variación de los parámetros básicos de cada algoritmo (tuneado) (número de nodos en redes, shrink en gradient boosting, etc.).

d) Los algoritmos a utilizar son obligatoriamente y como mínimo:

- Redes Neuronales
- Regresión Logística
- Bagging
- Random Forest
- Gradient Boosting
- Support Vector Machines
- También si se quiere y para comprender los datos se puede probar con un simple árbol pero no es obligatorio.

e) Es necesario utilizar validación cruzada, validación cruzada repetida o como mínimo training/test repetido.

f) Es necesario hacer alguna prueba de ensamblado.

Lectura ficheros

Nuestro dataset contiene una muestra de 5000 pacientes de diferentes edades, donde podemos observar quien de ellos a sufrido un ictus. Este dataset presenta 12 variables, 11 inputs y una dependiente objetivo binaria.

Fuente: *https://www.kaggle.com/fedesoriano/stroke-prediction-dataset*

Cargamos los ficheros donde están el conjunto de entrenamiento y el de test, ademas de tener un dataset solo el id y la variable objetivo

```
data <- read.csv("./healthcare-dataset-stroke-data.csv")
```

Analisís exploratorio (EDA)

Antes de empezar a crear los modelos vamos a hacer un analisis exploratorio de nuestra variable por si fuera necesario, imputar valores o cambiar el tipo de alguna de estas.

```
#compruebo los tipos
str(data)
```

```
## 'data.frame':    5110 obs. of  12 variables:
##  $ id               : int  9046 51676 31112 60182 1665 56669 53882 10434 27419 60491 ...
##  $ gender           : chr  "Male" "Female" "Male" "Female" ...
##  $ age              : num  67 61 80 49 79 81 74 69 59 78 ...
##  $ hypertension     : int  0 0 0 0 1 0 1 0 0 0 ...
##  $ heart_disease    : int  1 0 1 0 0 0 1 0 0 0 ...
##  $ ever_married     : chr  "Yes" "Yes" "Yes" "Yes" ...
##  $ work_type        : chr  "Private" "Self-employed" "Private" "Private" ...
##  $ Residence_type   : chr  "Urban" "Rural" "Rural" "Urban" ...
##  $ avg_glucose_level: num  229 202 106 171 174 ...
##  $ bmi              : chr  "36.6" "N/A" "32.5" "34.4" ...
##  $ smoking_status   : chr  "formerly smoked" "never smoked" "never smoked" "smokes" ...
##  $ stroke           : int  1 1 1 1 1 1 1 1 1 1 ...
```

Tras los resultados vistos en el apartado anterior podemos concluir a llevar a factor aquellas variables que consideramos categoricas, y transformar a "Yes/No", nuestra variable objetivo binaria.

```
# transfromamos en yes or no la variable obj
data$stroke<-ifelse(data$stroke==1,"Yes","No")

# convert to factor
data[,c(2,4,5,6,7,8,11,12)] <- lapply(data[,c(2,4,5,6,7,8,11,12)], factor)

# convert to nummeric
data$bmi <- as.numeric(data$bmi)

# comprobar tipos
str(data)
```

```
## 'data.frame':    5110 obs. of  12 variables:
##  $ id               : int  9046 51676 31112 60182 1665 56669 53882 10434 27419 60491 ...
##  $ gender           : Factor w/ 3 levels "Female","Male",..: 2 1 2 1 1 2 2 1 1 1 ...
##  $ age              : num  67 61 80 49 79 81 74 69 59 78 ...
##  $ hypertension     : Factor w/ 2 levels "0","1": 1 1 1 1 2 1 2 1 1 1 ...
##  $ heart_disease    : Factor w/ 2 levels "0","1": 2 1 2 1 1 1 2 1 1 1 ...
##  $ ever_married     : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 1 2 2 ...
##  $ work_type        : Factor w/ 5 levels "children","Govt_job",..: 4 5 4 4 5 4 4 4 4 4 ...
##  $ Residence_type   : Factor w/ 2 levels "Rural","Urban": 2 1 1 2 1 2 1 2 1 2 ...
##  $ avg_glucose_level: num  229 202 106 171 174 ...
##  $ bmi              : num  36.6 NA 32.5 34.4 24 29 27.4 22.8 NA 24.2 ...
##  $ smoking_status   : Factor w/ 4 levels "formerly smoked",..: 1 2 2 3 2 1 2 2 4 4 ...
##  $ stroke           : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 2 2 2 2 2 ...
```

```
# Comprobar observaciones de la var objetivo
length(filter(data, stroke == "Yes")[,1])
```

```
## [1] 249
```

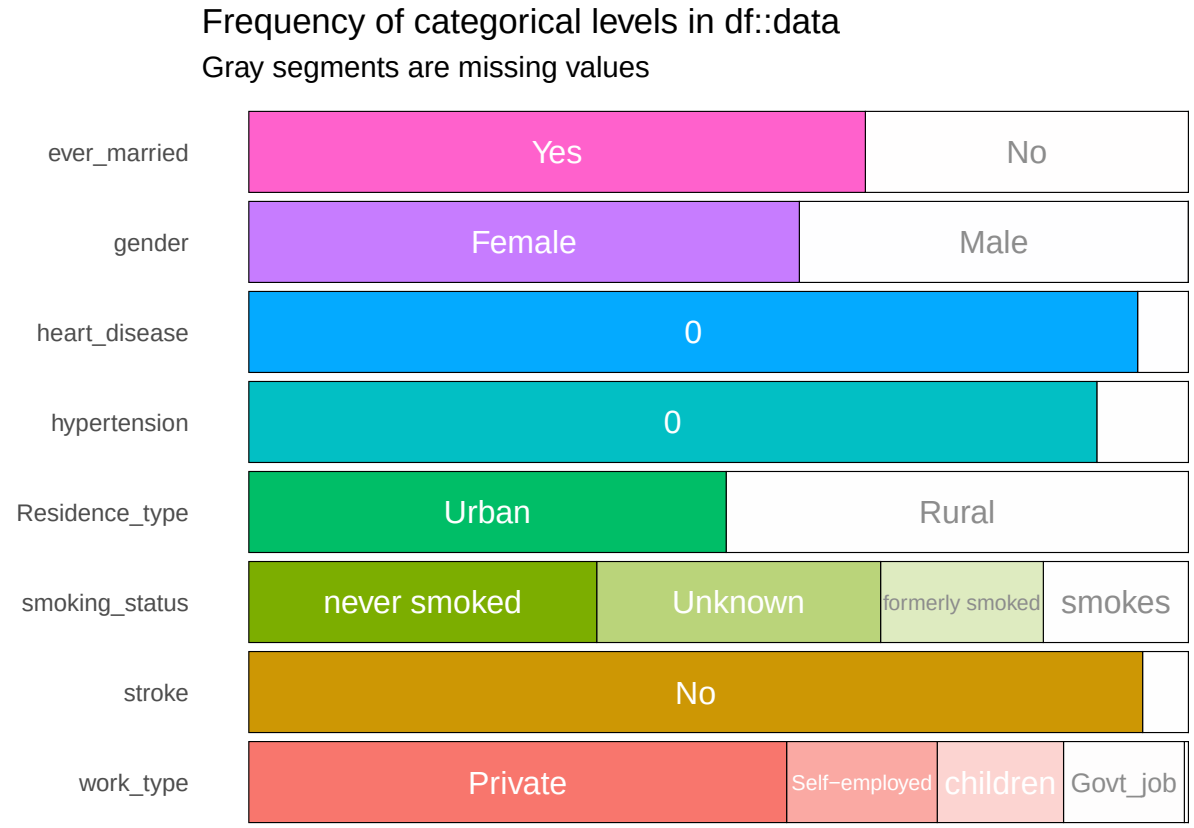
```
length(filter(data, stroke == "No")[,1])
```

```
## [1] 4861
```

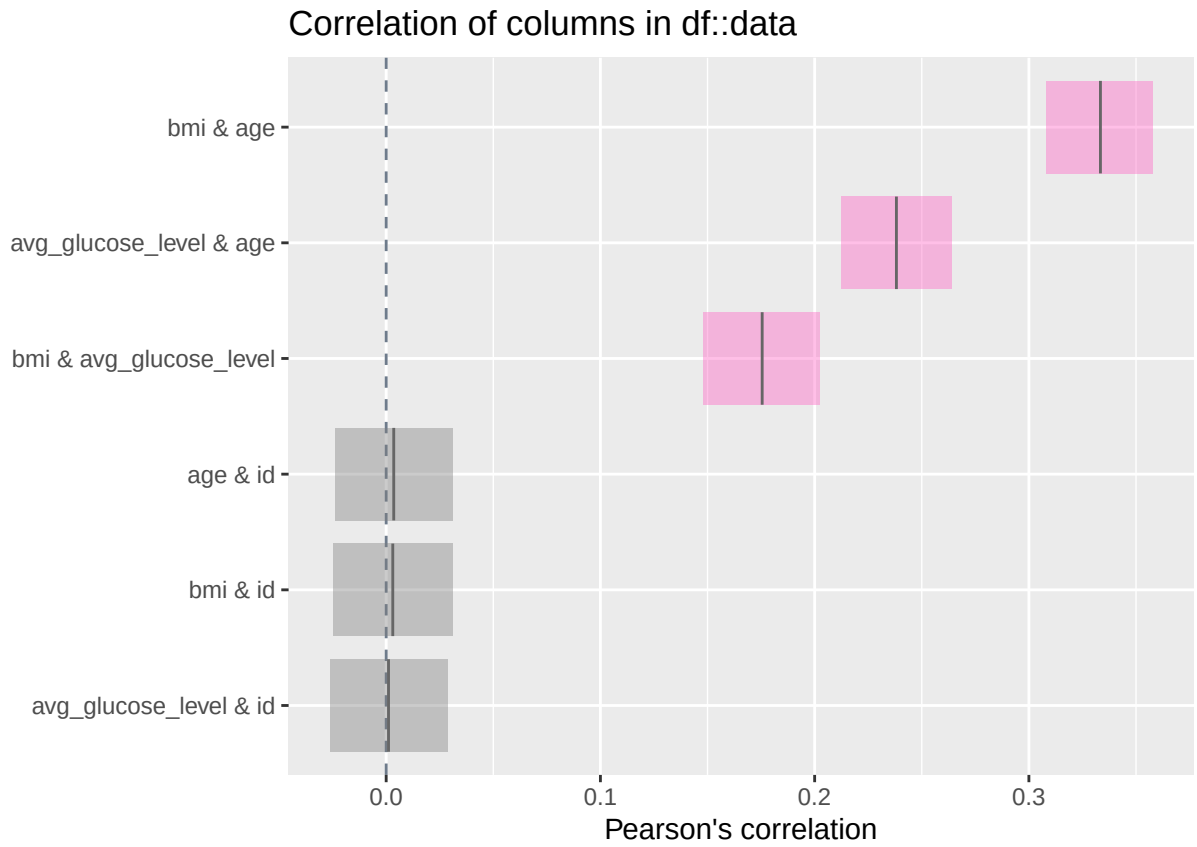
El numero de casos que si presentan ataque cerebral es muy pequeño respecto al numero de los que no, y esto prodria llevarnos a conclusiones sesgadas, pese a tener un accuracy bastante, por ello debemos ir con cuidado. Tras la eleccion del modelo seria conveniente comprobar este modelos con un dataset mas grande de datos.

En segundo lugar vamos a comprobar las variables categoricas que hay, la correlacion de variable por si fuera necesario quitar alguna de estas y el numero de NAs.

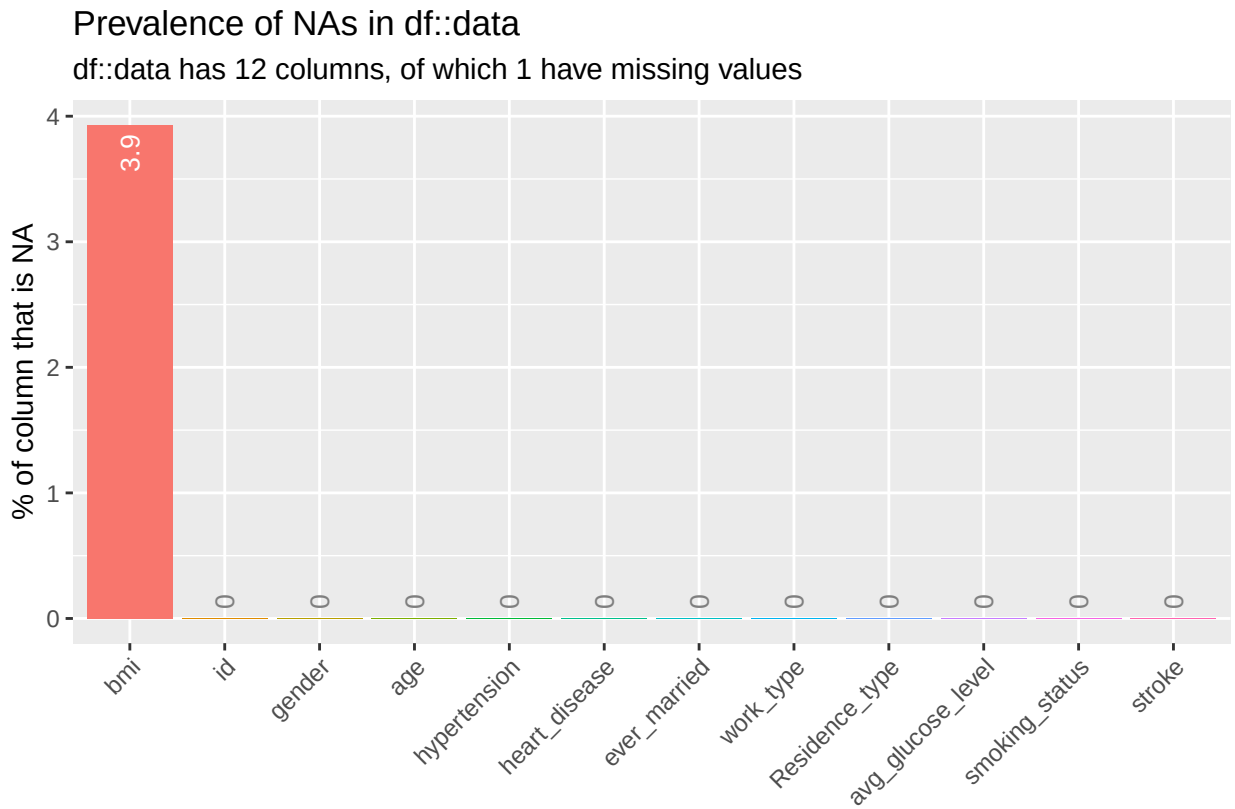
```
# Horizontal bar plot for categorical column composition
x <- inspect_cat(data)
show_plot(x)
```




```
# Correlation betwee numeric columns + confidence intervals
x <- inspect_cor(data)
show_plot(x)
```




```
# Occurence of NAs in each column ranked in descending order
x <- inspect_na(data)
show_plot(x)
```



Feature engineering

Nuestro primer paso para a aplicar feature engineering a nuestras variables es separarlas en continuas, categoricas y objetivo. La variable id no tiene relevancia en este dataset por ello he decidido dejarla fuera del analisis.

```
# lista de variables
# dput(names(data))
list_int <- c("id")
list_cont <- c("avg_glucose_level", "bmi", "age")
list_cat <- c("gender", "hypertension", "heart_disease", "ever_married",
              "work_type", "Residence_type", "smoking_status")
var_dep <- c("stroke")
col_var_dep<-data[,var_dep]
```

Estandarización de variables continuas

Estandarizamos las variables continuas con valores desde 0 hasta 1

```
# calc estandarizacion
means <-apply(data[,list_cont],2,mean,na.rm=TRUE)
sds<-sapply(data[,list_cont],sd,na.rm=TRUE)
```

```r
# var continuas estandarizadas
stroke_data <- scale(data[,list_cont], center = means, scale = sds)
# stroke_data <- data.frame(cbind(stroke_data,col_var_dep))

# union continuas y categoricas
index_cont<-which(colnames(data)%in%list_cont) # index cont
stroke_data<-cbind(stroke_data,data[,-index_cont]) # join
```

Eliminacion de missings

En lugar de imputar las observaciones missing he considerado eliminarlas del dataset a estudiar

```r
stroke_data<-na.omit(stroke_data,(!is.na(stroke_data)))

# comprobamos suficientes observaciones YES y NO
length(filter(stroke_data, stroke == "Yes")[,1])
```

```
## [1] 209
```

```r
length(filter(stroke_data, stroke == "No")[,1])
```

```
## [1] 4700
```

Dummies

Vamos a aplicar dummies a las variables categoricas, tratandolas de convertir a binarias mediante metodologia one-hot, es decir valor 1 si se cumple la observacion para esa variable categoriaca y 0 cuando no.

```r
stroke_data<- dummy.data.frame(stroke_data, list_cat, sep = ".")

#eliminamos los dummies pocos representados
stroke_data$work_type.Never_worked <- NULL
```

He decidico eliminar las dummi work_type.Never_worked, ya que apenas hay observaciones para esta variable.

Por ultimo para evitar futuros conflictos aplico la siguiente funcion, para cambiar el nombre de las columnas que puedan tener palabras reservadas.

```r
# Make Valid Column Names
colnames(stroke_data) <- make.names(colnames(stroke_data))
```

Seleccion de variables

En este paso coincidiendo con el apartado a) vamos a tratar de obtener las variables mas relevantes para generar modelos en nuestro dataset.

En primer lugar vamos a aplicar el algoritmo stepwise backward and fordwar y con criterio tonto AIC como BIC para ver cuales son las variables que mas importancia tienen en la regresion logistica.

```r
# Selección de variables por el metodo stepAIC
full<-glm(factor(stroke)~.,data=stroke_data,family = binomial(link="logit"))
null<-glm(factor(stroke)~1,data=stroke_data,family = binomial(link="logit"))

# aplicamos stepAIC
seleccionAIC<-stepAIC(null,scope=list(upper=full),direction="both")
seleccionBIC<-stepAIC(null,scope=list(upper=full),direction="both")
```

Obtenemos la importancia de las variables calculadas para nuestro modelo mediante estos metodos

```r
# AIC
seleccionAIC
```

```
##
## Call:  glm(formula = factor(stroke) ~ age + avg_glucose_level + hypertension.0 +
##     work_type.Self.employed + smoking_status.smokes + heart_disease.0,
##     family = binomial(link = "logit"), data = stroke_data)
##
## Coefficients:
##             (Intercept)                      age        avg_glucose_level
##                 -3.3850                   1.6360                   0.2174
##          hypertension.0  work_type.Self.employed    smoking_status.smokes
##                 -0.5507                  -0.3907                   0.3932
##         heart_disease.0
##                 -0.3577
##
## Degrees of Freedom: 4908 Total (i.e. Null);  4902 Residual
## Null Deviance:       1728
## Residual Deviance: 1366  AIC: 1380
```

```r
# BIC
seleccionBIC
```

```
##
## Call:  glm(formula = factor(stroke) ~ age + avg_glucose_level + hypertension.0 +
##     work_type.Self.employed + smoking_status.smokes + heart_disease.0,
##     family = binomial(link = "logit"), data = stroke_data)
##
## Coefficients:
##             (Intercept)                      age        avg_glucose_level
##                 -3.3850                   1.6360                   0.2174
##          hypertension.0  work_type.Self.employed    smoking_status.smokes
##                 -0.5507                  -0.3907                   0.3932
##         heart_disease.0
##                 -0.3577
##
## Degrees of Freedom: 4908 Total (i.e. Null);  4902 Residual
## Null Deviance:       1728
## Residual Deviance: 1366  AIC: 1380
```

Otra forma de poder seleccionar estas variables es repitiendo el proceso un numero determinado de veces, obteniendo varios modelos y observar las variables mas frecuentes en estos.

```r
source("funcion steprepetido binaria.R")

list_var <- names(stroke_data[,-24]) # dput

# AIC
listaAIC<-steprepetidobinaria(data=stroke_data,
                           vardep=var_dep,listconti=list_var,
                           sinicio=12340,
                           sfinal=12390,porcen=0.8,criterio="AIC")

tablaAIC <- listaAIC[[1]]

# BIC
listaBIC <- steprepetidobinaria(data=stroke_data,
                              vardep=var_dep,listconti=list_var,
                              sinicio=12340,
                              sfinal=12390,porcen=0.8,criterio="BIC")

tablaBIC <- listaBIC[[1]]
```

Obtenemos la seleccion de variables más repetidas y su frecuencia

```r
# table freq
head(tablaAIC,5)
```

```
##                                                                                                 model
## 1  age+avg_glucose_level+hypertension.0+work_type.Self.employed+smoking_status.smokes+heart_disease.
## 12       age+avg_glucose_level+hypertension.0+smoking_status.smokes+heart_disease.0+work_type.Privat
## 18                       age+avg_glucose_level+hypertension.0+work_type.Self.employed+heart_disease.
## 23                 age+avg_glucose_level+hypertension.0+work_type.Self.employed+smoking_status.smoke
## 28                       age+avg_glucose_level+hypertension.0+smoking_status.smokes+work_type.Privat
##    Freq contador
## 1    11        6
## 12    6        6
## 18    5        5
## 23    5        5
## 28    4        5
```

```r
head(tablaBIC,5)
```

```
##                                                        modelo Freq contador
## 1                                     age+avg_glucose_level   31        2
## 32                     age+avg_glucose_level+hypertension.0   13        3
## 45                    age+avg_glucose_level+heart_disease.0    1        3
## 46 age+avg_glucose_level+hypertension.0+work_type.Self.employed    1        4
## 47                     age+avg_glucose_level+hypertension.1    1        3
```

Tras los calculos anteriores he decidido quedarme con las variables obtenidas en metodo stepwise repetido con criterio AIC, ya que parece menos aqgresivo descartando variables. En concreto he decidido quedarme con las siguiente seleccion variables mostradas a continucion, ya que de 51 veces que repetimos el algoritmo, la ganadaora obtuvo una frecuencia de 11 veces frente al resto. Por ejemplo la 2ª seleccion fue repetida tan solo 6 veces.

```
# c("age", "avg_glucose_level", "hypertension.0", "work_type.Self.employed",
#   "smoking_status.smokes", "heart_disease.0")

# factor(stroke) ~ age + avg_glucose_level + hypertension.0 +
#   work_type.Self.employed + smoking_status.smokes + heart_disease.0
```

Modelos predictivos

En este apartado vamos a crear todos los modelos y realizarles un "tuneado" utilizand ola libreira caret. Asi, conseguiremos escoger los parametros adecuados obteniendo como resultado el mejor modelo, para en apartados posteriores poder hacer un analisis de sesgo-varianza. En este apartado vamos a cumplir con los apartados c), d) y e).

Para este proceso he decidido solo utilizar validacion cruzada, para poder agilizar tiempo en el calculo de los modelos. Aun con ello esto podria llevar a un sobreajuste de los algoritmos. En apartado del analisis sesgo-varianza utilizare la técnica de validación cruzada repetida, una vez ya tengamos decididos los parametros tras el "tuneo". Vamos a aplicar validacion cruzada sin repetir esto aligerara los calculos pero hay que tener en cuenta que perderemos control sobre la varianza.s

Es posible que algunos resultados no coincidan con respecto a las pruebas de integracion del codigo, debido a utilizar doParallel, ya que esto puede causar que las semillas no funciona como se espera y por tanto pueda cambiar un poco nuestros resultados

REGRESION LOGÍSTICA

Empezamos con la regresión logísstica, vamos a utilizar la validacion cruzada 4 grupos, para asegurarnos de tener un numero sifuiciente de observaciones "Yes/No" en cada grupo test.

En este apartado el único tuneo posible son la semilla y los grupos para la "cross validation". He seleccionado los que se muestran a continuación, ya que son lo que mejor resultados me han dado en las pruebas de integración.

```
set.seed(1234967)

control<-trainControl(method = "cv",
                      number=4,savePredictions = "all") # repeats=5

reg_log <- train(factor(stroke) ~ age + avg_glucose_level + hypertension.0
                 + work_type.Self.employed + smoking_status.smokes
                 + heart_disease.0,
                 data=stroke_data,method="glm",trControl=control)

reg_log
```

```
## Generalized Linear Model
##
## 4909 samples
##    6 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
```

```
## Summary of sample sizes: 3682, 3682, 3682, 3681
## Resampling results:
##
##   Accuracy   Kappa
##   0.9578328  0.01809725
```

RED

Usamos el paquete "avNNet" para utilizar 5 redes y hacer por lo tanto un promedio de estas para obtener nuestros resultados, evitando sesgar nuestros datos

```
control<-trainControl(method = "cv",
                      number=4,savePredictions = "all")

nnetgrid <- expand.grid(size=c(5,10,15,20),decay=c(0.01,0.1,0.001),bag=F)

avnnet <- train(factor(stroke) ~ age + avg_glucose_level + hypertension.0
                + work_type.Self.employed + smoking_status.smokes
                + heart_disease.0,
                data=stroke_data, method="avNNet",linout = TRUE,maxit=100,
                trControl=control,repeats=5, tuneGrid=nnetgrid)
```

Mostramos a continuación el resultado y llegamos a la conclusión de que los mejores parametros para nuestra red es decay=0.1 y un tamaño de la red de 5 nodos ya que obtiene el mismo accuracy, obteniendo un modelo mucho mas sencillo

```
avnnet
```

```
## Model Averaged Neural Network
##
## 4909 samples
##    6 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 3681, 3682, 3682, 3682
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##    5    0.001  0.9572215  0.03216986
##    5    0.010  0.9570178  0.02369868
##    5    0.100  0.9574251  0.01684577
##   10    0.001  0.9568140  0.04800243
##   10    0.010  0.9564065  0.03077202
##   10    0.100  0.9572213  0.01644532
##   15    0.001  0.9572213  0.04122371
##   15    0.010  0.9574251  0.04132249
##   15    0.100  0.9576287  0.02542383
##   20    0.001  0.9566103  0.03115885
##   20    0.010  0.9568140  0.04789108
##   20    0.100  0.9576287  0.02542383
```

```
##
## Tuning parameter 'bag' was held constant at a value of FALSE
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 15, decay = 0.1 and bag = FALSE.
```

ÁRBOL

Este metodo no vamos a profundizar mucho ya que posteriormente tenemos otros modelos compuestos por banggin, random forest, gradient boosting, etc.

```
control<-trainControl(method = "cv",number=4,savePredictions = "all")

arbolgrid <-  expand.grid(cp=c(0,0.001,0.01,0.1))

arbol <- train(factor(stroke) ~ age + avg_glucose_level + hypertension.0
               + work_type.Self.employed + smoking_status.smokes
               + heart_disease.0,
               data=stroke_data, method="rpart",minbucket=30,
               trControl=control,tuneGrid=arbolgrid)

arbol
```

```
## CART
##
## 4909 samples
##    6 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 3682, 3682, 3681, 3682
## Resampling results across tuning parameters:
##
##   cp     Accuracy   Kappa
##   0.000  0.9553879   0.0786938174
##   0.001  0.9553879   0.0786938174
##   0.010  0.9572215  -0.0004004497
##   0.100  0.9574253   0.0000000000
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.1.
```

Podemos observar que el parametro "cp" para este modelo es 0.1 y minbucket = 30.

BANGGIN

El siguiente modelo es un modelo random forest, el más común de los arboles, pero con la peculiaridad de que mtry = 6 (numero de variables del modelo). Mantenemos la misma semilla, ya que es la que mejor resultado aporta para crear el modelo y 4 grupos en la validacion cruzada. Y los parametros mas optimos tras el tuneado son ntree=1000 y nodesize=10.

```
rfgrid<-expand.grid(mtry=c(6))

control<-trainControl(method = "cv",number=4,savePredictions = "all",
                      classProbs=TRUE)

bangging <-  train(factor(stroke) ~ age + avg_glucose_level + hypertension.0
                   + work_type.Self.employed + smoking_status.smokes
                   + heart_disease.0, data=stroke_data,method="rf",
                   trControl=control,tuneGrid=rfgrid,linout = FALSE,
                   ntree=1000,nodesize=10,replace=TRUE)

bangging
```

```
## Random Forest
##
## 4909 samples
##    6 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 3681, 3682, 3682, 3682
## Resampling results:
##
##   Accuracy   Kappa
##   0.9562033  0.01399572
##
## Tuning parameter 'mtry' was held constant at a value of 6
```

RANDOM FORES

Para este algoritmo el proceso de tuneo, ha sido importante estudiar tanto el mtry, como el numero de arboles generados. Otro parametro para tunear podria ser nodesize ya que nos indica el umero maximo de nodos generados y mara la complejidad de los arboles. Mantenemos la misma semilla para modelo anteriores junto a los 4 grupos para la validacion cruzada.

```
rfgrid<-expand.grid(mtry=c(3,4,5,6))

control<-trainControl(method = "cv",number=4,savePredictions = "all",
                      classProbs=TRUE)

rf <- train(factor(stroke) ~ age + avg_glucose_level + hypertension.0
            + work_type.Self.employed + smoking_status.smokes
            + heart_disease.0, data=stroke_data,method="rf",
            trControl=control,tuneGrid=rfgrid,linout = FALSE,ntree=500,
            nodesize=12,replace=TRUE,importance=TRUE)

rf
```

```
## Random Forest
##
## 4909 samples
```

```
##    6 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 3681, 3682, 3682, 3682
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   3     0.9568143  0.007460533
##   4     0.9566104  0.006770788
##   5     0.9566104  0.015285540
##   6     0.9559992  0.005376954
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 3.
```

Como conclusión podemos ver como nuestro nuestro modelo tiene una mayor tasa de aciertos con 500 tras probar desde 100 hasta 1000 en intervalos de 100. Por otro lado, vamos a maracar un node size de 12 ya que conseguimos una acuraccy mas elevada sin sobreajustar. Por último, mtry = 3, ya que este también nos aporta un número más elevado de aciertos haciendo el bagging para generar el modelo mas simple y con menos variables, solo el 50% de las disponibles.

GRADIENT BOOSTING

Para este modelo construiremos arboles basados en la dirección de decrecimiento dada por el negativo del gradiente, de la función de error. Seguimos utilizando la validacion cruzada en 4 grupos y manteniendo la mism semilla de modelos anteriores.

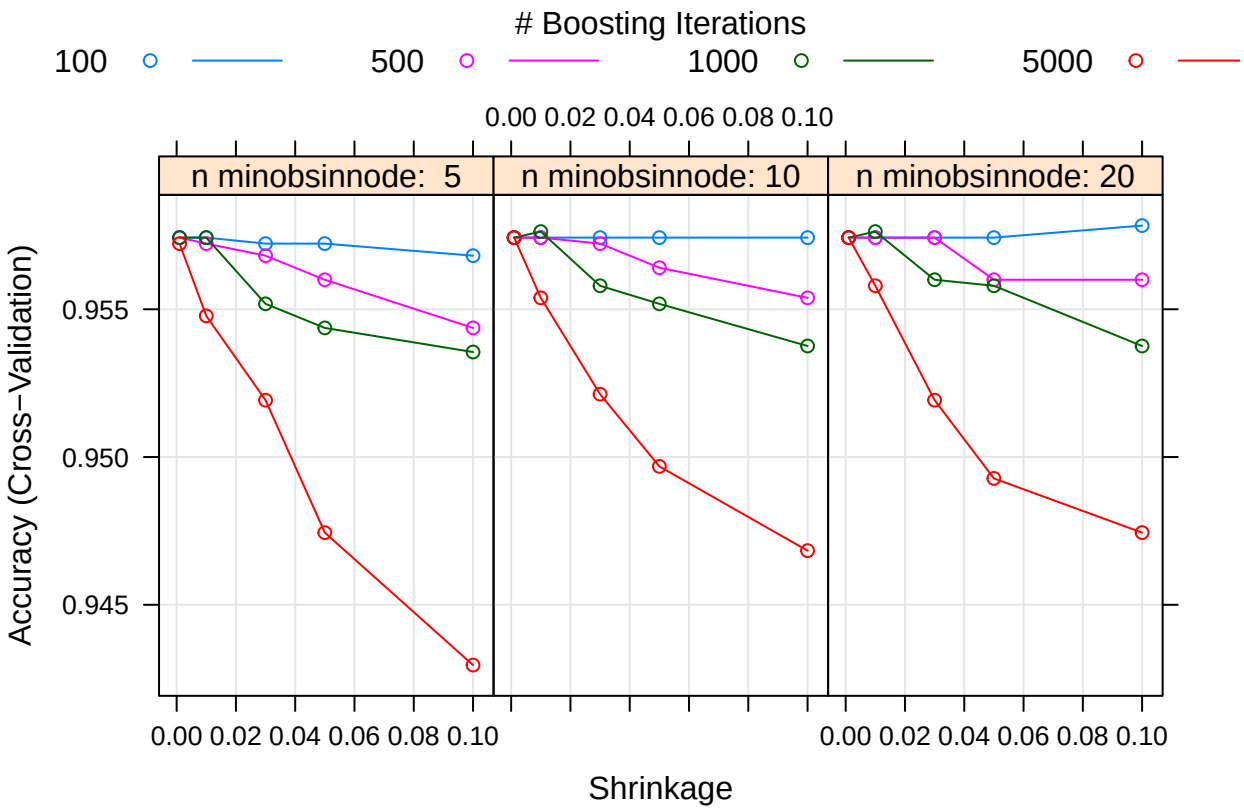
```
gbmgrid<-expand.grid(shrinkage=c(0.1,0.05,0.03,0.01,0.001),
                     n.minobsinnode=c(5,10,20),
                     n.trees=c(100,500,1000,5000),
                     interaction.depth=c(2))

control<-trainControl(method = "cv",number=4,savePredictions = "all",
                     classProbs=TRUE)

gbm <- train(factor(stroke) ~ age + avg_glucose_level + hypertension.0
             + work_type.Self.employed + smoking_status.smokes
             + heart_disease.0,
              data=stroke_data,
              method="gbm",trControl=control,tuneGrid=gbmgrid,
              distribution="bernoulli", bag.fraction=1,verbose=FALSE)

gbm
```

```
## Stochastic Gradient Boosting
##
## 4909 samples
##    6 predictor
##    2 classes: 'No', 'Yes'
##
```

```
## No pre-processing
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 3682, 3681, 3682, 3682
## Resampling results across tuning parameters:
##
##   shrinkage  n.minobsinnode  n.trees  Accuracy   Kappa
##   0.001       5               100     0.9574253   0.0000000000
##   0.001       5               500     0.9574253   0.0000000000
##   0.001       5              1000     0.9574253   0.0000000000
##   0.001       5              5000     0.9572215  -0.0004004497
##   0.001      10               100     0.9574253   0.0000000000
##   0.001      10               500     0.9574253   0.0000000000
##   0.001      10              1000     0.9574253   0.0000000000
##   0.001      10              5000     0.9574253   0.0000000000
##   0.001      20               100     0.9574253   0.0000000000
##   0.001      20               500     0.9574253   0.0000000000
##   0.001      20              1000     0.9574253   0.0000000000
##   0.001      20              5000     0.9574253   0.0000000000
##   0.010       5               100     0.9574253   0.0000000000
##   0.010       5               500     0.9572215  -0.0004004497
##   0.010       5              1000     0.9574253   0.0086481741
##   0.010       5              5000     0.9547765   0.0121484464
##   0.010      10               100     0.9574253   0.0000000000
##   0.010      10               500     0.9574253   0.0000000000
##   0.010      10              1000     0.9576290   0.0090486238
##   0.010      10              5000     0.9553878   0.0207722543
##   0.010      20               100     0.9574253   0.0000000000
##   0.010      20               500     0.9574253   0.0000000000
##   0.010      20              1000     0.9576290   0.0090486238
##   0.010      20              5000     0.9557953   0.0140540907
##   0.030       5               100     0.9572215  -0.0004004497
##   0.030       5               500     0.9568138   0.0160497558
##   0.030       5              1000     0.9551840   0.0127378766
##   0.030       5              5000     0.9519244   0.0336386938
##   0.030      10               100     0.9574253   0.0000000000
##   0.030      10               500     0.9572215   0.0165936846
##   0.030      10              1000     0.9557953   0.0138347478
##   0.030      10              5000     0.9521281   0.0283983872
##   0.030      20               100     0.9574253   0.0000000000
##   0.030      20               500     0.9574253   0.0086481741
##   0.030      20              1000     0.9559992   0.0059883895
##   0.030      20              5000     0.9519245   0.0208382366
##   0.050       5               100     0.9572215  -0.0004004497
##   0.050       5               500     0.9559988   0.0145173397
##   0.050       5              1000     0.9543692   0.0111129430
##   0.050       5              5000     0.9474422   0.0447408270
##   0.050      10               100     0.9574253   0.0000000000
##   0.050      10               500     0.9564065   0.0150709015
##   0.050      10              1000     0.9551840   0.0202604747
##   0.050      10              5000     0.9496833   0.0500152361
##   0.050      20               100     0.9574253   0.0000000000
##   0.050      20               500     0.9559992   0.0059883895
##   0.050      20              1000     0.9557953   0.0140540907
##   0.050      20              5000     0.9492761   0.0155480160
```

```
##    0.100         5                 100      0.9568140   0.0075258407
##    0.100         5                 500      0.9543692   0.0111129430
##    0.100         5                1000      0.9535539   0.0453883522
##    0.100         5                5000      0.9429606   0.0477887475
##    0.100        10                 100      0.9574253   0.0086481741
##    0.100        10                 500      0.9553876   0.0283149815
##    0.100        10                1000      0.9537578   0.0249257916
##    0.100        10                5000      0.9468316   0.0680117616
##    0.100        20                 100      0.9578326   0.0179224419
##    0.100        20                 500      0.9559990   0.0143822361
##    0.100        20                1000      0.9537578   0.0179592648
##    0.100        20                5000      0.9474426   0.0453391067
##
## Tuning parameter 'interaction.depth' was held constant at a value of 2
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 100, interaction.depth =
##  2, shrinkage = 0.1 and n.minobsinnode = 20.
```

```
plot(gbm)
```



En el gráfico podemos observar como el numero maximo de nodos no tiene apenas importancia para nuestro modelo. Mayor relecancia adquiere el numero de arboles o iteraciones que se usa para construir el modelo. Por último, el parametro mas importante shrinkage que mide la velocidad de ajuste.

Para nuestro datos tras el tuneo he decidido seleccionar como ganadores para construir mi modelo con un accuracy de 0.9578328 con los siguientes parametros:

shrinkage = 0.015, minobsinnode = 5, trees = 100,

He seleccionado estos, ya que construye un modelo menos complejo para nuestros datos, pese a tener la misma tasa de aciertos que:

shrinkage = 0.100, minobsinnode = 20, trees = 100

XGBOOST

Este modelo se debe a una variante de gradient bosting que trata de reducir el sobreajuste del modelo. Para este modelo los parametos seran los mimso a demas de gamma (constante de regularización). Seguimos utilizando la validacion cruzada en 4 grupos y manteniendo la mism semilla de modelos anteriores.

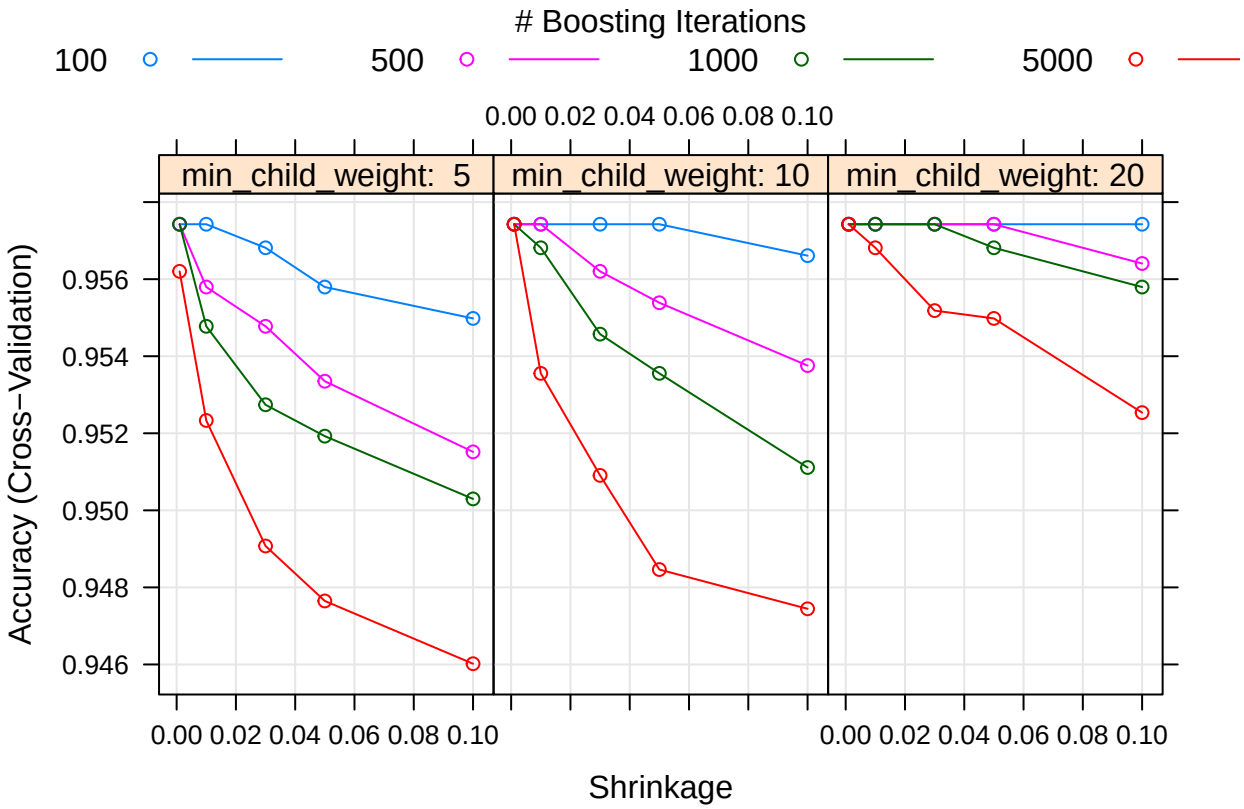
```
xgbmgrid <- expand.grid( min_child_weight=c(5,10,20),
                         eta=c(0.1,0.05,0.03,0.01,0.001),
                         nrounds=c(100,500,1000,5000),
                        max_depth=6,gamma=0,colsample_bytree=1,subsample=1)

control <- trainControl(method = "cv",number=4,savePredictions = "all",
                        classProbs=TRUE)

xgbm <- train(factor(stroke) ~ age + avg_glucose_level + hypertension.0
              +  work_type.Self.employed + smoking_status.smokes
              + heart_disease.0,
              data=stroke_data, method="xgbTree",trControl=control,
              tuneGrid=xgbmgrid,verbose=FALSE)

xgbm
```

```
## eXtreme Gradient Boosting
##
## 4909 samples
##    6 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 3682, 3682, 3682, 3681
## Resampling results across tuning parameters:
##
##   eta    min_child_weight  nrounds  Accuracy   Kappa
##   0.001   5                 100     0.9574253   0.000000000
##   0.001   5                 500     0.9574253   0.000000000
##   0.001   5                1000     0.9574253   0.000000000
##   0.001   5                5000     0.9562028  -0.002348913
##   0.001  10                 100     0.9574253   0.000000000
##   0.001  10                 500     0.9574253   0.000000000
##   0.001  10                1000     0.9574253   0.000000000
##   0.001  10                5000     0.9574253   0.000000000
##   0.001  20                 100     0.9574253   0.000000000
##   0.001  20                 500     0.9574253   0.000000000
##   0.001  20                1000     0.9574253   0.000000000
##   0.001  20                5000     0.9574253   0.000000000
##   0.010   5                 100     0.9574253   0.000000000
```

```
##   0.010   5                500     0.9557954  -0.003123077
##   0.010   5               1000     0.9547769  -0.004993938
##   0.010   5               5000     0.9523327   0.021375629
##   0.010  10                100     0.9574253   0.000000000
##   0.010  10                500     0.9574253   0.000000000
##   0.010  10               1000     0.9568140  -0.001187732
##   0.010  10               5000     0.9535544   0.008482069
##   0.010  20                100     0.9574253   0.000000000
##   0.010  20                500     0.9574253   0.000000000
##   0.010  20               1000     0.9574253   0.000000000
##   0.010  20               5000     0.9568140  -0.001187732
##   0.030   5                100     0.9568140  -0.001187732
##   0.030   5                500     0.9547770   0.011391007
##   0.030   5               1000     0.9527400   0.007055269
##   0.030   5               5000     0.9490742   0.050077835
##   0.030  10                100     0.9574253   0.000000000
##   0.030  10                500     0.9562028  -0.002348913
##   0.030  10               1000     0.9545731   0.010558196
##   0.030  10               5000     0.9509071   0.039957547
##   0.030  20                100     0.9574253   0.000000000
##   0.030  20                500     0.9574253   0.000000000
##   0.030  20               1000     0.9574253   0.000000000
##   0.030  20               5000     0.9551845   0.003750627
##   0.050   5                100     0.9557953  -0.003109644
##   0.050   5                500     0.9533513   0.008556173
##   0.050   5               1000     0.9519255   0.020968302
##   0.050   5               5000     0.9476480   0.052539920
##   0.050  10                100     0.9574253   0.000000000
##   0.050  10                500     0.9553878   0.004175432
##   0.050  10               1000     0.9535544   0.008482069
##   0.050  10               5000     0.9484630   0.048150948
##   0.050  20                100     0.9574253   0.000000000
##   0.050  20                500     0.9574253   0.000000000
##   0.050  20               1000     0.9568140  -0.001187732
##   0.050  20               5000     0.9549808   0.011502861
##   0.100   5                100     0.9549806   0.003627416
##   0.100   5                500     0.9515177   0.033231404
##   0.100   5               1000     0.9502965   0.050737822
##   0.100   5               5000     0.9460188   0.073400125
##   0.100  10                100     0.9566103  -0.001561631
##   0.100  10                500     0.9537581   0.008831972
##   0.100  10               1000     0.9511112   0.032905110
##   0.100  10               5000     0.9474444   0.051718393
##   0.100  20                100     0.9574253   0.000000000
##   0.100  20                500     0.9564065  -0.001948463
##   0.100  20               1000     0.9557956  -0.003136104
##   0.100  20               5000     0.9525363   0.022154629
##
## Tuning parameter 'max_depth' was held constant at a value of 6
## Tuning
##  parameter 'colsample_bytree' was held constant at a value of 1
##
## Tuning parameter 'subsample' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
```

```
## The final values used for the model were nrounds = 100, max_depth = 6, eta
##  = 0.001, gamma = 0, colsample_bytree = 1, min_child_weight = 5 and subsample
##  = 1.
```

```
plot(xgbm)
```



Como la variante anterior podemos observar que el mejor modelo es igualmente el que se obtiene con los mismos parametros, y brindandonos un accuracy de 0.9576290:

eta/shrinkage = 0.015, min_child_weight/minobsinnode = 5, nrounds/trees = 100

Con respecto a otros parametros del modelo como ganma la mejor opcion despues de probarlo es con un valor de 0.

SVM

En este grupo podemos diferenciar tres modelos el SVM basico o lineal, el SVM Polinomial o SVM RBF, dependiendo de los metodos de separacion empleados.

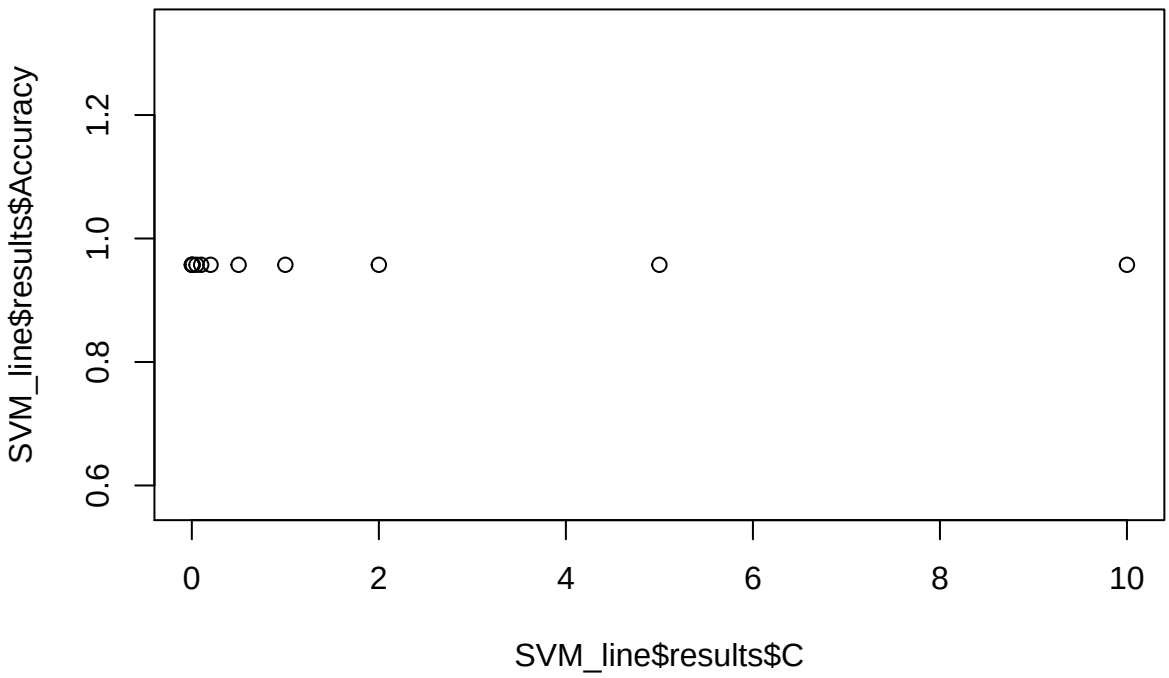
SVM linear

En este metodo conservamos la misma semilla y los mimos grupos para la validacion cruzada. Para este mudelo el tuneo mas a tener en cuenta que modemos hacer es con el parametro C (la constante de regularización)

```r
SVMgrid<-expand.grid(C=c(0.0001,0.001,0.01,0.05,0.1,0.2,0.5,1,2,5,10))

control<-trainControl(method = "cv",number=4,savePredictions = "all")

SVM_line <- train(data=stroke_data, factor(stroke) ~ age + avg_glucose_level
                  + hypertension.0 + work_type.Self.employed
                  + smoking_status.smokes + heart_disease.0,
                  method="svmLinear",trControl=control,
                  tuneGrid=SVMgrid,verbose=FALSE)

# SVM_line
SVM_line$results
```

```
##         C  Accuracy Kappa   AccuracySD KappaSD
## 1  1e-04 0.9574253     0 0.0003899105       0
## 2  1e-03 0.9574253     0 0.0003899105       0
## 3  1e-02 0.9574253     0 0.0003899105       0
## 4  5e-02 0.9574253     0 0.0003899105       0
## 5  1e-01 0.9574253     0 0.0003899105       0
## 6  2e-01 0.9574253     0 0.0003899105       0
## 7  5e-01 0.9574253     0 0.0003899105       0
## 8  1e+00 0.9574253     0 0.0003899105       0
## 9  2e+00 0.9574253     0 0.0003899105       0
## 10 5e+00 0.9574253     0 0.0003899105       0
## 11 1e+01 0.9574253     0 0.0003899105       0
```

```r
plot(SVM_line$results$C,SVM_line$results$Accuracy)
```



Para este modelo podemos comprobar tanto en la tabla como en el gráfico, con todas obtenemos el mismo valor de accuracy, esto puede deberse a que los datos podrían quedar bastante diferenciados en dos clúster entre otras cosas, por ello he decidico escoger la menor C, es decir C = 1e-04.

SVMPoly

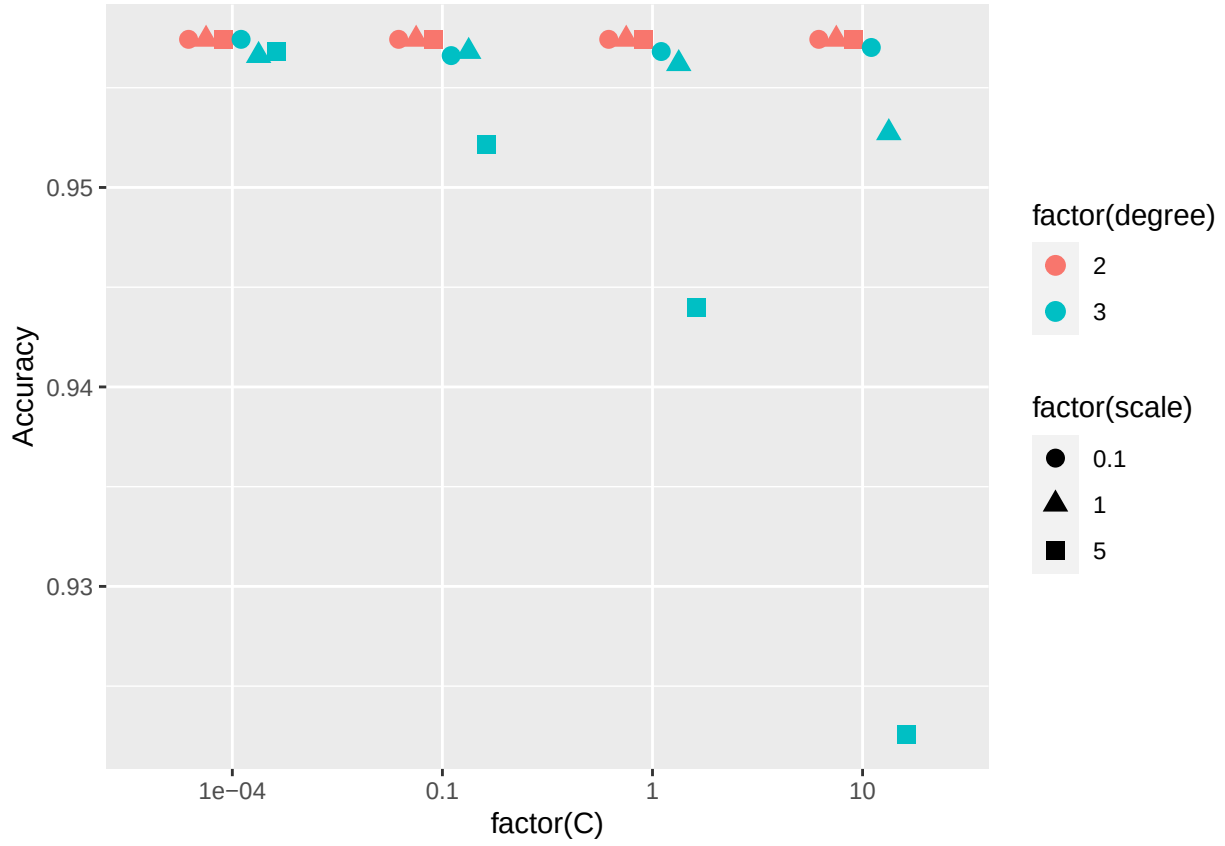
Para este SVM añadimos la escala y el grado del polinomio para poder hacer la separación. La semilla y los grupos de validacion cruzada sera la misma a las anteriores.

Tras pruebas en la integracion para poder construir mas rapidamente el Pmarkdown he decidido poner en el grid solo los parametros mas destacables para pintar su traza en la gráfica

```
SVMgrid<-expand.grid(C=c(0.0001,0.1,1,10),
                     degree=c(2,3),scale=c(0.1,1,5))

control<-trainControl(method = "cv",
                      number=2,savePredictions = "all")


SVM_Poly <- train(data=stroke_data,factor(stroke) ~ age + avg_glucose_level
                  + hypertension.0 + work_type.Self.employed
                  + smoking_status.smokes + heart_disease.0,
                  method="svmPoly",trControl=control,
                  tuneGrid=SVMgrid,verbose=FALSE)

SVM_Poly
```

```
## Support Vector Machines with Polynomial Kernel
##
## 4909 samples
##    6 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (2 fold)
## Summary of sample sizes: 2455, 2454
## Resampling results across tuning parameters:
##
##   C      degree  scale  Accuracy   Kappa
##   1e-04  2       0.1    0.9574252  0.00000000
##   1e-04  2       1.0    0.9574252  0.00000000
##   1e-04  2       5.0    0.9574252  0.00000000
##   1e-04  3       0.1    0.9574252  0.00000000
##   1e-04  3       1.0    0.9566105  0.02253546
##   1e-04  3       5.0    0.9568143  0.03166302
##   1e-01  2       0.1    0.9574252  0.00000000
##   1e-01  2       1.0    0.9574252  0.00000000
##   1e-01  2       5.0    0.9574252  0.00000000
##   1e-01  3       0.1    0.9566105  0.02253546
##   1e-01  3       1.0    0.9568142  0.03174336
##   1e-01  3       5.0    0.9521283  0.03620274
##   1e+00  2       0.1    0.9574252  0.00000000
##   1e+00  2       1.0    0.9574252  0.00000000
##   1e+00  2       5.0    0.9574252  0.00000000
##   1e+00  3       0.1    0.9568143  0.03166302
##   1e+00  3       1.0    0.9562029  0.03035369
##   1e+00  3       5.0    0.9439798  0.05369148
##   1e+01  2       0.1    0.9574252  0.00000000
##   1e+01  2       1.0    0.9574252  0.00000000
##   1e+01  2       5.0    0.9574252  0.00000000
##   1e+01  3       0.1    0.9570179  0.03980338
##   1e+01  3       1.0    0.9527401  0.06538758
##   1e+01  3       5.0    0.9225920  0.10408952
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were degree = 2, scale = 0.1 and C = 1e-04.
```

```
# plot
dat<-as.data.frame(SVM_Poly$results)
ggplot(dat, aes(x=factor(C), y=Accuracy,
                color=factor(degree),pch=factor(scale))) +
  geom_point(position=position_dodge(width=0.5),size=3)
```



Tras la ejecucion y tuneado de parametros, viendo el output del modelo y la grafica podemos concluir que:

- El parametro C debera de tener un valor 1e-04
- El grado polinomial apropiado es de 2
- La escala que debemos tomar para construir nuestro modelo es de 0.01

SVMRBF

Para este SVM Radial (no linial) añadimos sigma, con el cual trataremos de tunear este modelo. La semilla y los grupos de validacion cruzada sera la misma a las anteriores.

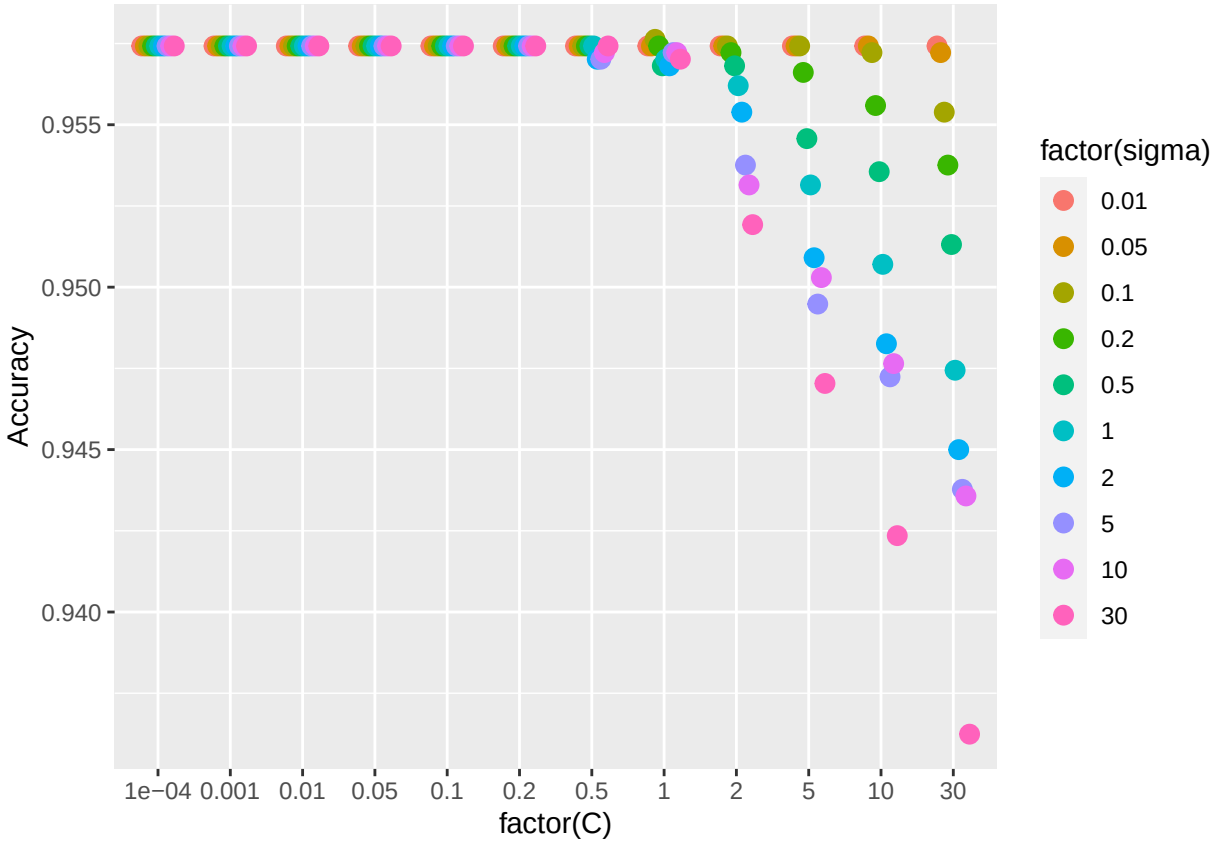
```
SVMgrid<-expand.grid(C=c(0.0001,0.001,0.01,0.05,0.1,0.2,0.5,1,2,5,10,30),
                     sigma=c(0.01,0.05,0.1,0.2,0.5,1,2,5,10,30))

control<-trainControl(method = "cv",
                      number=2,savePredictions = "all")

SVM_Radial <- train(data=stroke_data, factor(stroke) ~ age + avg_glucose_level
                    + hypertension.0 + work_type.Self.employed
                    + smoking_status.smokes + heart_disease.0,
                    method="svmRadial",trControl=control,
                    tuneGrid=SVMgrid,verbose=FALSE)

SVM_Radial
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 4909 samples
##    6 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (2 fold)
## Summary of sample sizes: 2454, 2455
## Resampling results across tuning parameters:
##
##   C      sigma  Accuracy   Kappa
##   1e-04   0.01  0.9574252   0.0000000000
##   1e-04   0.05  0.9574252   0.0000000000
##   1e-04   0.10  0.9574252   0.0000000000
##   1e-04   0.20  0.9574252   0.0000000000
##   1e-04   0.50  0.9574252   0.0000000000
##   1e-04   1.00  0.9574252   0.0000000000
##   1e-04   2.00  0.9574252   0.0000000000
##   1e-04   5.00  0.9574252   0.0000000000
##   1e-04  10.00  0.9574252   0.0000000000
##   1e-04  30.00  0.9574252   0.0000000000
##   1e-03   0.01  0.9574252   0.0000000000
##   1e-03   0.05  0.9574252   0.0000000000
##   1e-03   0.10  0.9574252   0.0000000000
##   1e-03   0.20  0.9574252   0.0000000000
##   1e-03   0.50  0.9574252   0.0000000000
##   1e-03   1.00  0.9574252   0.0000000000
##   1e-03   2.00  0.9574252   0.0000000000
##   1e-03   5.00  0.9574252   0.0000000000
##   1e-03  10.00  0.9574252   0.0000000000
##   1e-03  30.00  0.9574252   0.0000000000
##   1e-02   0.01  0.9574252   0.0000000000
##   1e-02   0.05  0.9574252   0.0000000000
##   1e-02   0.10  0.9574252   0.0000000000
##   1e-02   0.20  0.9574252   0.0000000000
##   1e-02   0.50  0.9574252   0.0000000000
##   1e-02   1.00  0.9574252   0.0000000000
##   1e-02   2.00  0.9574252   0.0000000000
##   1e-02   5.00  0.9574252   0.0000000000
##   1e-02  10.00  0.9574252   0.0000000000
##   1e-02  30.00  0.9574252   0.0000000000
##   5e-02   0.01  0.9574252   0.0000000000
##   5e-02   0.05  0.9574252   0.0000000000
##   5e-02   0.10  0.9574252   0.0000000000
##   5e-02   0.20  0.9574252   0.0000000000
##   5e-02   0.50  0.9574252   0.0000000000
##   5e-02   1.00  0.9574252   0.0000000000
##   5e-02   2.00  0.9574252   0.0000000000
##   5e-02   5.00  0.9574252   0.0000000000
##   5e-02  10.00  0.9574252   0.0000000000
##   5e-02  30.00  0.9574252   0.0000000000
##   1e-01   0.01  0.9574252   0.0000000000
##   1e-01   0.05  0.9574252   0.0000000000
```

```
##   1e-01   0.10  0.9574252   0.0000000000
##   1e-01   0.20  0.9574252   0.0000000000
##   1e-01   0.50  0.9574252   0.0000000000
##   1e-01   1.00  0.9574252   0.0000000000
##   1e-01   2.00  0.9574252   0.0000000000
##   1e-01   5.00  0.9574252   0.0000000000
##   1e-01  10.00  0.9574252   0.0000000000
##   1e-01  30.00  0.9574252   0.0000000000
##   2e-01   0.01  0.9574252   0.0000000000
##   2e-01   0.05  0.9574252   0.0000000000
##   2e-01   0.10  0.9574252   0.0000000000
##   2e-01   0.20  0.9574252   0.0000000000
##   2e-01   0.50  0.9574252   0.0000000000
##   2e-01   1.00  0.9574252   0.0000000000
##   2e-01   2.00  0.9574252   0.0000000000
##   2e-01   5.00  0.9574252   0.0000000000
##   2e-01  10.00  0.9574252   0.0000000000
##   2e-01  30.00  0.9574252   0.0000000000
##   5e-01   0.01  0.9574252   0.0000000000
##   5e-01   0.05  0.9574252   0.0000000000
##   5e-01   0.10  0.9574252   0.0000000000
##   5e-01   0.20  0.9574252   0.0000000000
##   5e-01   0.50  0.9574252   0.0000000000
##   5e-01   1.00  0.9574252   0.0000000000
##   5e-01   2.00  0.9570177  -0.0008008995
##   5e-01   5.00  0.9570177  -0.0008008995
##   5e-01  10.00  0.9572214  -0.0004039431
##   5e-01  30.00  0.9574252   0.0000000000
##   1e+00   0.01  0.9574252   0.0000000000
##   1e+00   0.05  0.9574252   0.0000000000
##   1e+00   0.10  0.9576289   0.0091275606
##   1e+00   0.20  0.9574252   0.0086481741
##   1e+00   0.50  0.9568139  -0.0011910488
##   1e+00   1.00  0.9570177   0.0077138531
##   1e+00   2.00  0.9568139   0.0072585064
##   1e+00   5.00  0.9572214   0.0244853066
##   1e+00  10.00  0.9572214   0.0244853066
##   1e+00  30.00  0.9570178  -0.0008077582
##   2e+00   0.01  0.9574252   0.0000000000
##   2e+00   0.05  0.9574252   0.0000000000
##   2e+00   0.10  0.9574252   0.0086481741
##   2e+00   0.20  0.9572214   0.0081770082
##   2e+00   0.50  0.9568139   0.0072585064
##   2e+00   1.00  0.9562027   0.0059373902
##   2e+00   2.00  0.9553879   0.0121702792
##   2e+00   5.00  0.9537579   0.0230014583
##   2e+00  10.00  0.9531471   0.0291427037
##   2e+00  30.00  0.9519253   0.0356023250
##   5e+00   0.01  0.9574252   0.0000000000
##   5e+00   0.05  0.9574252   0.0086481741
##   5e+00   0.10  0.9574252   0.0086481741
##   5e+00   0.20  0.9566103   0.0150240688
##   5e+00   0.50  0.9545732   0.0188280689
##   5e+00   1.00  0.9531474   0.0158405793
```

```
##   5e+00   2.00  0.9509060   0.0326512888
##   5e+00   5.00  0.9494804   0.0502595212
##   5e+00  10.00  0.9502954   0.0656079677
##   5e+00  30.00  0.9470364   0.0573864394
##   1e+01   0.01  0.9574252   0.0000000000
##   1e+01   0.05  0.9574252   0.0086481741
##   1e+01   0.10  0.9572214   0.0081770082
##   1e+01   0.20  0.9555917   0.0446057279
##   1e+01   0.50  0.9535546   0.0464159166
##   1e+01   1.00  0.9507028   0.0324591839
##   1e+01   2.00  0.9482577   0.0465493038
##   1e+01   5.00  0.9472400   0.0449548697
##   1e+01  10.00  0.9476472   0.0832989555
##   1e+01  30.00  0.9423512   0.0697117817
##   3e+01   0.01  0.9574252   0.0000000000
##   3e+01   0.05  0.9572214   0.0081770082
##   3e+01   0.10  0.9553879   0.0441224270
##   3e+01   0.20  0.9537582   0.0472167860
##   3e+01   0.50  0.9513135   0.0554942192
##   3e+01   1.00  0.9474427   0.0700113132
##   3e+01   2.00  0.9449991   0.0524099780
##   3e+01   5.00  0.9437767   0.0785519871
##   3e+01  10.00  0.9435732   0.0837586375
##   3e+01  30.00  0.9362401   0.0767604671
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.1 and C = 1.
```

```
#plot
dat<-as.data.frame(SVM_Radial$results)
ggplot(dat, aes(x=factor(C), y=Accuracy,
                color=factor(sigma))) +
  geom_point(position=position_dodge(width=0.5),size=3)
```



Para este modelo como se indica en el output y apreciamos en la grafica sigma y C deben valer 0.2 y 2 respectivamente para lograr el mayor accuracy.

Modelos ganadores y ánalisis sesgo-varianza

En este apartado, una vez el tuneadoo de nuestros modelos vamos a pasar realizar el analisis sesgo-varianza, para este apartado y la construccion de los modelos utilizaremos las funciones propuestas del profesor, las cuales nos brindaran las medias de accuracy en los modelos aplicando la validacion cruzada.

Para todos los modelos construidos se van a utilizar los parametros mas adecuado tras el tuneo, con la misma semilla, ademas de, 4 grupos para la validacion cruzada repetida 5 veces.

En este punto vamos a resolver los criterios del apartado b).

Tambien elegiremos los candidatos para ser los posibles modelos ganadores.

```
#LOGISTICA
source("cruzadas avnnet y log binaria.R")

medias_model_1 <- cruzadalogistica(data=stroke_data,
                    vardep="stroke",listconti= c("age", "avg_glucose_level",
                      "hypertension.0","work_type.Self.employed",
                      "smoking_status.smokes", "heart_disease.0"),
                    listclass=c(""), grupos=4,sinicio=1234967,repe=5)

medias_model_1$modelo="Logística"
```

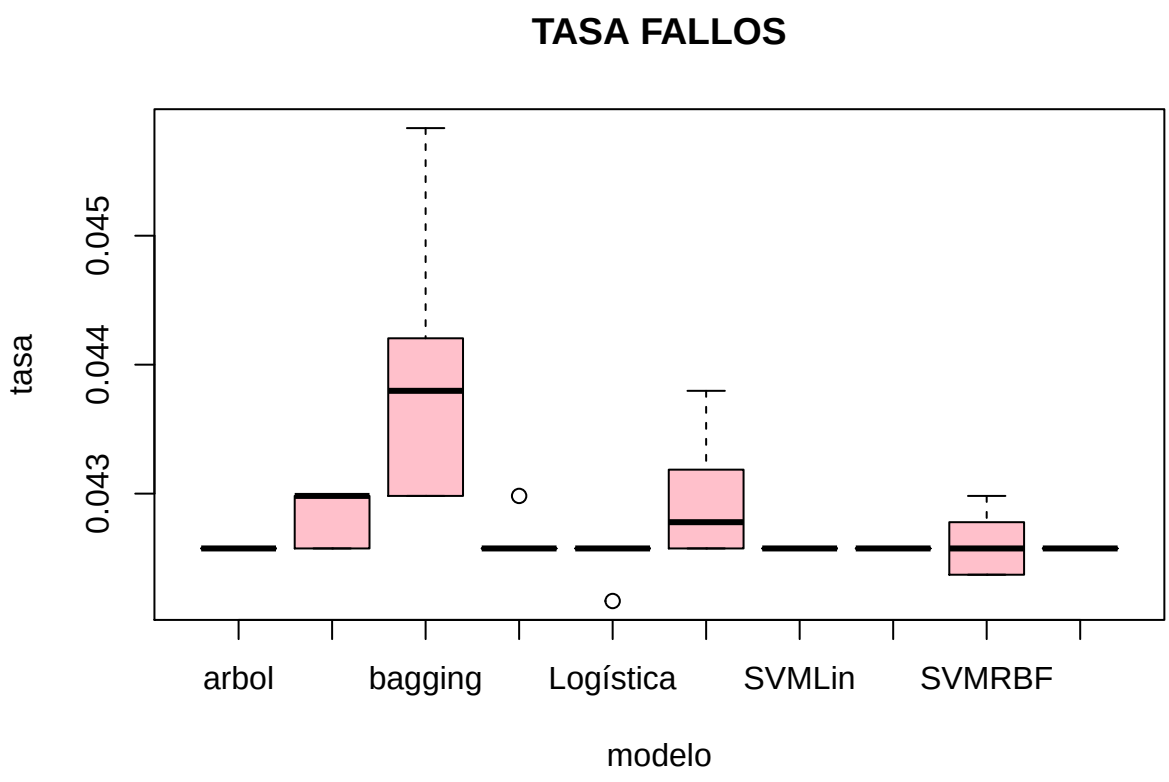
```r
# RED
medias_model_2 <- cruzadaavnnetbin(data=stroke_data,
                     vardep="stroke",listconti= c("age", "avg_glucose_level",
                        "hypertension.0","work_type.Self.employed",
                        "smoking_status.smokes", "heart_disease.0"),
                     listclass=c(""),grupos=4,sinicio=1234967,repe=5,
                     size=c(5),decay=c(0.1),repeticiones=5,itera=100,
                     trace = TRUE)

medias_model_2$modelo="avnnet1"


# ARBOL
source ("cruzada arbolbin.R")

medias_model_3 <- cruzadaarbolbin(data=stroke_data,
                      vardep="stroke",listconti= c("age", "avg_glucose_level",
                         "hypertension.0","work_type.Self.employed",
                         "smoking_status.smokes", "heart_disease.0"),
                      listclass=c(""),grupos=4,sinicio=1234967,repe=5,
                      cp=c(0.1),minbucket =30)

medias_model_3$modelo="arbol"


# BANGGIN
source ("cruzada rf binaria.R")

medias_model_4 <- cruzadarfbin(data=stroke_data,
                    vardep="stroke",listconti= c("age", "avg_glucose_level",
                       "hypertension.0","work_type.Self.employed",
                       "smoking_status.smokes", "heart_disease.0"),
                    listclass=c(""),grupos=4,sinicio=1234967,repe=5,
                    nodesize=10,mtry=6,ntree=1000,replace=TRUE)

medias_model_4$modelo="bagging"


# RANDOM FORES
medias_model_5 <- cruzadarfbin(data=stroke_data,
                     vardep="stroke",listconti= c("age", "avg_glucose_level",
                        "hypertension.0","work_type.Self.employed",
                        "smoking_status.smokes", "heart_disease.0"),
                     listclass=c(""),grupos=4,sinicio=1234967,repe=5,
                     nodesize=12,mtry=3,ntree=500,replace=TRUE)

medias_model_5$modelo="rf"


# GRADIENT BOOSTING
source ("cruzada gbm binaria.R")

medias_model_6 <- cruzadagbmbin(data=stroke_data,
```

```
                    vardep="stroke",listconti= c("age", "avg_glucose_level",
                       "hypertension.0","work_type.Self.employed",
                       "smoking_status.smokes", "heart_disease.0"),
                    listclass=c(""),grupos=4,sinicio=1234967,repe=5,
                    n.minobsinnode=5,shrinkage=0.015,n.trees=100,
                    interaction.depth=2)

medias_model_6$modelo="gbm"


# XGBOOST
source ("cruzada xgboost binaria.R")

medias_model_7 <-cruzadaxgbmbin(data=stroke_data,
                    vardep="stroke",listconti= c("age", "avg_glucose_level",
                       "hypertension.0","work_type.Self.employed",
                       "smoking_status.smokes", "heart_disease.0"),
                    listclass=c(""),grupos=4,sinicio=1234967,repe=5,
                    min_child_weight=5,eta=0.015,nrounds=100,max_depth=6,
                    gamma=0,colsample_bytree=1,subsample=1,
                    alpha=0,lambda=0,lambda_bias=0)

medias_model_7$modelo="xgbm"


# SVM

# SVM linear
source ("cruzada SVM binaria lineal.R")

medias_model_8 <- cruzadaSVMbin(data=stroke_data,
                    vardep="stroke",listconti= c("age", "avg_glucose_level",
                       "hypertension.0","work_type.Self.employed",
                       "smoking_status.smokes", "heart_disease.0"),
                    listclass=c(""),grupos=4,sinicio=1234967,repe=5,
                    C=0.0001)

medias_model_8$modelo="SVMLin"


# SVMPoly
source ("cruzada SVM binaria polinomial.R")

medias_model_9 <- cruzadaSVMbinPoly(data=stroke_data,
                    vardep="stroke",listconti= c("age", "avg_glucose_level",
                     "hypertension.0","work_type.Self.employed",
                     "smoking_status.smokes", "heart_disease.0"),
                    listclass=c(""),grupos=4,sinicio=1234967,repe=5,
                    C=0.0001,degree=2,scale=0.01)

medias_model_9$modelo="SVMPoly"
```

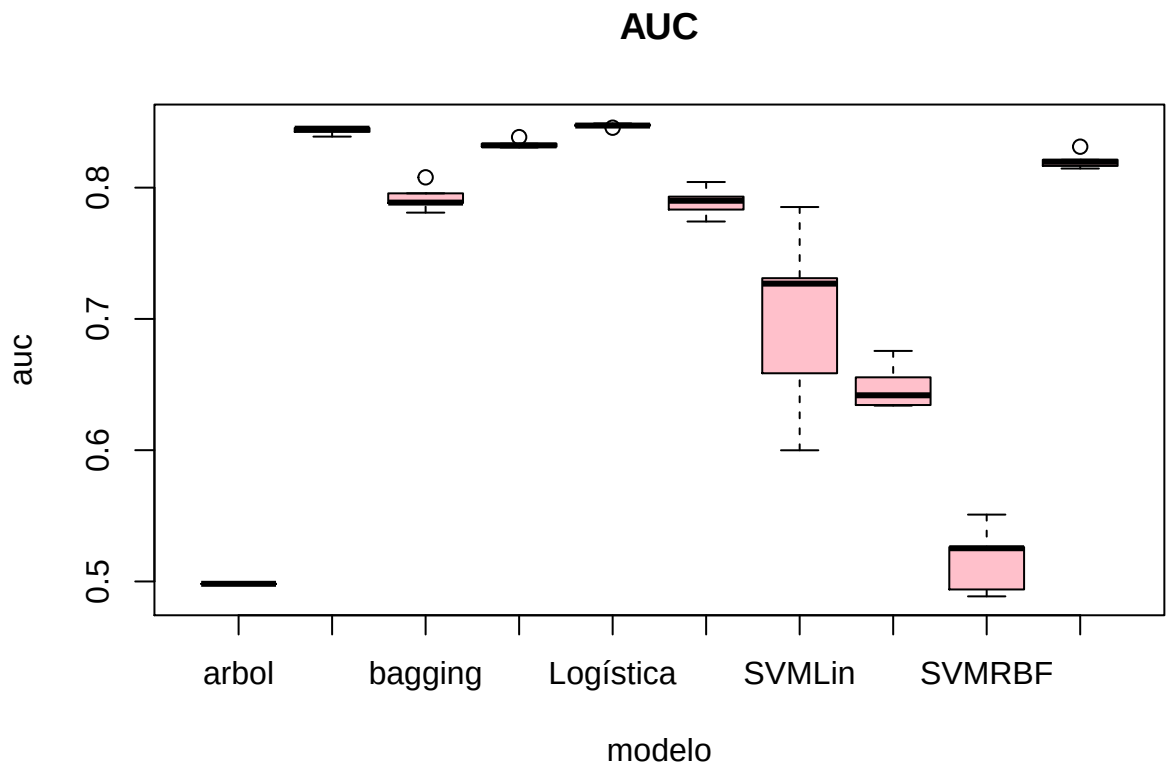
```r
# SVMRBF
source ("cruzada SVM binaria RBF.R")

medias_model_10 <- cruzadaSVMbinRBF(data=stroke_data,
                      vardep="stroke",listconti= c("age", "avg_glucose_level",
                       "hypertension.0","work_type.Self.employed",
                       "smoking_status.smokes", "heart_disease.0"),
                      listclass=c(""),grupos=4,sinicio=1234967,repe=5,
                      C=2,sigma=0.2)

medias_model_10$modelo="SVMRBF"
```

Una vez creados estos modelos con validacion cruzada vamos a pintar todos los resultados obtendios analizand sesgo y varianza de cada modelo, además veremos que modelo es el mas apropiado para nuestros datos.

```r
## PINTAR DATOS SESGO-VARIANZA
union_models<-rbind(medias_model_1,medias_model_2,medias_model_3,
                    medias_model_4,medias_model_5,medias_model_6,
                    medias_model_7,medias_model_8,medias_model_9,
                    medias_model_10)
# tasa fallos
boxplot(data=union_models,tasa~modelo,col="pink",main="TASA FALLOS")
```



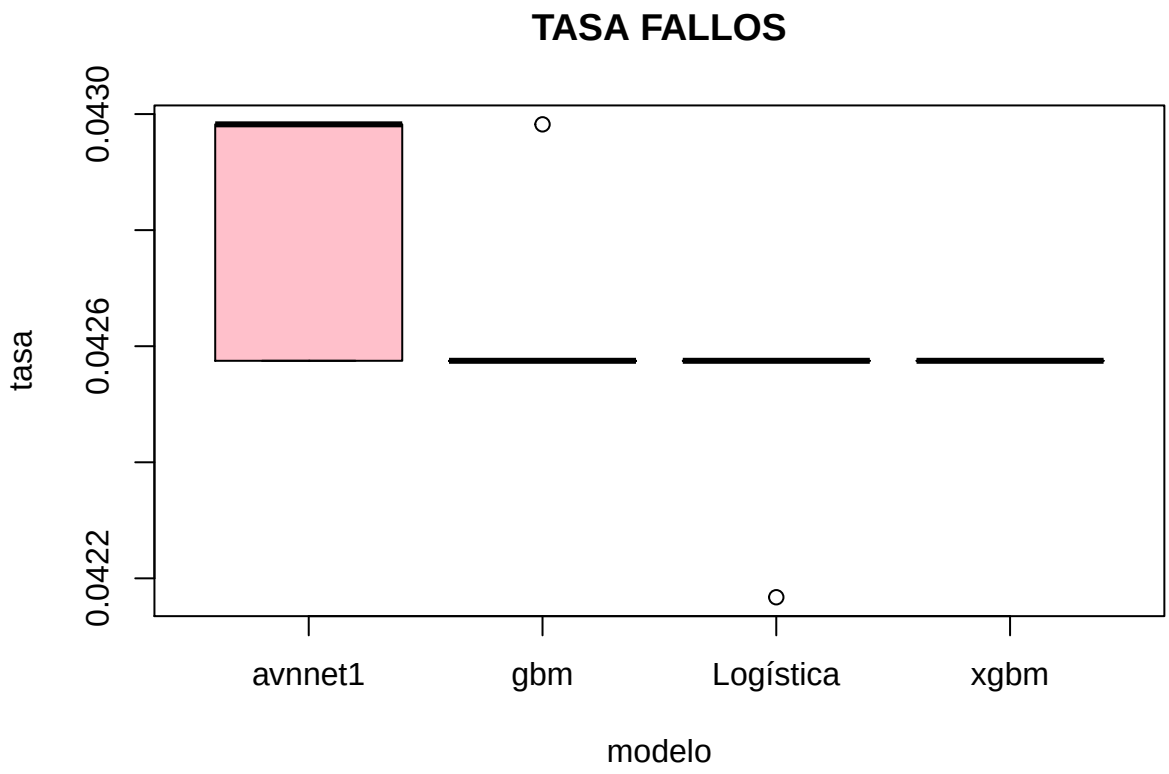
```r
# auc
boxplot(data=union_models,auc~modelo,col="pink",main="AUC")
```



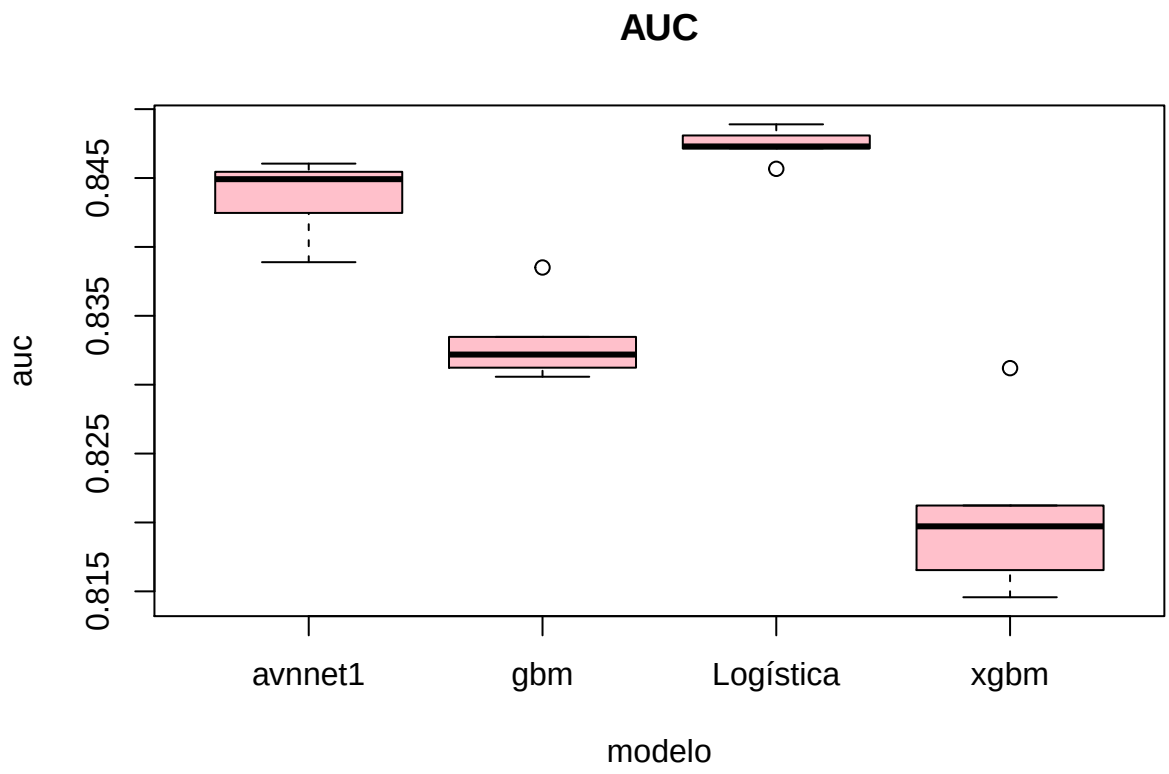
Empezamos examinando la tasa de fallos donde podemos ver como el modelo con mayort tasa de fallos y mas varianza es bagging, por otro lado llama la atención como SVM Radial consigue en algunos casos una tasa de fallos meneor que el resto.

Sin embargo, al contrario de lo qe se contempla en la grafica AUC, podemos comprobar como cualquier metodo de SVM no es apropiado para la prediccion de este tipo de datos. Podemos observar como los modelos de regresion logistica, redes neuronales y gbm/xgmb estan compitiendo por ser el mejor modelo, a continuacion vemos una grafica ampliada solo con estos modelos.

```r
## ZOOM
union_models_zoom<-rbind(medias_model_1,medias_model_2,
                  medias_model_6,medias_model_7)
# tasa fallos
boxplot(data=union_models_zoom,tasa~modelo,col="pink",main="TASA FALLOS")
```




```
# auc
boxplot(data=union_models_zoom,auc~modelo,col="pink",main="AUC")
```



Podemos ver que pese a tener todos un AUC bastante elvado por encima de 0.8, los modelos de regresión logística y la red neuronal son los mas interesantes para nuestros datos, en este caso he decidido quedarme con el modelo de regresion logistica, debido a su baja varianza y ya que es un modelo mas estable y menos complejo que el construido con una red.

Ensamblado

Para finalizar este ejercicio queda probar con la ensamblacion de modelos, y si la combinación de modelos anteriores, podría ser una opcion interesante para generar un modelo predictivo mas potente para nuestros datos. En este punto cumpliremos con los objetivos del apartado f).

Para esta ensamblado de modelos solo vamos a aplicar el metodo de promediado, es decir este ensamblado se calculará en base a las medias de los modelos que queramos combinar.

Otro metodo de ensamblado, que no va ser aplicada en este ejercicio sería utilizar como variables imput las predicciones de los modelos previamete calculado y generando un modelo de prediccion nuevo

Este promediado nos ayudará a rebajar la varianza de los modelos ya calculados.

He decidido hacer el ensamblado combinando los 4 modelos ya propuestos como posibles ganadores en el apartado anterior, que son: la red, regresion logística, gradient boosting y xgboosting.

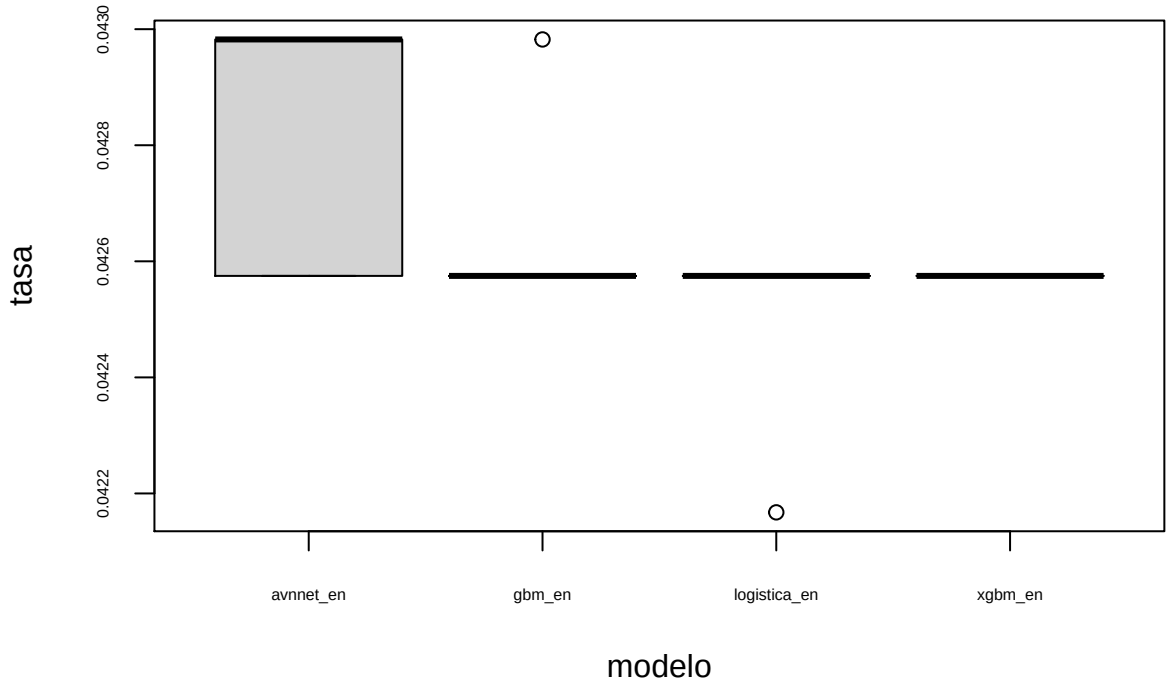
```
source("cruzadas ensamblado binaria fuente.R")

vardep<-"stroke"
listconti<-c("age", "avg_glucose_level",
             "hypertension.0","work_type.Self.employed",
```

```
            "smoking_status.smokes", "heart_disease.0")
listclass<-c("")
grupos<-4
sinicio<-1234967
repe<-5

# REGRESION LOGISTICA
medias_model_1_en <- cruzadalogistica(data=stroke_data,
                      vardep=vardep,listconti=listconti,
                      listclass=listclass,grupos=grupos,
                      sinicio=sinicio,repe=repe)

medias_model_1_bis<-as.data.frame(medias_model_1_en[1])
medias_model_1_bis$modelo <- "logistica_en"
predi_model_1 <- as.data.frame(medias_model_1_en[2])
predi_model_1$logistica_en <- predi_model_1$Yes

# RED
medias_model_2_en <- cruzadaavnnetbin(data=stroke_data,
                      vardep=vardep,listconti=listconti,
                      listclass=listclass,grupos=grupos,
                      sinicio=sinicio,repe=repe,
                      size=c(5),decay=c(0.1),repeticiones=5,itera=100,)

medias_model_2_bis<-as.data.frame(medias_model_2_en[1])
medias_model_2_bis$modelo <- "avnnet_en"
predi_model_2 <- as.data.frame(medias_model_2_en[2])
predi_model_2$avnnet_en <- predi_model_2$Yes

# GRADIENT BOOSTING
medias_model_6_en <- cruzadagbmbin(data=stroke_data,
                        vardep=vardep,listconti=listconti,
                        listclass=listclass,grupos=grupos,
                        sinicio=sinicio,repe=repe,
                        n.minobsinnode=5,shrinkage=0.015,n.trees=100,
                        interaction.depth=2)

medias_model_6_bis<-as.data.frame(medias_model_6_en[1])
medias_model_6_bis$modelo <- "gbm_en"
predi_model_6 <- as.data.frame(medias_model_6_en[2])
predi_model_6$gbm_en <- predi_model_6$Yes

# XGBOOST
medias_model_7_en <- cruzadaxgbmbin(data=stroke_data,
                      vardep=vardep,listconti=listconti,
                      listclass=listclass,grupos=grupos,
                      sinicio=sinicio,repe=repe,
                      min_child_weight=5,eta=0.015,nrounds=100,max_depth=6,
                      gamma=0,colsample_bytree=1,subsample=1,
                      alpha=0,lambda=0,lambda_bias=0)

medias_model_7_bis<-as.data.frame(medias_model_7_en[1])
medias_model_7_bis$modelo <- "xgbm_en"
```

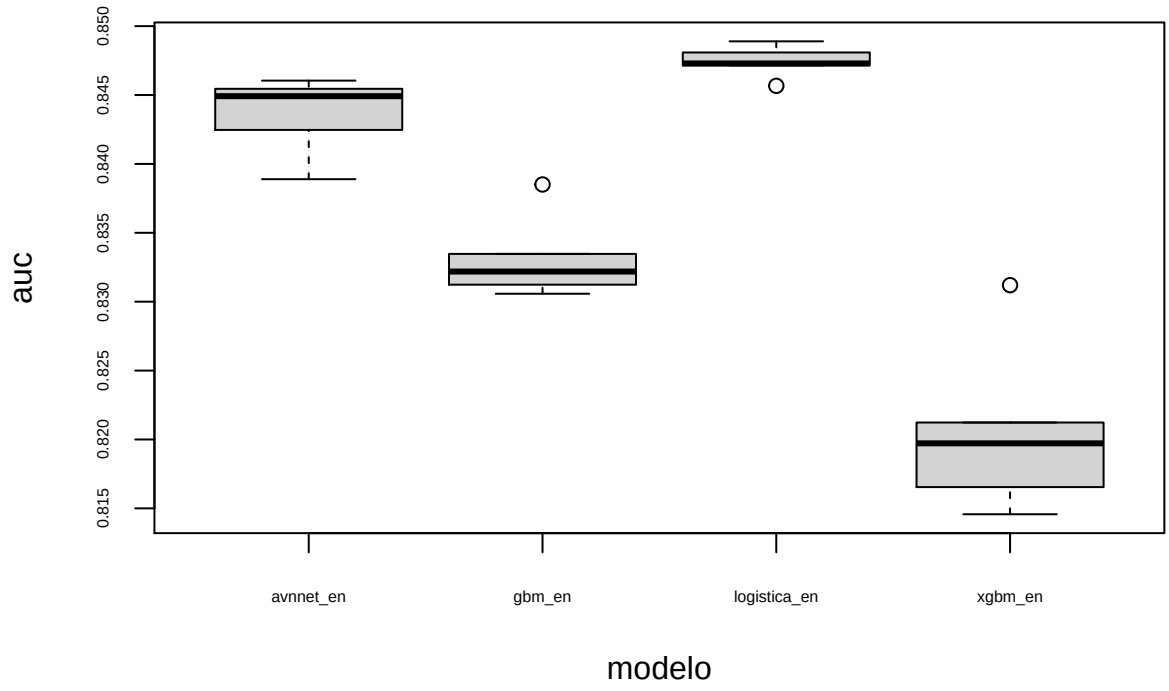
```
predi_model_7 <- as.data.frame(medias_model_7_en[2])
predi_model_7$xgbm_en <- predi_model_7$Yes

union_models_bis<-rbind(medias_model_1_bis,medias_model_2_bis,
                        medias_model_6_bis,medias_model_7_bis)
```

Calculados los modelos de nuevo nos queda pintar la tasa de fallos y AUC, como podemos ver en los siguientes graficos, los resultados obtenidos son iguales que los anteriores. Con la diferencia de haber usado otras funciones para su constuccion, esto se debe a las variables extra que aporta para el ensamblado promediado de los moedelos.

```
# PLOT
par(cex.axis=0.5)
boxplot(data=union_models_bis,tasa~modelo)
```




```
boxplot(data=union_models_bis,auc~modelo)
```



Procedemos a calcular los modelos ensamblado.

He considerado crear 3 modelos ensamblados para probar esta herramienta, aunque considero que con la poca varianza de la regresion logísstica, tasa de fallos y el AUC, sería un modelo bastante bueno y robusto, para nuestros datos.

```
# Empezamos con el ensamblado
uni_predi<-cbind(predi_model_1,predi_model_2,predi_model_6,predi_model_7)
uni_predi<- uni_predi[, !duplicated(colnames(uni_predi))]

# promedios
uni_predi$predi_11 <- (uni_predi$logistica_en+uni_predi$avnnet_en)/2
uni_predi$predi_12 <- (uni_predi$avnnet_en+uni_predi$gbm_en
                       +uni_predi$xgbm_en)/3
uni_predi$predi_13 <- (uni_predi$logistica_en+uni_predi$avnnet_en
                        +uni_predi$gbm_en+uni_predi$xgbm_en)/4
```

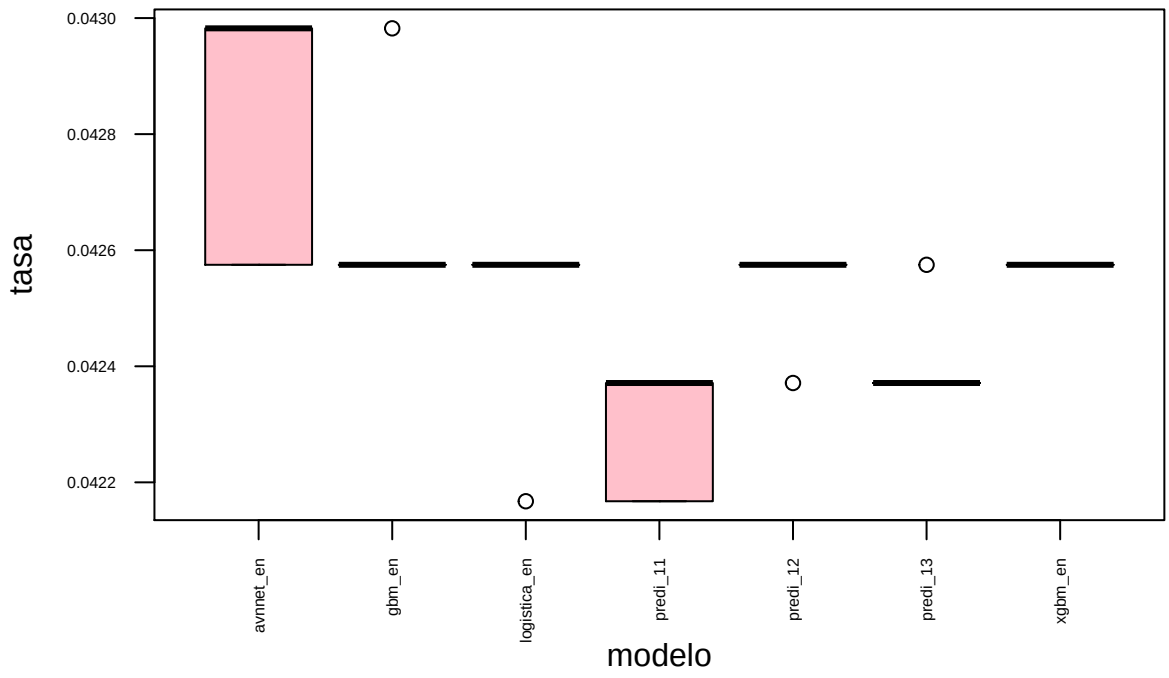
A continuacion se muestra un pipeline de codigo necesario para poder procesar los modelos de ensamblado.

```
# PROCESADO DE ENSAMBLADOS
listado<-c("logistica_en", "avnnet_en",
           "gbm_en", "xgbm_en", "predi_11",
           "predi_12",  "predi_13")

# Defino funcion tasafallos
tasafallos<-function(x,y) {
```

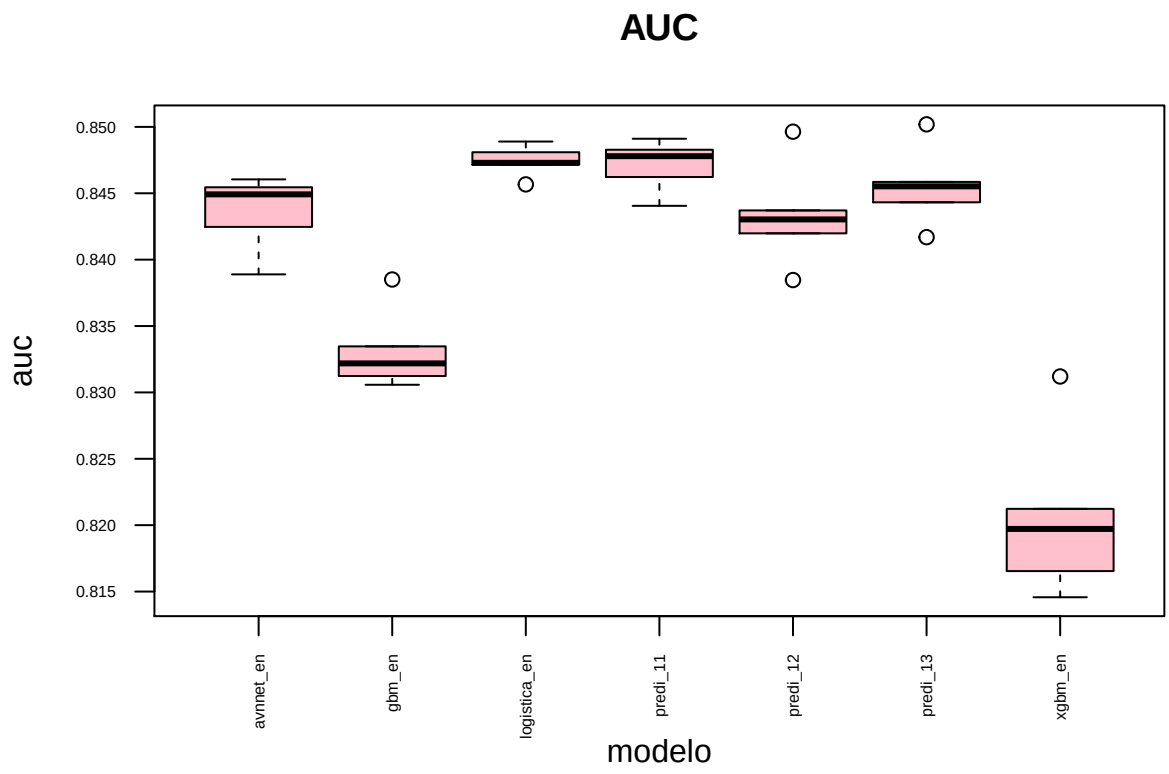
```
  confu<-confusionMatrix(x,y)
  tasa<-confu[[3]][1]
  return(tasa)
}

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(uni_predi$Rep))
uni_predi$Rep<-as.factor(uni_predi$Rep)
uni_predi$Rep<-as.numeric(uni_predi$Rep)


medias0<-data.frame(c())
for (prediccion in listado)
{
  uni_predi$proba<-uni_predi[,prediccion]
  uni_predi[,prediccion]<-ifelse(uni_predi[,prediccion]>0.5,"Yes","No")
  for (repe in 1:repeticiones)
  {
    paso <- uni_predi[(uni_predi$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)
  }
}
```

Por ultimo tras el procesado de los ensamblados vamso a pintar finalmente estos modelos para analizar tasa de fallos, AUC, sesgo y varianza, obtener las conclusiones pertinentes y seleccionar nuestro modelo ganador.

```
# PLOT
par(cex.axis=0.5,las=2)
# FALLOS
boxplot(data=medias0,tasa~modelo,col="pink",main="TASA FALLOS")
```

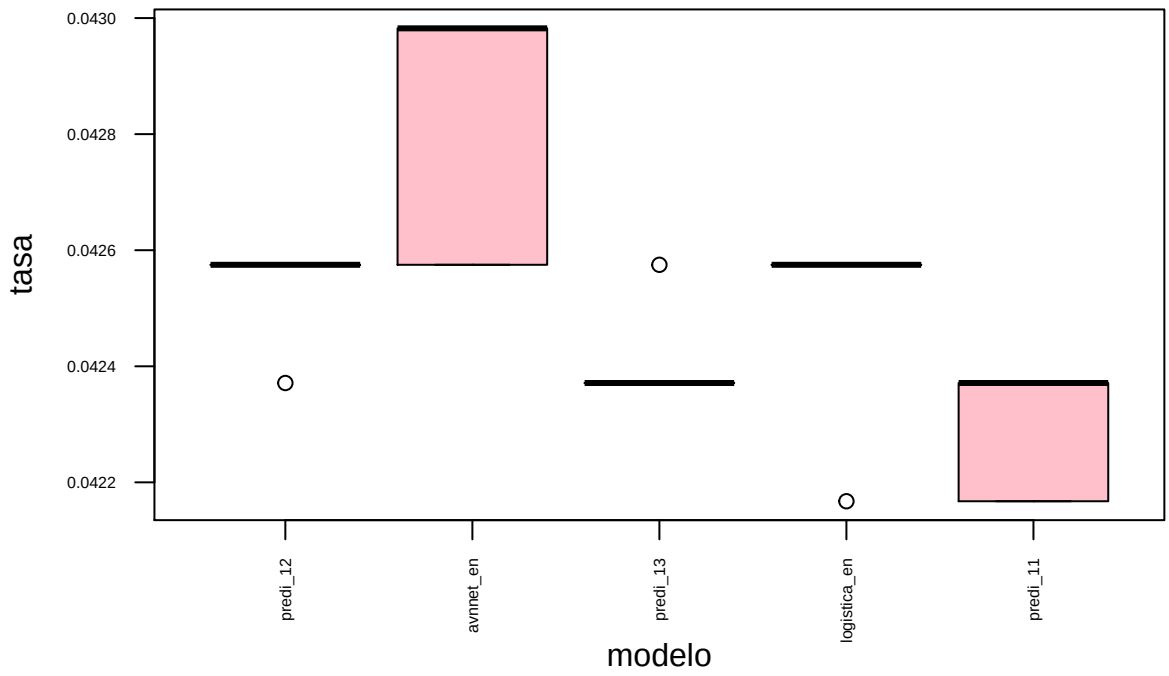



```
# AUC
boxplot(data=medias0,auc~modelo,col="pink",main="AUC")
```

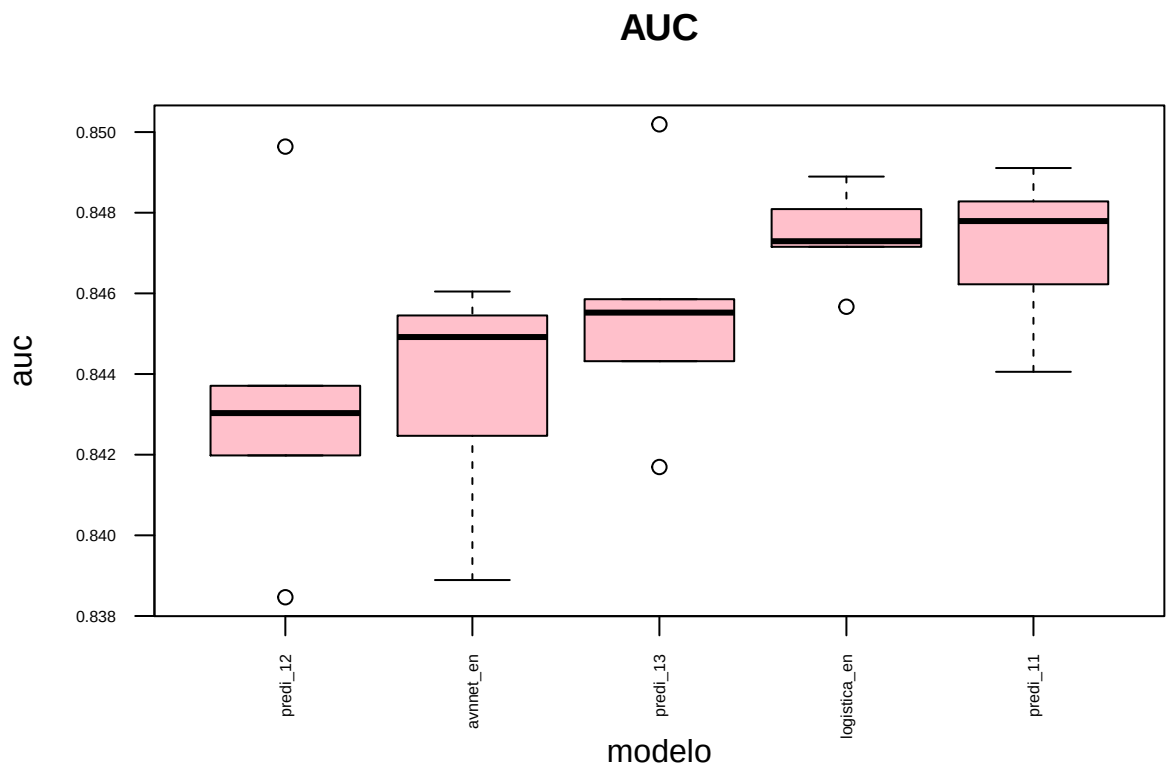


Podemos ver como los algoritmos de arboles se quedan bastante por debajo del resto, por ello vamos a hacer "zoom" a los modelos calculados con la regresión lineal, red neuronal y los modelos ensamblado.

```
# ZOOM
listado_zoom<-c("logistica_en", "avnnet_en",
                "predi_11", "predi_12", "predi_13")
medias0$modelo<-as.character(medias0$modelo)
mediasver<-medias0[medias0$modelo %in% listado_zoom,]
mediasver$modelo <- with(mediasver,
                         reorder(modelo,auc, median))

# PLOT
par(cex.axis=0.5,las=2)
# FALLOS
boxplot(data=mediasver,tasa~modelo,col="pink",main="TASA FALLOS")
```




```
# AUC
boxplot(data=mediasver,auc~modelo,col="pink",main='AUC')
```



En conclusión, podemos ver como la regresion logistica y el modelo de ensamblado con logistica y red, son los claros ganadores, aun que realmente sea en una escala de 0.001 por lo tanto podemos afirmar que los 4 modelos son buenos.

Conclusión

Como modelo ganador, finalmente he decidido elegir el de regresión logística ya que pese a que el modelo de ensamblado estó un poco por encima en AUC el modelo de regrseión logística es un modelo claramente mucho mas sencillo y con una varianza bastante mas baja respecto al resto de modelos.

Por último destacar que el modelo de ensamblado logistica y red es el que menor tasa de fallos presenta, aunque como he comentado anteriormente al ser una escala del tercer decimal, a penas tendria importancia y nos giamos por AUC como métrica mas importante y sencillez del modelo.