Machine Learning con R

Ignacio Correcher Sánchez

29/06/2023

Contents

1 Presentación de los datos

El dataset que se ha escogido tiene como título "**The Gender Pay Gap in the General Social Survey**" y se ha obtenido de la página web sugerida https://vincentarelbundock.github.io/. Estos datos contienen información sobre personas encuestadas entre 1974 y 2018, donde podemos encontrar su género, ocupación, edad, nivel de estudios, etc.
El objetivo del trabajo será estudiar y predecir la variable *gender*, la cual representa el género del individuo.

En el dataset tenemos un total de 61697 observaciones de 11 variables distintas, a saber:

- year: Año de la encuesta.
- realrinc: Sueldo anual, homogeneizado a dolares estadounidenses con valor de 1986.
- age: Edad.
- occ10: Código de ocupación, en base a la codificación de 2010.
- occrecode: Recodificación del código de ocupación para clasificarlo dentro de las 11 grandes categorías.
- prestg10: Puntuación de prestigio del trabajo desarrollado, i.e, valoración del puesto de trabajo según la socidedad.
- childs: Número de hijos.
- wrkstat: Situación laboral, por ejemplo, trabajo a tiempo completo, tiempo parcial, desempleado, etc.
- gender: Hombre o mujer.
- educcat: Nivel de estudios.
- maritalcat: Estado civil, por ejemplo, casado, soltero, divorciado, etc.

1.1 Visualización de las variables

Antes de empezar se realizará un ajuste al número de observaciones, ya que 60k pueden ser demasiadas, cosa que ralentizarían mucho los algoritmos y el desarrollo del análisis. Se reducirá el dataset a 5000 observaciones, pero manteniendo la proporción de la variable gender que tenemos originalmente, ya que sigue estando muy por encima del número requerido, el cual eran 300 observaciones

```
wagesDS <- wagesDS[caret::createDataPartition(y = wagesDS$gender, times = 1,
                   p = 5000 / nrow(wagesDS), list = FALSE),]
```

A continuación se va a realizar un análisis exploratorio, de forma visual, de algunas variables que se han considerado interesantes.

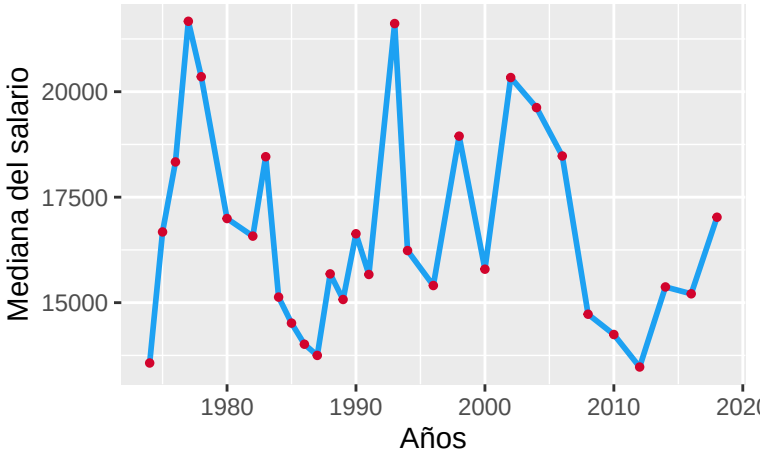

Figure 1: Evolución de la mediana del salario

Se puede observar como evoluciona la mediana del salario de las personas encuestadas, alcanzando picos en años como 1985 y 1996. Unos máximos de 20k anuales, que para ser EEUU podría no parecer demasiado, pero cuando luego analicemos los grupos laborales y los estudios más predominantes entre la población estudiada entenderemos que estas cifras se ajustan a lo esperado.

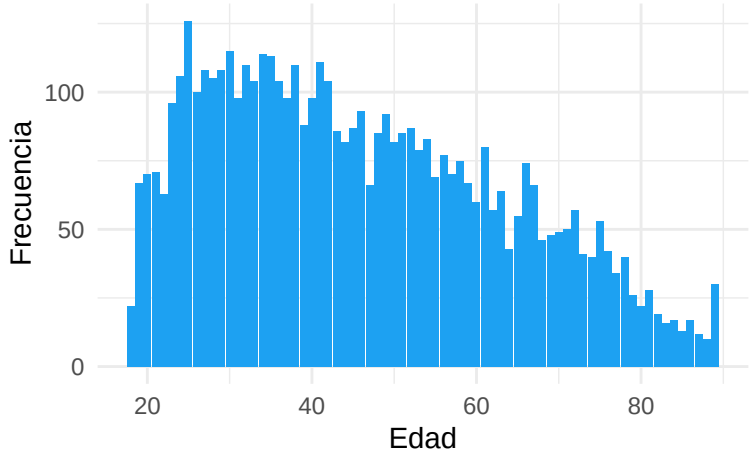

Figure 2: Edad de las personas encuestadas

La mayoría de las personas forman parte del dataset tienen una edad comprendida entre 20 y 50 años. La distribución de personas que tenemos tiene sentido con la distribución del mundo real, donde tiene un mayor porcentaje de representación personas jóvenes que personas de una edad más avanzada, es por ello que la muestra obtenida puede ser representativa en cuanto a la edad de la población total.

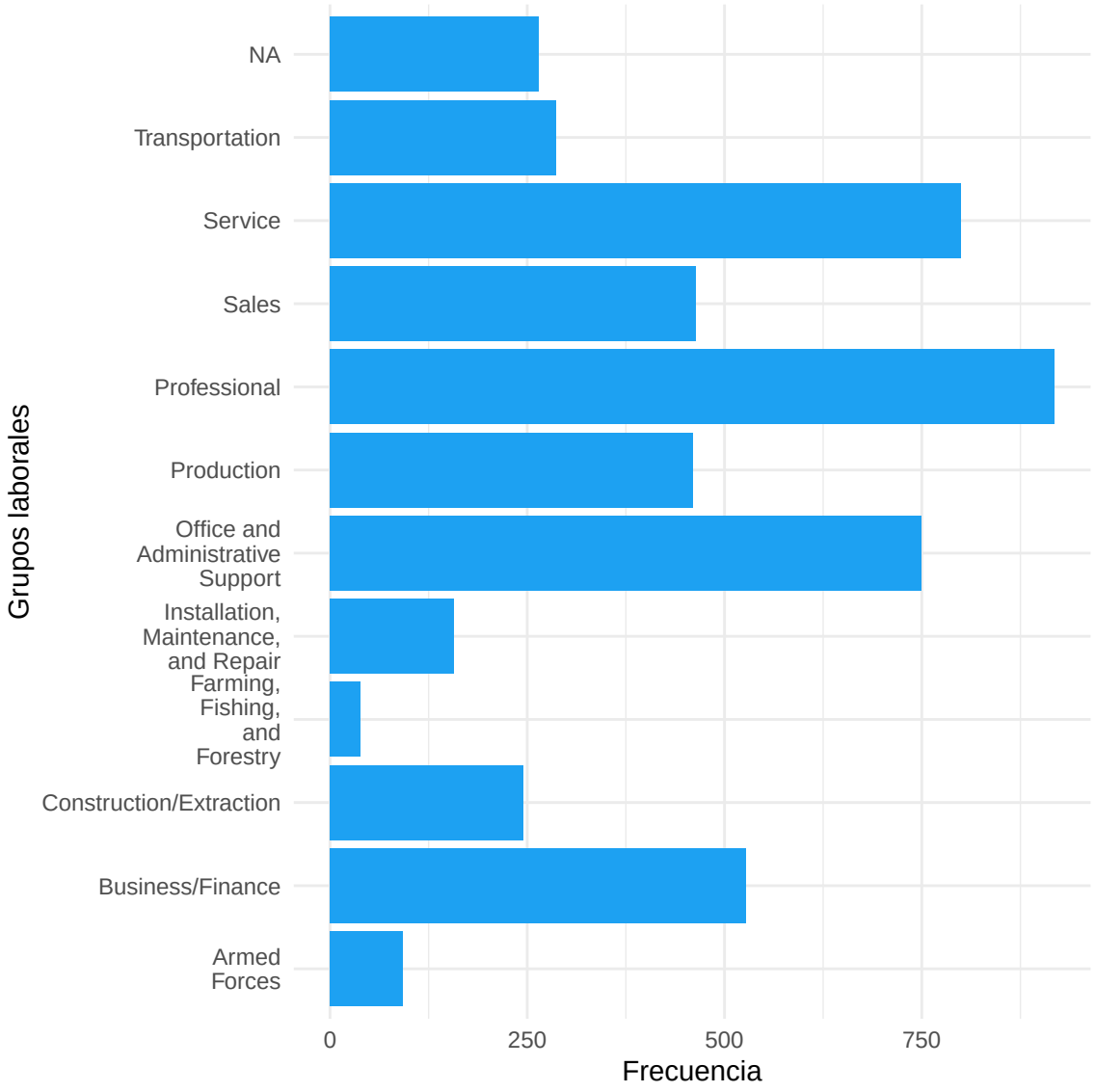

Figure 3: Frecuencia de grupos laborales

Los grupos laborales que se encuentran más representados son profesionales, servicios y administrativos.

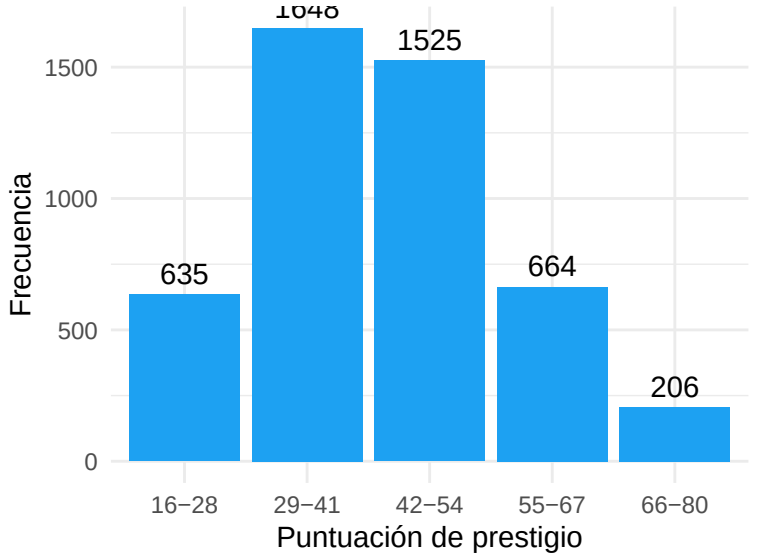

Figure 4: Distribución de las puntuaciones laborales

Como era de esperar la "puntuación de prestigio" de los trabajos de las personas encuestadas siguen una distribución parecida a la normal.

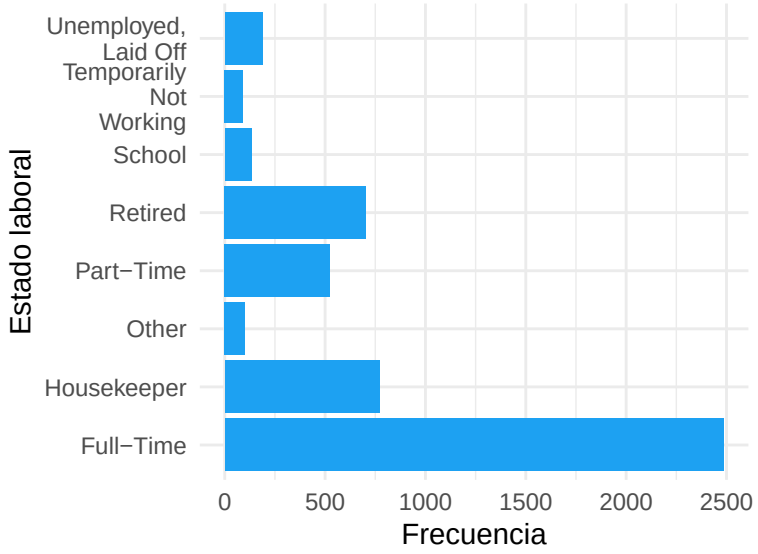

Figure 5: Estado laboral de los encuestados

Con una gran diferencia la mayoría de individuos que han tomado parte en este estudio están contratados a tiempo completo, la segunda categoría más representada son los amos y amas de casa y la tercera son los jubilados.

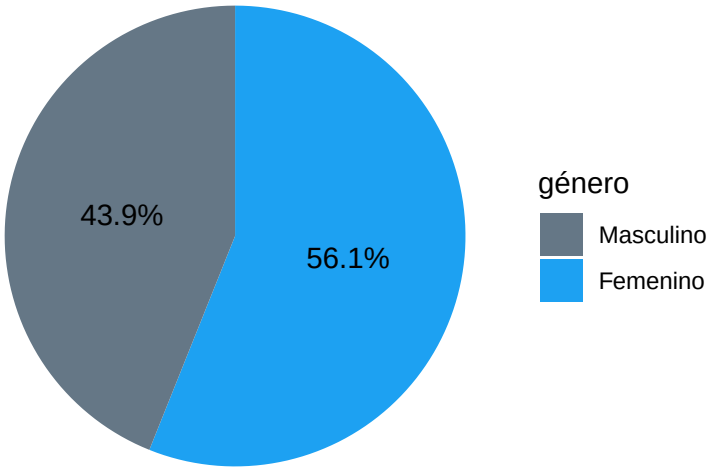

Figure 6: Distribución del género entre los encuestados

Como podemos ver **la clase minoritaria en el dataset correspondería con el género masculino**, pero realmente no supondría un problema ya que los porcentajes de observaciones de ambas clases resultan muy parejos. Tendríamos al genero masculino con un 44% y al femenino con un 56%.

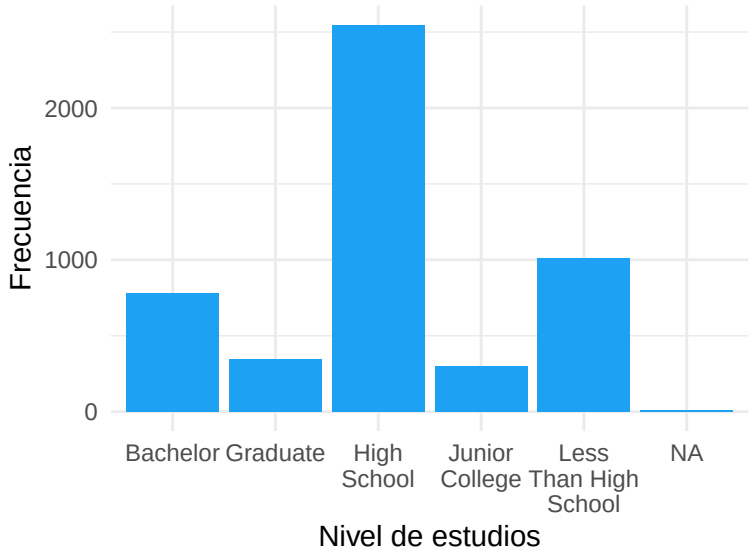

Figure 7: Nivel de estudios de los encuestados

Con gran diferencia el mayor nivel de estudios completado por los encuestados es el instituto, seguido por gente que no lo ha completado.

1.2 Depuración

Ahora que ya hemos realizado una descripción gráfica de las variables originales, vamos a proceder a depurarlas y preparar el dataset para su utilización en los modelos.
En primer lugar separaremos las variables en categóricas, continuas y variable dependiente, y también codificaremos la variable output como Yes/No, dependiendo de su género es femenino o masculino:

```
vContinuas <- c("age","realrinc", "occ10")
vCategoricas <- c("year", "occrecode", "prestg10","childs", "wrkstat",
                  "educcat", "maritalcat")
vDependiente <- "Female"

wagesDS$Female<-ifelse(wagesDS$gender=="Female","Yes","No")
wagesDS$Female <- as.factor(wagesDS$Female)
wagesDS$Female<-relevel(wagesDS$Female,ref="No")
wagesDS <- wagesDS[, c(vContinuas, vCategoricas,vDependiente)]
```

wagesDS <- wagesDS[caret::createDataPartition(y = wagesDS$Female, times = 1, p = 5000 / nrow(wagesDS), list = FALSE),]

Ahora vamos a proceder a resolver el problema de los missings, en primer lugar vamos a dar un primer vistazo a la cantidad de missings de cada variable:

```
gg_miss_var(wagesDS)
```

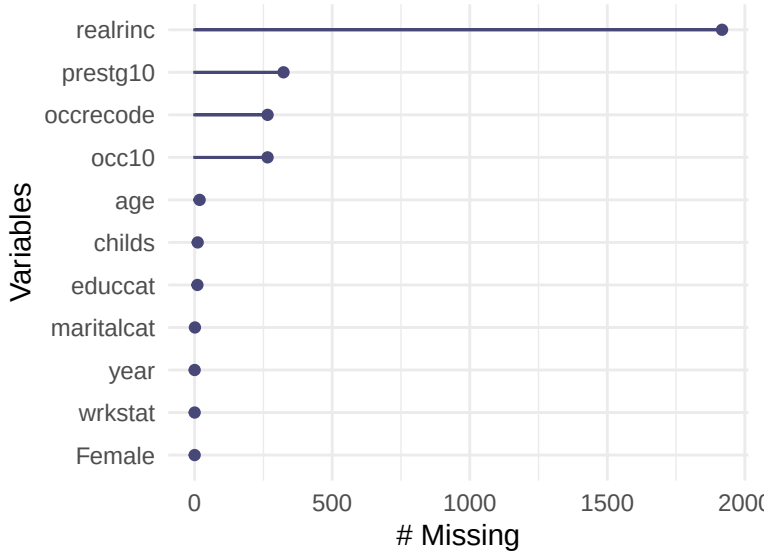

Figure 8: Total de missings en el dataset original

Como podemos observar, la única variable que preocupa en cuanto a missings es *realrinc*, la cual representa el salario. Es una variable, a priori, **MUY** importante para el estudio, ya que lo que se pretende es analizar la desigualdad salarial en base al género. Debido a esta importancia hay 2 situaciones que plantearse:

- Eliminar todas las observaciones con missings en esta variable, son cerca de 2k missings, lo que dejaría un valor de 3k observaciones.

- También podemos imputarlas en base a la media, o mejor aún, utilizando la mediana, ya que el salario es una variable que suele tener mucha varianza y no suele tener una distribución demasiado simétrica.

Se ha optado por realizar una imputación de los missings en base a la mediana del grupo laboral al que pertenece cada individuo, ya que las 2k observaciones que hay que eliminar supondrían casi 1/3 de la muestra original. Posteriormente se realizará el análisis de missings y en función del resultado se procederá con los restantes.

```
wagesDS <- wagesDS %>% group_by(occrecode) %>% mutate(realrinc = ifelse(is.na(realrinc),
                            trunc(median(realrinc, na.rm = TRUE)),
                            realrinc))

gg_miss_var(wagesDS)
```

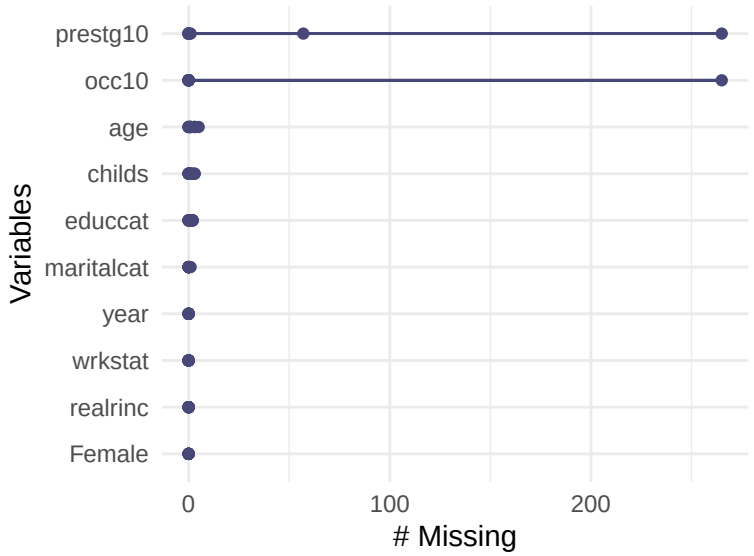

Figure 9: Missings después de las imputaciones

Los missings que quedan en las variables respecto a la totalidad de observaciones son muy pocos, por lo que optaremos por eliminarlos drásticamente.

```
wagesDS <- na.omit(wagesDS)
```

Se procede ahora a la estandarización de las variables continuas, el resultado serán variables con media 0 y desviación típica 1, cuyos valores oscilarán en el intervalo $[-3, 3]$ de la recta real

```
means <-apply(wagesDS[,vContinuas],2,mean,na.rm=TRUE)
sds<-sapply(wagesDS[,vContinuas],sd,na.rm=TRUE)

wagesDS_cont<-scale(wagesDS[,vContinuas], center = means, scale = sds)
wagesDS<-data.frame(cbind(wagesDS_cont,wagesDS[, c(vCategoricas,vDependiente)]))
```

El siguiente paso es crear las variables dummies y filtrar por las que tengan una frecuencia inferior a un punto de corte deseado, en nuestro caso como tenemos 5k observaciones tomaremos ese punto como 200.

```r
frec <- ldply(wagesDS[,vCategoricas], function (x) t(rbind(names(table(x)),table(x))))
names(frec) <- c("variable", "nivel", "frecuencia")

frec$frecuencia <- as.numeric(frec$frecuencia)

wagesDSdum <- dummy.data.frame(wagesDS, vCategoricas, sep = ".")
frec200 <- frec[frec$frecuencia < 200,]
frec200$dum <- paste(frec200$variable, frec200$nivel, sep = ".")
dumPR <- dput(frec200$dum)
wagesDSdum[, dumPR] <- NULL
```

2 Selección de variables

En primer lugar se va a realizar un stepAIC para obtener un listado de variables como candidatas, y luego se utilizará la función "step repetido binaria" para obtener ya la lista definitiva de variable importantes.

```
data <- wagesDSdum

full <- glm(Female~., data=data, family = binomial)
null <- glm(Female~1, data=data, family = binomial)
selectAIC <- stepAIC(null, scope=list(upper=full), direction = 'both', trace = FALSE)
```

Utilizando la lista obtenida, con step repetido se calculará la selección de variables definiiva.

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

listaAIC <- steprepetidobinaria(data = data, vardep = c("Female"),
            listconti = c("wrkstat.Housekeeper", "occrecode.Construction/Extraction",
                          "realrinc","occrecode.Office and Administrative Support",
                          "occ10", "maritalcat.Widowed","prestg10.48",
                          "wrkstat.Part-Time","prestg10.45", "wrkstat.Retired",
                          "occrecode.Professional", "childs.0","prestg10.47",
                          "maritalcat.Married", "occrecode.Sales",
                          "occrecode.Service","occrecode.Business/Finance",
                          "occrecode.Production", "maritalcat.Never Married",
                          "age", "year.2014","prestg10.35", "year.2006", "childs.1",
                          "prestg10.38","wrkstat.Full-Time"), sinicio = 12345,
            sfinal = 12385, porcen = 0.8, criterio = 'AIC')

vImportantesAIC <- dput(listaAIC[[2]][[1]])

listaBIC <- steprepetidobinaria(data = data, vardep = c("Female"),
            listconti = c("wrkstat.Housekeeper", "occrecode.Construction/Extraction",
                          "realrinc", "occrecode.Office and Administrative Support",
                          "occ10", "maritalcat.Widowed", "prestg10.48",
                          "wrkstat.Part-Time","prestg10.45", "wrkstat.Retired",
                          "occrecode.Professional","childs.0","prestg10.47",
                          "maritalcat.Married", "occrecode.Sales",
                          "occrecode.Service","occrecode.Business/Finance",
                          "occrecode.Production", "maritalcat.Never Married","age",
                          "year.2014", "prestg10.35", "year.2006", "childs.1",
                          "prestg10.38","wrkstat.Full-Time"), sinicio = 12345,
            sfinal = 12385, porcen = 0.8, criterio = 'BIC')

vImportantesBIC <- dput(listaBIC[[2]][[1]])
```

Ahora se realizará una valoración cruzada repetida, se calcularán las medias de errores que presenta cada modelo y con ello se mostrarán unos gráficos de cajas y bigotes para elegir al ganador.

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

data <- wagesDSdum

medias1 <- cruzadalogistica(data = data, vardep = c("Female"),
```

```
          listconti =c("wrkstat.Housekeeper", "occrecode.Construction/Extraction",
                       "realrinc", "occrecode.Office and Administrative Support",
                       "occ10", "maritalcat.Widowed", "prestg10.48",
                       "wrkstat.Part-Time","prestg10.45", "wrkstat.Retired",
                       "occrecode.Professional","childs.0","prestg10.47",
                       "maritalcat.Married", "occrecode.Sales",
                       "occrecode.Service","occrecode.Business/Finance",
                       "occrecode.Production", "maritalcat.Never Married","age",
                       "year.2014", "prestg10.35", "year.2006", "childs.1",
                       "prestg10.38","wrkstat.Full-Time"),
          listclass = c(""), grupos = 4, sinicio = 1234, repe=5)

medias1$modelo="Step_AIC_total"


medias2 <- cruzadalogistica(data = data, vardep = c("Female"), listconti=vImportantesAIC,
                            listclass = c(""), grupos = 4, sinicio = 1234, repe=5)

medias2$modelo="Step_AIC_Repetido"


medias3 <- cruzadalogistica(data = data, vardep = c("Female"), listconti=vImportantesBIC,
                            listclass = c(""), grupos = 4, sinicio = 1234, repe=5)

medias3$modelo="Step_BIC_Repetido"

union1 <- rbind(medias1, medias2, medias3)

par(cex.axis = 0.8)
boxplot(data = union1, col="#1DA1F2", tasa~modelo)
```

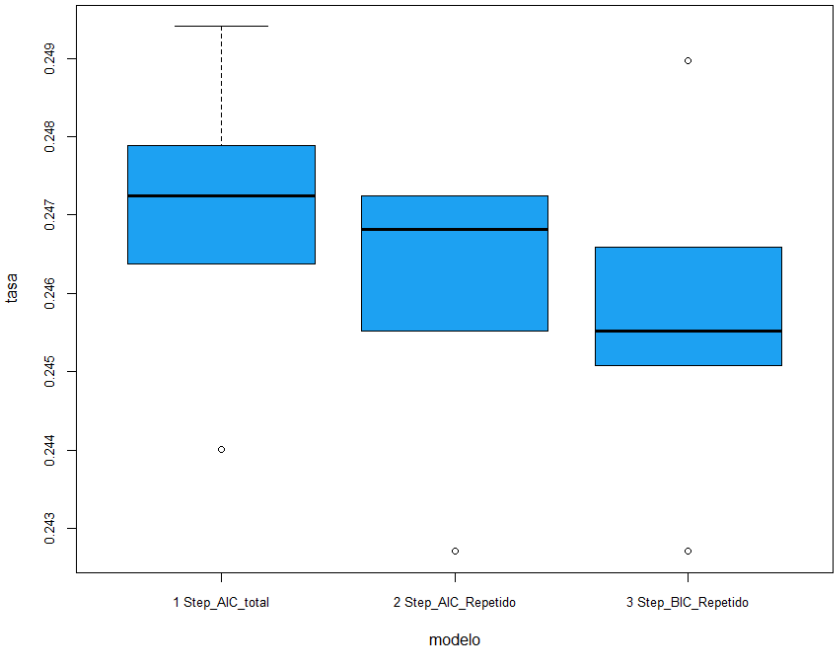

Figure 10: Comparación de los errores de los modelos utilizados para la selección de variables

En los modelos se puede analizar que el primero tiene una mayor varianza y mayor sesgo, ya que se trata del modelo inicial que utilizaba todas las variables disponibles. Por otro lado los otros 2 modelos tienen varianzas parecidas, por ello se elegirá el que tenga una tasa de error menor, es decir, el modelo ganador será el *Step Repetido BIC*.

3 Tuneado de algoritmos

3.1 Redes neuronales

En cuanto a las redes neuronales se utilizará el modelo de avNNet, se desarrollará grid con los parámetros variables del modelo, como son el número de nodos y el learning rate, para posteriormente tomar la combinación de aquellos parámetros que más favorezcan al modelo.

```
set.seed(123)

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

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

redavnnet <- train(Female~wrkstat.Housekeeper+`occrecode.Office and Administrative Support`
                   +occrecode.Service+occ10+realrinc+`wrkstat.Part-Time`+
                     maritalcat.Widowed+prestg10.47+occrecode.Professional+
                     childs.0+occrecode.Sales+`occrecode.Business/Finance`+
                     occrecode.Production+prestg10.48+age+maritalcat.Married+
                     `maritalcat.Never Married`, data = data, method="avNNet",
                   lineout = FALSE, maxit=100, trControl = control,
                   tuneGrid = avnnetgrid, repeats = 5)
```

En este caso se fija una semilla para poder realizar comparaciones sin tener el factor aleatorio de por medio, se define un trainControl usando validaciones cruzadas y se indican los parámetros que queremos comparar en el grid. Finalmente, se han detallado las variables que queremos utilizar, acordes con lo obtenido en el apartado anterior, el método que en este caso es **avNNet** y **lineout = FALSE** ya que la variable dependiente no es contínua, sino que toma los valores "Yes" y "No".
Los resultados obtenidos son los siguientes:

Model Averaged Neural Network

4627 samples 17 predictor 2 classes: 'No', 'Yes'

No pre-processing Resampling: Cross-Validated (4 fold) Summary of sample sizes: 3471, 3470, 3470, 3470 Resampling results across tuning parameters:

size decay Accuracy Kappa
5 0.001 0.7655057 0.5262224 5 0.010 0.7704768 0.5359075 5 0.100 0.7689652 0.5328297 10 0.001 0.7668037 0.5286336 10 0.010 0.7693973 0.5339123 10 0.100 0.7702620 0.5356127 15 0.001 0.7652906 0.5262064 15 0.010 0.7650758 0.5259489 15 0.100 0.7687465 0.5329960 20 0.001 0.7588070 0.5125404 20 0.010 0.7618322 0.5192379 20 0.100 0.7696134 0.5345447

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 = 5, decay = 0.01 and bag = FALSE.

Por lo que los valores del grid que proporcionan un mejor acierto del modelo son con **size = 5** y **decay = 0.01**

Se puede comparar el modelo obtenido con los modelos anteriores de la siguiente forma:

```
set.seed(123)

medias4 <- cruzadaavnnetbin(data=data, vardep = "Female",
           listconti = c("wrkstat.Housekeeper",
                         "occrecode.Office and Administrative Support",
                         "occrecode.Service", "occ10", "realrinc",
                         "wrkstat.Part-Time", "maritalcat.Widowed",
                         "prestg10.47", "occrecode.Professional", "childs.0",
                         "occrecode.Sales","occrecode.Business/Finance",
                         "occrecode.Production", "prestg10.48","age",
                         "maritalcat.Married", "maritalcat.Never Married"),
           listclass = c(""),grupos = 4, sinicio=1234, repe=5,
           repeticiones = 5, itera=100,size = c(5), decay = c(0.01))

medias4$modelo = '4 Redes AvNNet'

union2 <- rbind(medias1, medias2, medias3, medias4)

par(cex.axis = 0.8)
boxplot(data = union2, col="#1DA1F2", tasa~modelo)
```

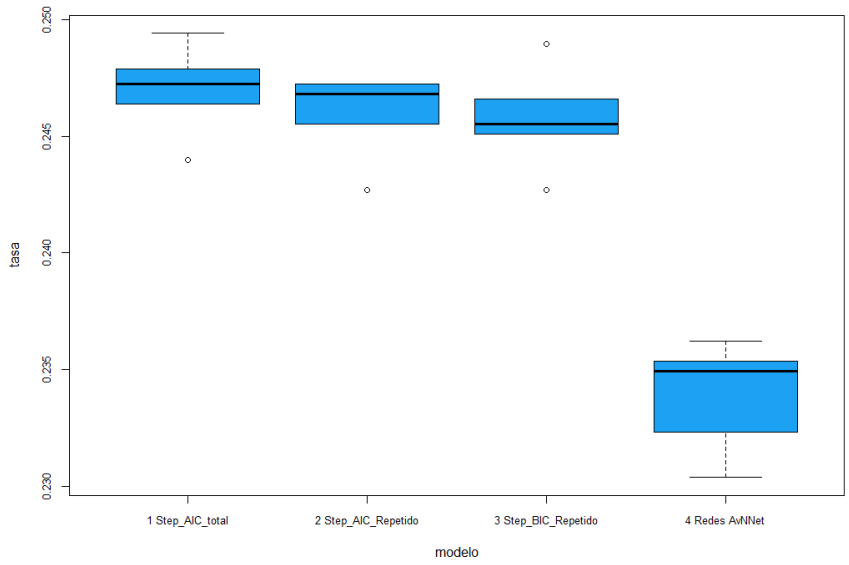

Figure 11: Comparación de modelos anteriores con el nuevo modelo de redes con 100 iteraciones

Como se puede ver a simple vista, simplemente con 100 iteraciones el modelo obtenido ya mejora de manera sustancial a los 3 modelos anteriores.

Se ha vuelto a hacer un nuevo tuneado del modelo, esta vez con 250 iteraciones en lugar de 100, para comprobar la mejora. Esta vez la combinación de parámetros más óptima ha sido size = 10, decay = 0.1 y la comparación con los modelos anteriores queda de la siguiente forma:

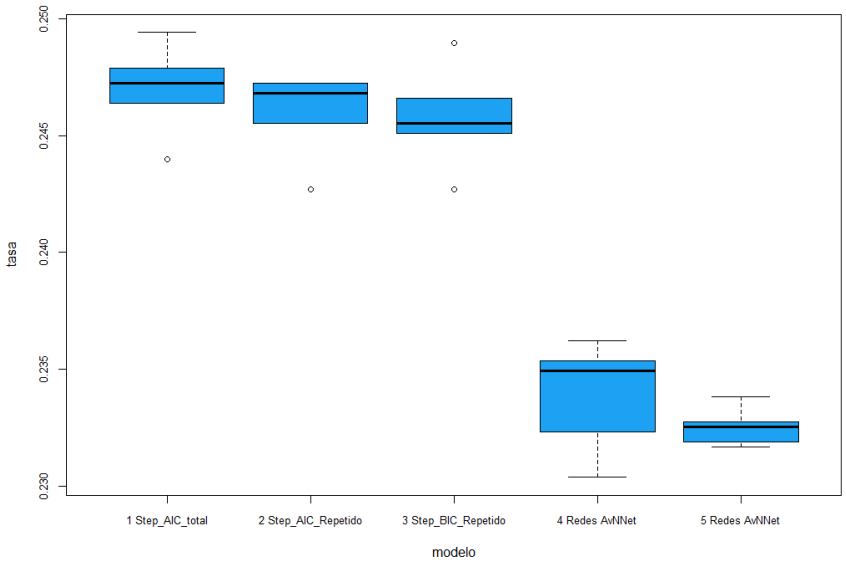

Figure 12: Comparación de modelos anteriores con el nuevo modelo de redes con 250 iteraciones

Se aprecia una gran mejora respecto al modelo anterior, sobre todo en la varianza.

En este último caso, con la fórmula para la obtención de parámetros $n(param) = h(k+1)+h+1$, donde $h = 10$ es el número de nodos y $k = 17$ el número de variables se obtiene un total de 191 parámetros, que con 4627 observaciones dan un total de 24 observaciones por parámetro.

En definitiva, el último modelo de redes es por el momento el que mejores resultados presenta, por lo que de momento sería el modelo "ganador".

3.2 Bagging, random forest

En primer lugar, se ha hecho una iteración con un grid de mtry, parámetro que indica cuántas variables se sortean para iniciar cada nodo del árbol.

```
set.seed(123)

rfgrid <- expand.grid(mtry = c(3,4,5,6,7,8,9,10,11))

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

rf <- train(Female~., data = data, method = "rf", trControl = control, tuneGrid = rfgrid,
            linout = FALSE, ntree = 300, nodesize = 10, replace = TRUE,
            importance = TRUE)
```

Los resultados finales de este árbol han sido **mtry = 9** con 0.7866855 de precisión, y recomienda un sampsize de 3470.

A pesar de que tenemos muchísimas variables, se ha realizado un estudio para saber cuales intervienen de una mejor y peor forma en el modelo y los resultados han sido que *occ10* (el código del tipo de trabajo), *wrkstat.Housekeeper* (estado de trabajo amo/a de casa), *realrinc* (ingresos anuales) y *age* (edad) son las que

aportan más accuracy al modelo; en cambio muchas variables dummies referentes a los años del estudio aportan negativamente.

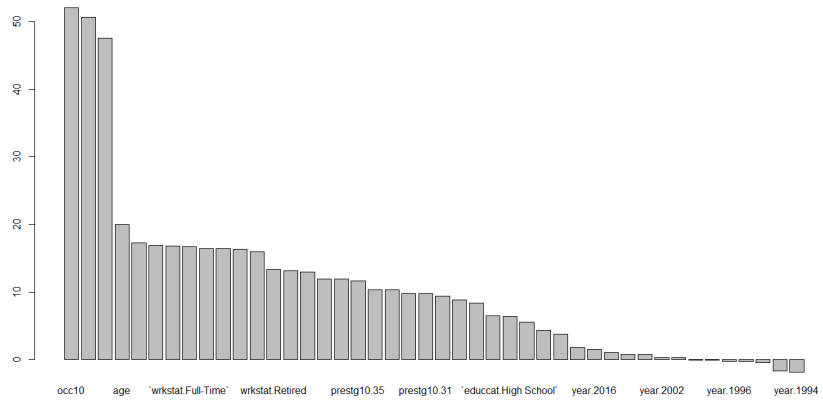

Figure 13: Importancia de Variables Random Forest

Para realizar el tuneado de ntrees, se ha utilizado el siguiente código:

```
set.seed(123)

rfbis <- randomForest(Female~., data=data, mtry = 9, ntree = 3000, nodesize = 10,
                      replace = TRUE)

plot(rfbis$err.rate[,1])
```

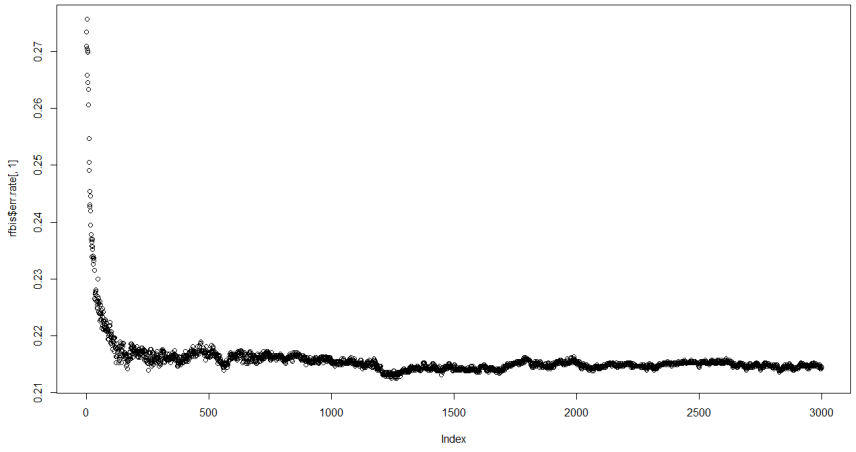

Figure 14: Variación de ntrees

Por el gráfico podemos ver que a partir de 250, 500, 1000 árboles los resultados no cambiarían.
En el siguiente código calculamos un modelo bagging, ya que en lugar de utilizar el valor de mtry sugerido, usaremos el total de variables que intervienen en el modelo, lo que implica que se el sorteo de variables en cada nodo se realizará entre todas las posibles.

```
source("cruzada rf binaria.R")

medias6 <- cruzadarfbin(data=data, vardep = "Female",
           listconti = c("wrkstat.Housekeeper",
                         "occrecode.Office and Administrative Support",
                         "occrecode.Service", "occ10", "realrinc",
                         "wrkstat.Part-Time", "maritalcat.Widowed",
                         "prestg10.47", "occrecode.Professional", "childs.0",
                         "occrecode.Sales","occrecode.Business/Finance",
                         "occrecode.Production", "prestg10.48","age",
                         "maritalcat.Married", "maritalcat.Never Married"),
           listclass = c(""), grupos = 4, sinicio=1234, repe=5,
           nodesize = 10, mtry = 17, ntree = 1000, replace = TRUE,
           sampsize = 2500)

medias6$modelo = '6 Bagging sampsize 150'

union4 <- rbind(medias1, medias2, medias3, medias4, medias5, medias6)

par(cex.axis = 0.8)
boxplot(data = union4, col="#1DA1F2", tasa~modelo)
```

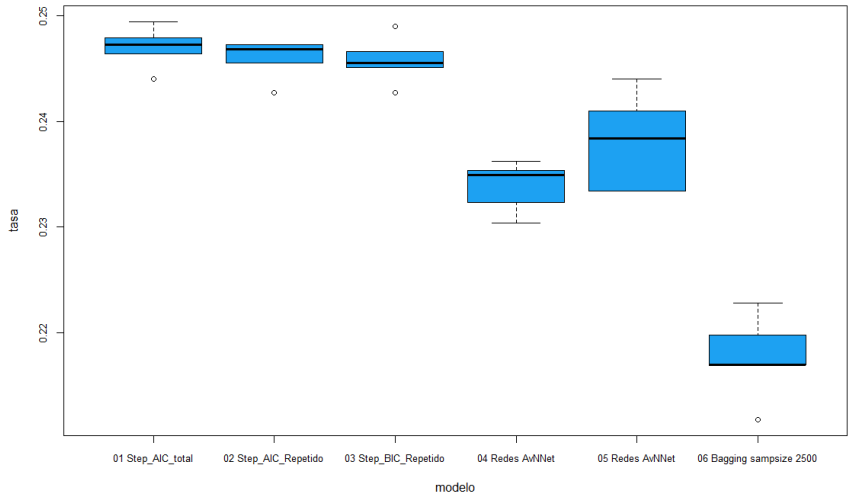

Figure 15: Comp con Bagging

Se ha vuelto a calcular el modelo para ajustar el parámetro *sampsize*, esta vez al que se había recomendado con un valor de 3470, y realizar un Random Forest en lugar de un bagging, utilizando el valor de mtry sugerido. Y los resultados comparando con los mejores modelos de cada tipo son los siguientes:

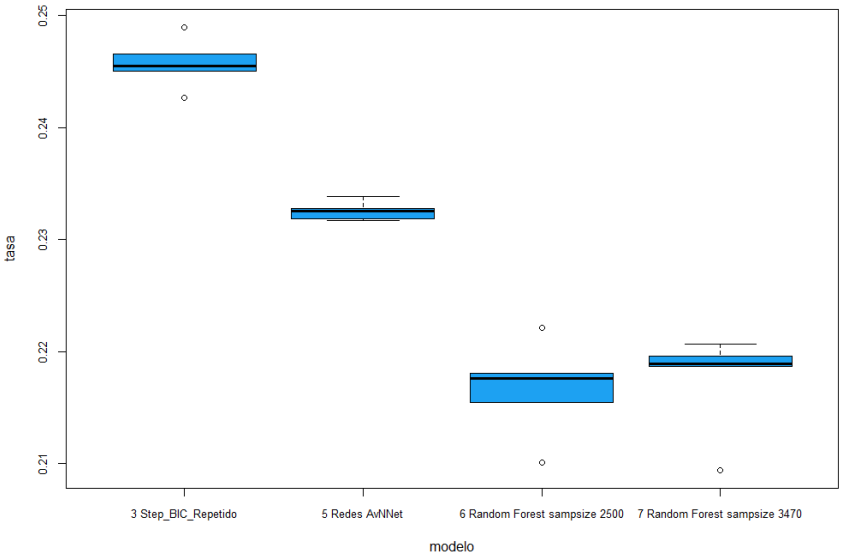

Figure 16: Comp con Random Forest sampsize = 3470

El nuevo modelo presenta una tasa de fallo un poco mayor que el anterior Random Forest, pero su varianza es muchísimo menor, por lo que realmente se considera como modelo ganador al que se ha obtenido con sampsize = 3470.

3.3 Gradient Boosting

Para tunear el modelo de gradient boosing lo que se hará en primer lugar es hallar mediante un grid el valor óptimo para los parámetros ntrees, shrinkage y minobsinnode. Para ello se ha utilizado el siguiente código:

```
set.seed(123)

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(Female~wrkstat.Housekeeper+`occrecode.Office and Administrative Support`
             +occrecode.Service+occ10+realrinc+`wrkstat.Part-Time`
             +maritalcat.Widowed+prestg10.47+occrecode.Professional
             +childs.0+occrecode.Sales+`occrecode.Business/Finance`
             +occrecode.Production+prestg10.48+age+maritalcat.Married
             +`maritalcat.Never Married`, data=data, method = "gbm",
             trControl = control, tuneGrid = gbmgrid,
             distribution = "bernoulli", bag.fraction = 1,
             verbose = FALSE)
```

Y se han obtenido que los valores para los parámetros más óptimos son n.trees = 5000, interaction.depth = 2, shrinkage = 0.05 y n.minobsinnode = 20, que devuelve un accuracy de 0.7953324

Haciendo un plot del glm se obtiene

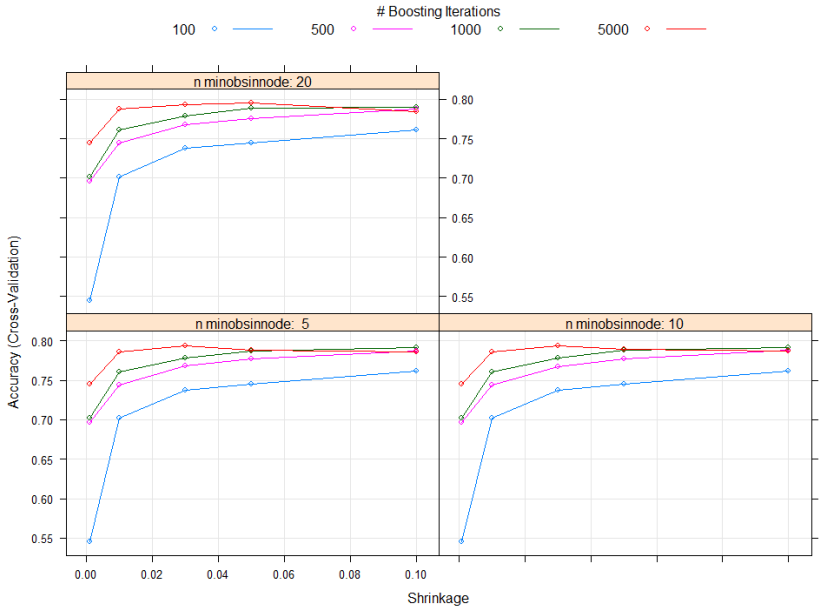

Figure 17: plot(gbm)

Se va a estudiar en profundidad el parámetro ntrees, dejando fijados los valores sugeridos anteriormente y viendo cómo es la evolución de la accuracy.

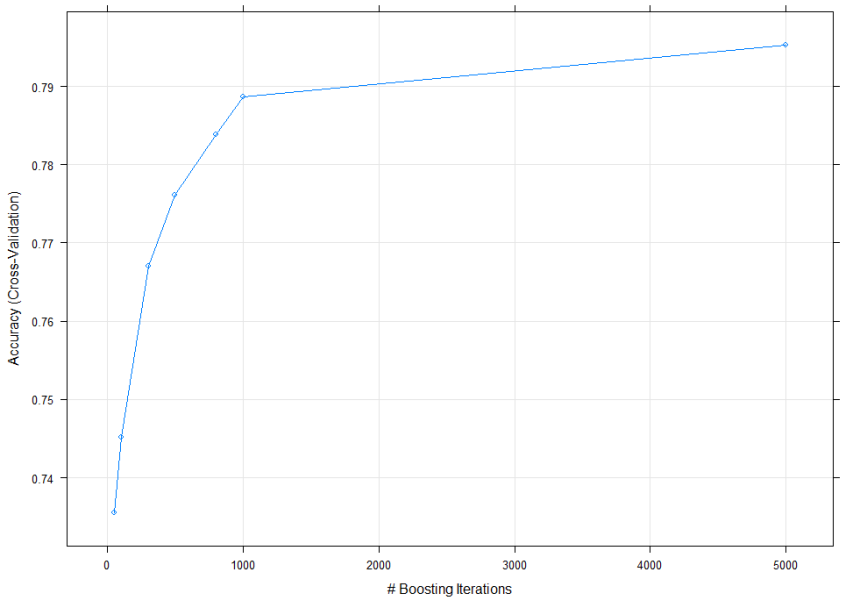

Figure 18: ntrees GBM

Como se puede apreciar, si lo que buscamos es el mayor accuracy se debe utilizar un ntrees = 5000, pero hay que tener cuidado porque un número muy elevado de árboles puede llevar al sobreajuste en el Gradient Boosting, aunque de manera muy leve.

Se va a estudiar ahora la importancia de variables en GBM, para ello hacemos un plot del summary del modelo y veremos que variables aportan más accuracy al mismo.

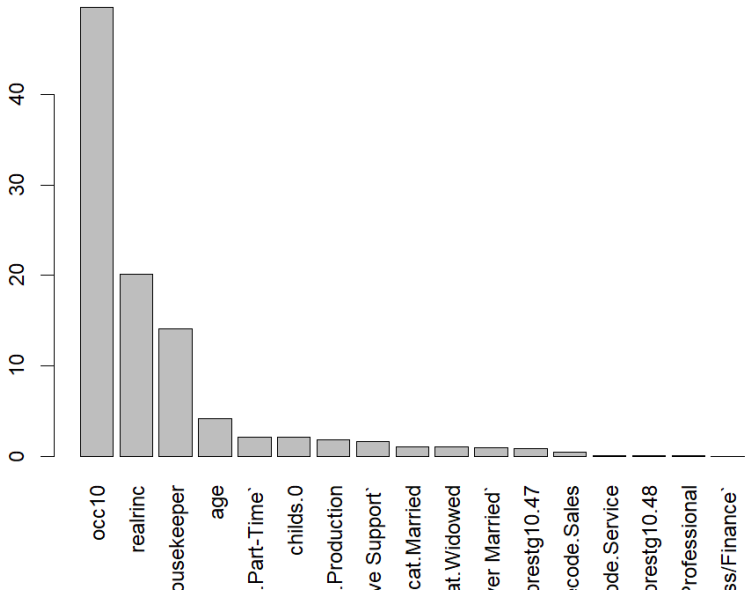

Figure 19: Importancia de Variables en GBM

Se puede ver que las variables que aportan prácticamente todo el accuracy al modelo son, como en casos anteriores, *occ10*, *realrinc*, *workstat.Houskeeper* y *age*. El resto prácticamente no aportan nada.

Si se realiza la comparación (CV repetida) con los modelos ganadores anteriores de la tasa de error (cuanto menor mejor) y del AUC (cuanto mayor mejor), se obtiene lo siguiente:

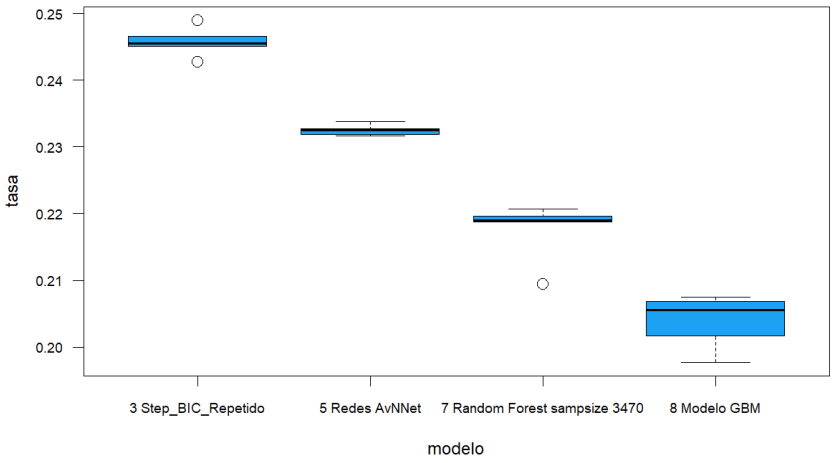

Figure 20: Tasa error con GBM

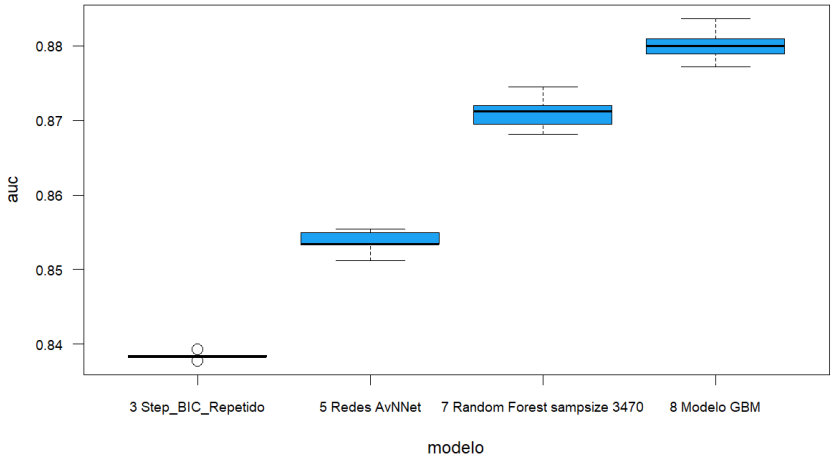

Figure 21: AUC con GBM

Finalmente, se va a probar ahora a tunear el parámetro **bag.fraction**, se va a utilizar un valor de 0.5 y ver cómo se comporta el modelo. Los resultados del grid han sido los mismos que en el modelo anterior n.trees = 5000, interaction.depth = 2, shrinkage = 0.05 y n.minobsinnode = 20, pero esta vez devuelven un accuracy de 0.7957657, ligeramente superior al que teníamos.

3.4 SVM

3.4.1 SVM lineal

En primer lugar se va a estudiar el caso lineal, donde únicamente se realizará un ajuste del parámetro C, cabe tener en cuenta que a mayor C se tendrá menor sesgo pero mayor sobreajuste.

```
set.seed(123)

SVMlin1grid <- expand.grid(C = c(0.01,0.05,0.1,0.2,0.5,1,2,5,10))

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

SVMlin1 <- train(data=data, Female~wrkstat.Housekeeper
                 +`occrecode.Office and Administrative Support`
                 +occrecode.Service+occ10+realrinc+`wrkstat.Part-Time`
                 +maritalcat.Widowed+prestg10.47+occrecode.Professional
                 +childs.0+occrecode.Sales+`occrecode.Business/Finance`
                 +occrecode.Production+prestg10.48+age+maritalcat.Married
                 +`maritalcat.Never Married`, method = "svmLinear",
                 trControl = control, tuneGrid = SVMlin1grid,
                 verbose = FALSE)

SVMlin1$results

plot(SVMlin1$results$C, SVMlin1$results$Accuracy)
```

Los resultados que se obtienen son los que siguen

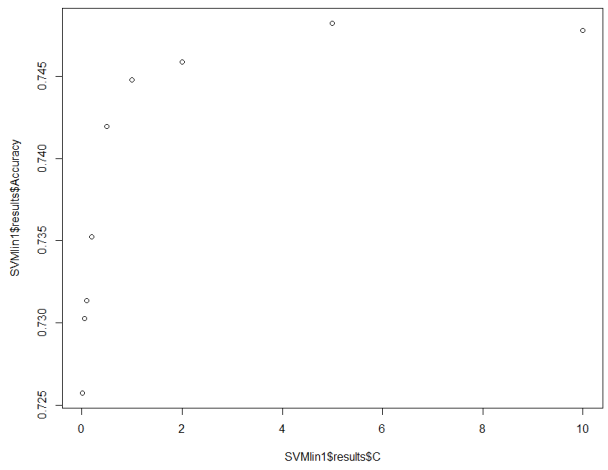

Figure 22: Accuracy para diferentes valores de C en SVL lineal

Se puede observar que para los valores introducidos en el grid en C=5 se obtiene un mayor accuracy, aún así para comprobar los valores intermedios se va a estudiar qué ocurre en el subintervalo [2,5]:

```
set.seed(123)

SVMlin2grid <- expand.grid(C = c(2,2.5,3,3.5,4,4.5,5))

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

SVMlin2 <- train(data=data, Female~wrkstat.Housekeeper+
                   `occrecode.Office and Administrative Support`+occrecode.Service
                 +occ10+realrinc+`wrkstat.Part-Time`+maritalcat.Widowed
                 +prestg10.47+occrecode.Professional+childs.0
                 +occrecode.Sales+`occrecode.Business/Finance`
                 +occrecode.Production+prestg10.48+age+maritalcat.Married
                 +`maritalcat.Never Married`, method = "svmLinear",
                 trControl = control, tuneGrid = SVMlin1grid,
                 verbose = FALSE)

SVMlin2$results

plot(SVMlin2$results$C, SVMlin2$results$Accuracy)
```

Y se obtiene

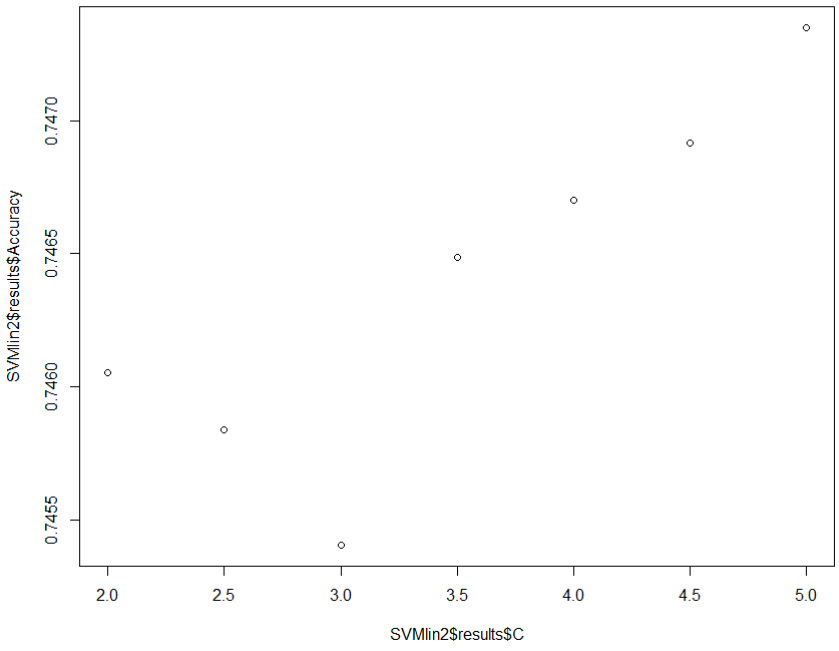

Figure 23: Accuracy para diferentes valores de C en SVL lineal

Con este último plot ya se puede fijar que el valor de C que mejor accuracy nos devuelve es C=5, por tanto este será el modelo SVM lineal final que realmente es un competidor a la regresión logística.

3.4.2 SVM Polinomial

En este caso se van a analizar tanto el parámetro C, como el grado del polinomio donde como cabe esperar a mayor grado menos sesgo pero más varianza y sobreajuste y también el parámetro de escala.

```
set.seed(123)

SVMpol1grid <- expand.grid(C = c(0.01,0.1,0.5,1,5,10), degree = c(2,3),
                           scale = c(0.1,0.5,1))

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

SVMpol1 <- train(data=data, Female~wrkstat.Housekeeper
                 +`occrecode.Office and Administrative Support`
                 +occrecode.Service+occ10+realrinc+`wrkstat.Part-Time`
                 +maritalcat.Widowed+prestg10.47+occrecode.Professional
                 +childs.0+occrecode.Sales+`occrecode.Business/Finance`
                 +occrecode.Production+prestg10.48+age+maritalcat.Married
                 +`maritalcat.Never Married`, method = "svmPoly",
                 trControl = control, tuneGrid = SVMpol1grid,
                 verbose = FALSE)

dat<-as.data.frame(SVMpol1$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)
```

Con ayuda de la librería ggplot2 se van a visualizar los resultados del modelo de una forma más clara.

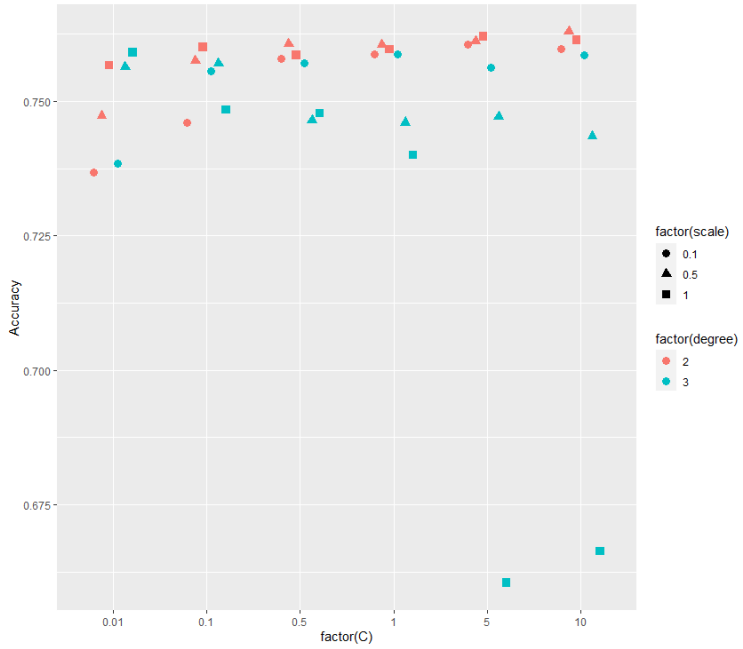

Figure 24: Accuracy para diferentes valores del grid en SVM polinómica

De está gráfica lo primero que se observa es que siempre los valores en naranja (grado 2) predominan en accuracy a los valores turquesa (grado 3), por lo que se va a filtrar los resultados únicamente por resultados de polinomios de grado 2 y así poder decidir de una manera más clara.

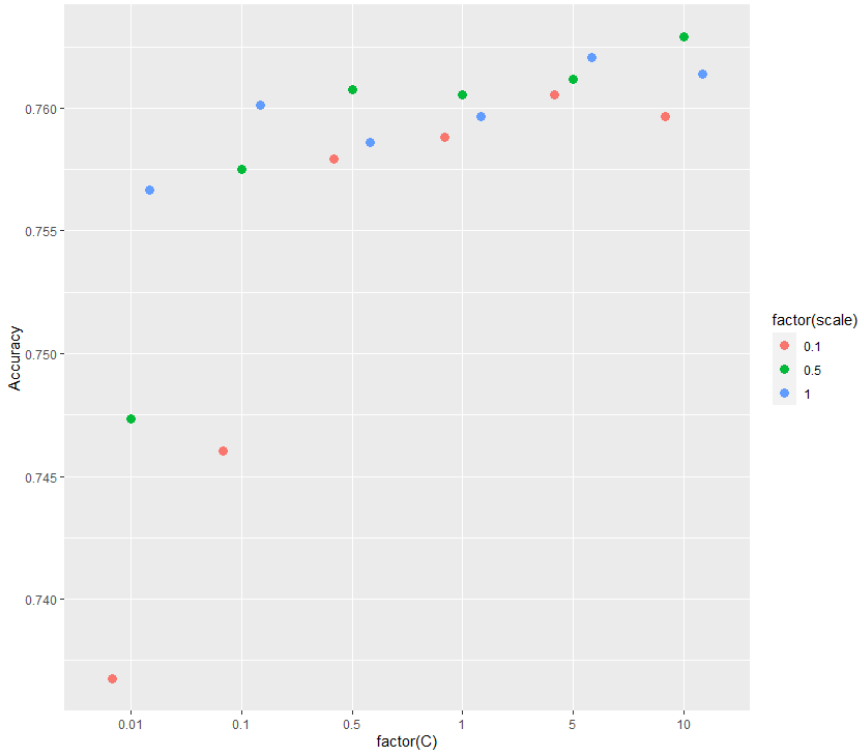

Figure 25: Accuracy para diferentes valores del grid en SVM polinómica de grado 2

Aquí tendríamos que el mayor accuracy se obtiene para un factor $C = 10$ para una escala de 0.5, aunque otros resultados también validos serían $C = 5$ y scale $= 1$ o $C = 0.5$ y scale $= 0.5$

3.4.3 Radial Basis Function (Gaussiano)

En este último modelo de SVM se estudiará la variación de los parámetros C y sigma, como suele ser costumbre, a mayor sigma tendremos menos sesgo pero mayor varianza y sobreajuste.

Para ello, se construirá un grid como hemos hecho anterioremente con algunos valores de estos parámetros y se estudiará cual es la combinación que aporta una mayor accuracy.

```
set.seed(123)

SVMrbfgrid <- expand.grid(C = c(0.01,0.1,0.5,1,5,10), sigma = c(0.01,0.1,0.5,1,5,10,30))

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

SVMrbf <- train(data=data, Female~wrkstat.Housekeeper
                +`occrecode.Office and Administrative Support`
                +occrecode.Service+occ10+realrinc+`wrkstat.Part-Time`
                +maritalcat.Widowed+prestg10.47+occrecode.Professional
```

```
                +childs.0+occrecode.Sales+`occrecode.Business/Finance`
                +occrecode.Production+prestg10.48+age+maritalcat.Married
                +`maritalcat.Never Married`, method = "svmRadial",
                trControl = control, tuneGrid = SVMrbfgrid,
                verbose = FALSE)

datRbf<-as.data.frame(SVMrbf$results)

ggplot(datRbf, aes(x = factor(C), y = Accuracy, color = factor(sigma)))
+ geom_point(position = position_dodge(width = 0.5), size = 3)
```

Como en el caso anterior del RBF polinómico, mediante ggplot2 se han visualizado los diferentes valores de accuracy para cada comibnación de C y sigma del grid. Los resultados obtenidos son los siguientes:

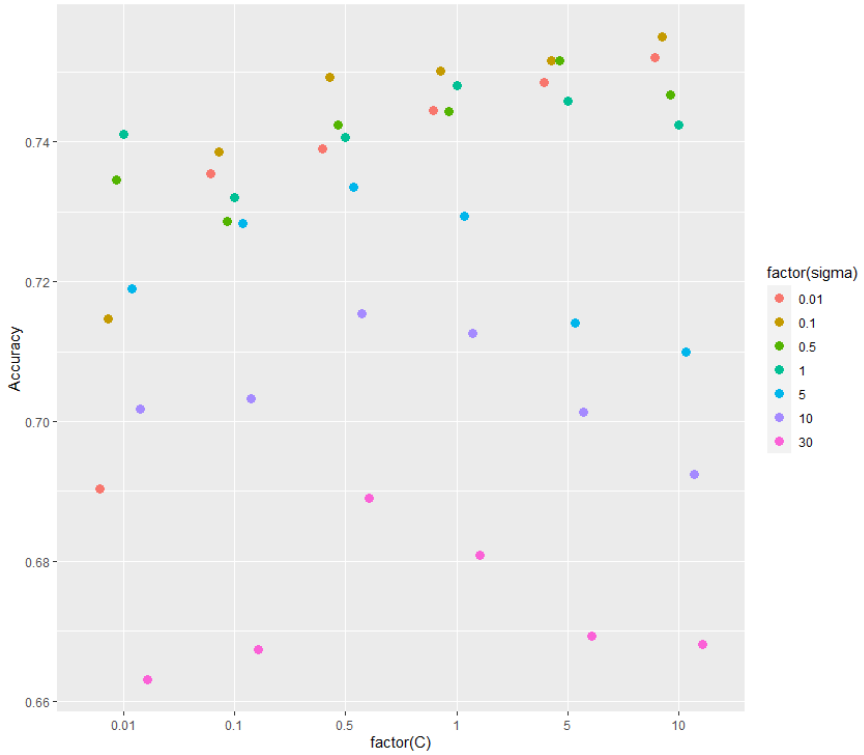

Figure 26: Accuracy para diferentes valores del grid en SVM radial

Como se puede observar, el mayor accuracy se consigue con sigma = 0.1 y C = 10.

3.4.4 Comparación

Finalmente se van a comparar, mediante validación cruzada, los 3 modelos obtenidos con los mejores modelos de cada tipo que se había desarrollado en apartados anteriores. Con el código similar al que se ha mostrado en otros puntos, y utilizando los parámetros más optimos analizados en este apartado, se obtienen los siguientes resultados en cuanto a las tasas de error:

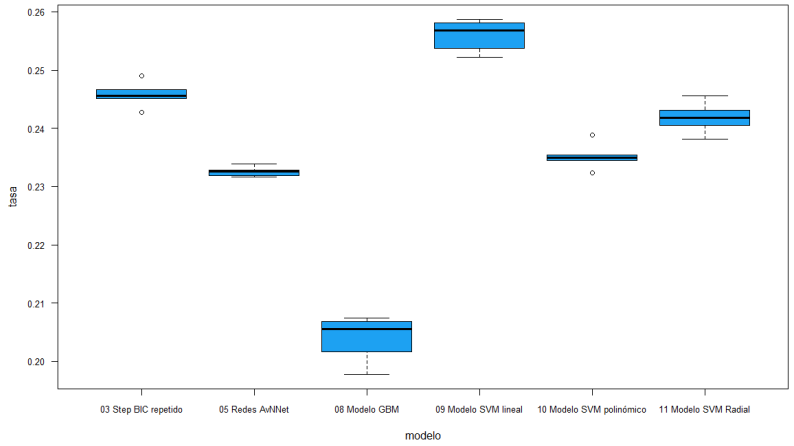

Figure 27: Comparación por la tasa de error incluyendo 3 modelos SVM

Como se puede observar, el modelo que sigue teniendo una tasa de error menor es el obtenido mediante el algoritmo GBM. Vamos a comprobar ahora el AUC, que recordemos que cuanto mayor sea indicará un mejor poder predictivo en nuestro modelo:

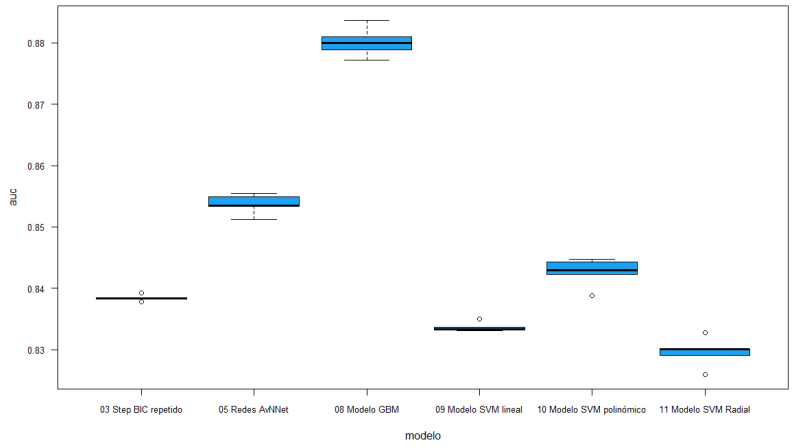

Figure 28: Comparación por el AUC de error incluyendo 3 modelos SVM

A pesar de que el mejor modelo de los 3 nuevos es el polinómico (recordemos que hemos usado grado 2), no alcanza tampoco al modelo GBM, que sin ninguna duda y fijándonos tanto en el error como el AUC, por el momento, sería el modelo ganador.

3.5 Ensamblado

La parte del código que se ha utilizado para preparar los archivos generados por los algorítmos se encontrará en el anexo final, ya que es un proceso bastante repetitivo y ocuparía mucho espacio.

Aquí vamos a partir teniendo ya las predicciones de los mejores modelos evaluados y se van a unir en un mismo dataframe y establecer combinaciones y promedios con ellas para obtener más puntuaciones de predicción:

```
uniPredi <- cbind(predi1, predi5, predi8, predi9, predi10, predi11)
uniPredi <- uniPredi[,!duplicated(colnames(uniPredi))]

columnas <- c("logi", "Redes", "GBM", "SVMlin", "SVMpoli", "SVMrbf")
combinaciones_2 <- combn(columnas, 2)
for (i in 1:ncol(combinaciones_2)) {
  col1 <- combinaciones_2[1, i]
  col2 <- combinaciones_2[2, i]
  nueva_col <- paste0("predi", i, "med_2")
  uniPredi[, nueva_col] <- (uniPredi[, col1] + uniPredi[, col2]) / 2
}

combinaciones_3 <- combn(columnas, 3)
for (i in 1:ncol(combinaciones_3)) {
  col1 <- combinaciones_3[1, i]
  col2 <- combinaciones_3[2, i]
  col3 <- combinaciones_3[3, i]
  nueva_col <- paste0("predi", i, "med_3")
  uniPredi[, nueva_col] <- (uniPredi[, col1] + uniPredi[, col2] + uniPredi[, col3]) / 3
}

combinaciones_4 <- combn(columnas, 4)
for (i in 1:ncol(combinaciones_4)) {
  col1 <- combinaciones_4[1, i]
  col2 <- combinaciones_4[2, i]
  col3 <- combinaciones_4[3, i]
  col4 <- combinaciones_4[4, i]
  nueva_col <- paste0("predi", i, "med_4")
  uniPredi[, nueva_col] <- (uniPredi[, col1] + uniPredi[, col2] + uniPredi[, col3] +
                              uniPredi[, col4]) / 4
}
```

En este trozo de código en primer lugar hemos juntado las columnas de todos nuestros archivos de predicciones y hemos eliminado duplicados, lo que ha quitado gran parte de las variables. Luego se ha indicado los nombres de las columnas que contienen predicciones y de las cuales se quiere hacer promedios, solamente hemos elegido grupos de 2, 3 y 4. Se puede hacer un cálculo rápido y observar que realmente estamos añadiendo un total de $\binom{6}{2} + \binom{6}{3} + \binom{6}{4} = 15 + 20 + 15 = 50$ variables nuevas al dataframe de predicciones.

Analizando estas predicciones con un código que se ha adjuntado en el anexo donde se construye un dataframe con la tasa de fallos y el auc promedio de ellas, obteniendo los siguientes resultados:

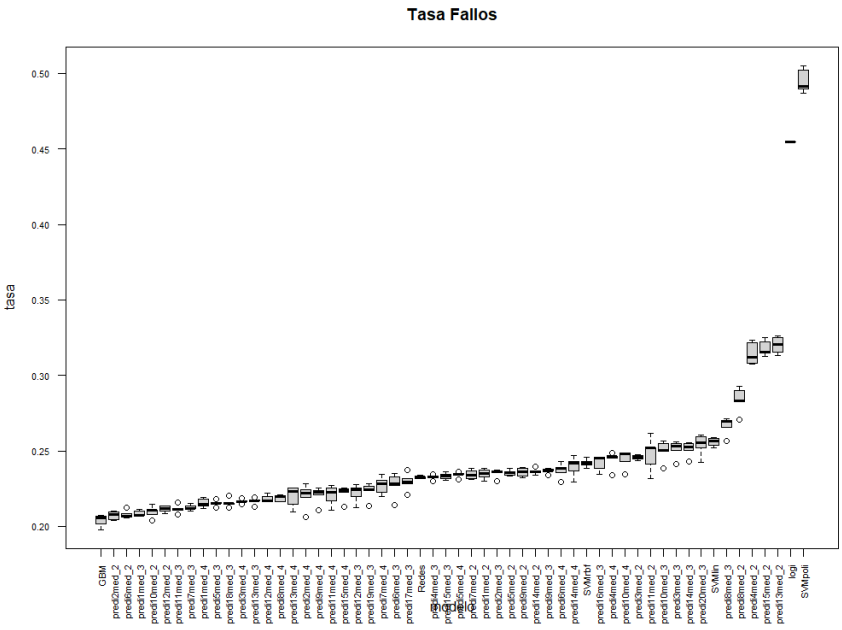

Figure 29: Tasa de Fallos Modelos

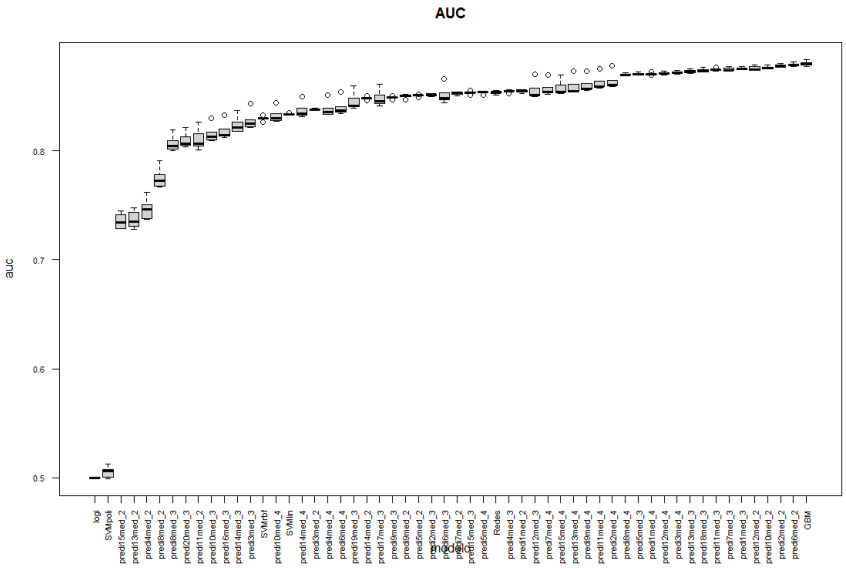

Figure 30: AUC Modelos

Por lo que se puede apreciar en ambos gráficos es que el modelo con menor tasa de fallos y mayor AUC es el GBM.

4 AutoML (h2o)

Se ha utilizado la librearía h2o para entrenar una serie de modelos sobre el dataframe que se ha usado durante todo el informe. Para realizar el entrenamiento de los modelos se ha usado el siguiente código:

```
load(file = "preparedData.Rda")
vardep <- "Female"
listconti <- c("wrkstat.Housekeeper",
               "occrecode.Office and Administrative Support",
               "occrecode.Service", "occ10", "realrinc",
               "wrkstat.Part-Time", "maritalcat.Widowed",
               "prestg10.47", "occrecode.Professional",
               "childs.0", "occrecode.Sales",
               "occrecode.Business/Finance", "occrecode.Production",
               "prestg10.48", "age", "maritalcat.Married",
               "maritalcat.Never Married")
listclass <- c("")



h2o.init(nthreads=8)


data$Female <- as.factor(data$Female)
train<- as.h2o(data)

start_time <- Sys.time()

aml <- h2o.automl(x = 2:9,y=1,training_frame = train,max_models = 20,seed = 1,
                  keep_cross_validation_predictions=TRUE)

lb <- aml@leaderboard
print(lb, n = nrow(lb))

aml@leader
end_time <- Sys.time()

end_time - start_time
```

A continuación se van a revisar algunos modelos entrenados por h2o.

4.1 Logística (GLM)

```
autoMLlogi <- h2o.getModel("GLM_1_AutoML_1_20230716_121335")
str(autoMLlogi)
autoMLlogi@allparameters
```

Se ha obtenido la siguiente tabla de coeficientes que conforman el modelo logístico generado por AutoML:

Coefficients: glm coefficients names coefficients standardized_coefficients 1 Intercept -0.001155 0.000000 2 realrinc 0.025408 0.025408 3 occ10 -0.010003 -0.010003 4 year.1994 -0.003813 -0.000806 5 year.1996 -0.000377 -0.000084 6 year.1998 -0.016825 -0.003495 7 year.2000 0.027996 0.005881 8 year.2002 0.094045 0.019354 9 year.2004 -0.073431 -0.015321

4.2 GBM (Modelo Ganador)

```
autoMLgbm <- h2o.getModel("GBM_grid_1_AutoML_1_20230716_121335_model_4")
autoMLgbm@allparameters

gbm1<-h2o.gbm(x = 2:9,y=1,training_frame = train,seed=23,
              ntrees=54,max_depth=17,min_rows = 30,sample_rate = 0.7,col_sample_rate = 1,
              col_sample_rate_per_tree = 0.4,nfolds=5)
```

En primer lugar se ha visto qué parámetros del algoritmo de GBM (modelo ganador) tiene en el modelo de AutoML, con dichos parámetros se ha generado un nuevo modelo y se han obtenido los siguientes resultados realizando validación cruzada:

H2ORegressionMetrics: gbm
** Reported on cross-validation data.
5-fold cross-validation on training data (Metrics computed for combined holdout predictions) **

MSE: 0.8106121
RMSE: 0.90034
MAE: 0.7503283
RMSLE: NaN
Mean Residual Deviance : 0.8106121

	Mean	SD	cv_1_valid	cv_2_valid	cv_3_valid	cv_4_valid	cv_5_valid
mae	0.750276	0.015017	0.751157	0.751224	0.727128	0.769220	0.752649
mean_residual_deviance	0.810753	0.025595	0.800048	0.812624	0.774180	0.841637	0.825277
mse	0.810753	0.025595	0.800048	0.812624	0.774180	0.841637	0.825277
r2	0.188594	0.015235	0.186952	0.211970	0.193232	0.177656	0.173162
residual_deviance	0.810753	0.025595	0.800048	0.812624	0.774180	0.841637	0.825277
rmse	0.900328	0.014245	0.894454	0.901456	0.879875	0.917408	0.908448
rmsle	NA	0.000000	NA	NA	NA	NA	NA

5 Análisis, decisiones y conclusiones

5.1 Modelo ganador

Sin ninguna duda, el modelo que mejor ajusta nuestros datos por tener un valor de AUC mayor y también ser el que menos tasa de fallo tiene es el modelo Gradient Boosting o (GBM), como se ha podido observar a lo largo de todo el estudio, desde que se ha obtenido siempre ha estado por encima del resto de modelos, alcanzando un nivel de accuracy de prácticamente 0.8, lo cual es una tasa de acierto bastante elevada.

Una vez tenemos el modelo calculado, podemos visualizar su matriz de confusión de la siguiente forma:

```
confusionMatrix(gbm$pred$pred, gbm$pred$obs)

Confusion Matrix and Statistics

          Reference
Prediction   No  Yes
       No  1598  442
       Yes  505 2082

               Accuracy : 0.7953
                 95% CI : (0.7834, 0.8069)
    No Information Rate : 0.5455
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.5862

 Mcnemar's Test P-Value : 0.04393


            Sensitivity : 0.7599
            Specificity : 0.8249
         Pos Pred Value : 0.7833
         Neg Pred Value : 0.8048
             Prevalence : 0.4545
         Detection Rate : 0.3454
   Detection Prevalence : 0.4409
      Balanced Accuracy : 0.7924

       'Positive' Class : No
```

¿Qué ocurre con la sensibilidad, especificidad y la precisión?

- Sensibilidad o ratio de Verdaderos Positivos: Hace referencia a obtener resultados positivos cuando realmente el resultado es positivo. Se ha obtenido un 0.75 y se calcula de la siguiente forma: $\frac{VP}{VP+FN}$, donde VP son los verdaderos positivos y FN los falsos negativos.
- Especificidad o ratio de Verdaderos Negativos: Se encarga de ofrecer una proporción de detección de verdaderos negativos. Se ha obtenido un 0.82 y se calcula de la siguiente forma: $\frac{VN}{VB+FP}$
- Precisión o valor predictivo positivo: Nos muestra cuanto valor tiene la predicción positiva de nuestro test, es decir cuantos verdaderos positivos hay entre el total de positivos (verdaderos positivos + falsos positivos). Se ha obtenido un 0.78 y se puede calcular de la siguiente forma: $\frac{VP}{VP+FP}$

5.2 Tabla de parámetros de la logistica

Se realiza un análisis en profundidad de los resultados obtenidos en el modelo logístico, para destacar que variables son significativas y cómo han afectado al modelo.

```
data$Female <- as.factor(data$Female)
data$Female<-relevel(data$Female,ref="No")

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

logi<- train(factor(Female)~wrkstat.Housekeeper
             +`occrecode.Office and Administrative Support`
             +occrecode.Service+occ10+realrinc+`wrkstat.Part-Time`
             +maritalcat.Widowed+prestg10.47+occrecode.Professional
             +childs.0+occrecode.Sales+`occrecode.Business/Finance`
             +occrecode.Production+prestg10.48+age+maritalcat.Married
             +`maritalcat.Never Married`
             ,data=data,method="bayesglm",trControl=control)

summary(logi)


Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.2310  -0.9013   0.1831   0.8586   4.2127

Coefficients:
                                                  Estimate Std. Error z value Pr(>|z|)
(Intercept)                                      -3.98403    0.29500 -13.505  < 2e-16 ***
wrkstat.Housekeeper                               3.33783    0.23954  13.934  < 2e-16 ***
`\\`occrecode.Office and Administrative Support\\``  4.78489    0.26158  18.292  < 2e-16 ***
occrecode.Service                                 5.03533    0.32836  15.335  < 2e-16 ***
occ10                                             1.47755    0.17062   8.660  < 2e-16 ***
realrinc                                         -0.44067    0.05971  -7.380 1.58e-13 ***
`\\`wrkstat.Part-Time\\``                         0.75221    0.11844   6.351 2.14e-10 ***
maritalcat.Widowed                                0.66883    0.16923   3.952 7.75e-05 ***
prestg10.47                                       0.88967    0.20915   4.254 2.10e-05 ***
occrecode.Professional                            6.11400    0.43441  14.074  < 2e-16 ***
childs.0                                         -0.39678    0.09719  -4.083 4.45e-05 ***
occrecode.Sales                                   4.23706    0.29529  14.349  < 2e-16 ***
`\\`occrecode.Business/Finance\\``                6.79497    0.55109  12.330  < 2e-16 ***
occrecode.Production                              1.69878    0.16447  10.329  < 2e-16 ***
prestg10.48                                       0.77574    0.19077   4.066 4.77e-05 ***
age                                              -0.18924    0.04444  -4.258 2.06e-05 ***
maritalcat.Married                               -0.54409    0.10017  -5.432 5.58e-08 ***
`\\`maritalcat.Never Married\\``                 -0.62283    0.13296  -4.684 2.81e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 6376.0  on 4626  degrees of freedom
Residual deviance: 4538.9  on 4609  degrees of freedom
```

```
AIC: 4574.9

Number of Fisher Scoring iterations: 8
```

Se ha obtenido que todas las variables del modelo son significativas. Los signos de las variables son realmente importantes, y nos indican que aquellas que tienen signos positivos como *wrkstat.Housekeeper* o *occrecode.Office and Administrative Support* aumentan la probabilidad de que el sexo del individuo sea femenino, ocurre en caso contrario en variables con signo negativo, como *realrinc* o *age*.

Podemos obtener el número de observaciones que tiene la clase minoritaria de la siguiente forma

```
sum(ifelse(data$Female == "No", 1, 0))
```

Lo cual devuelve un total de 2103 observaciones, por lo que con 17 parámetros en la logística queda un total de 123 observaciones de la clase minoritaria por parámetro.

5.3 Árbol simple

Se ha utilizado el siguiente código para generar el árbol de clasificación, se quiere destacar que por la cantidad de variables y las combinaciones entre ellas que esto implica, se ha utilizado minbucket = 100, para una mejor visualización del árbol.

```
library(rpart)
library(rpart.plot)
arbol <- rpart(factor(Female) ~ wrkstat.Housekeeper
               +`occrecode.Office and Administrative Support`
               +occrecode.Service+occ10+realrinc+`wrkstat.Part-Time`
               +maritalcat.Widowed+prestg10.47+occrecode.Professional
               +childs.0+occrecode.Sales+`occrecode.Business/Finance`
               +occrecode.Production+prestg10.48+age+maritalcat.Married
               +`maritalcat.Never Married`, data = data,
                minbucket =100,method = "class",
               parms=list(split="gini"),cp=0)

rpart.plot(arbol,extra=105,nn=TRUE)
```

También cabría destacar qque algunas variable continuas como realrinc y occ10 se encuentran estandarizadas, es por ello que los valores que se encuentran en el árbol están relativizados (media 0 y desv. típica 1), no son los valores "reales" de la variable; aún así nos sirven para entender qué tipo de observaciones se están dividiendo en cada nodo. \ Por ejemplo, en el primer nodo se dividen los valores de occ10 mayores (y menores) a 0.54, lo que serían los valores por encima de media desviación típica de la media, y su conjunto complementario.

5.4 Visualpred

Se han realizado 2 gráficos visualpred, uno con FAMD y otro con MCA. Los resultados obtenidos han sido los siguientes:

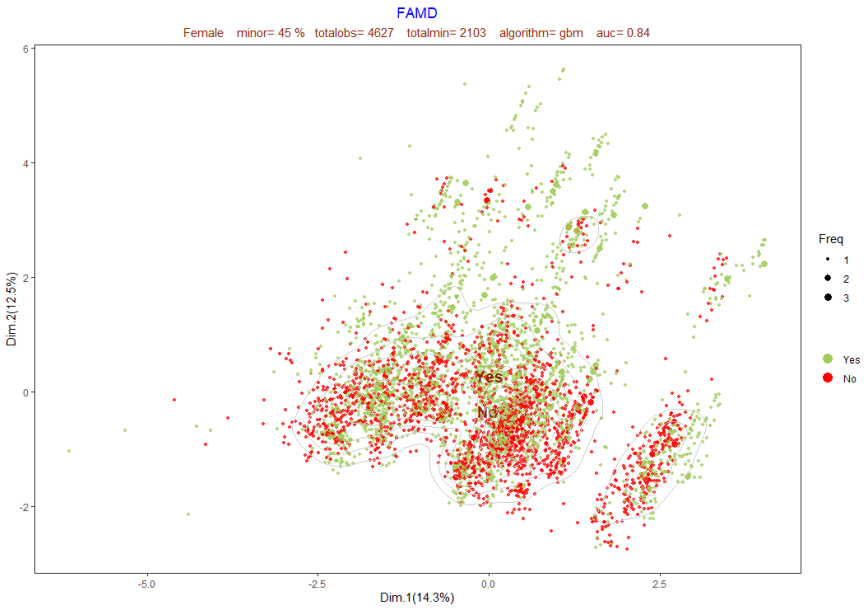

Figure 31: FAMD

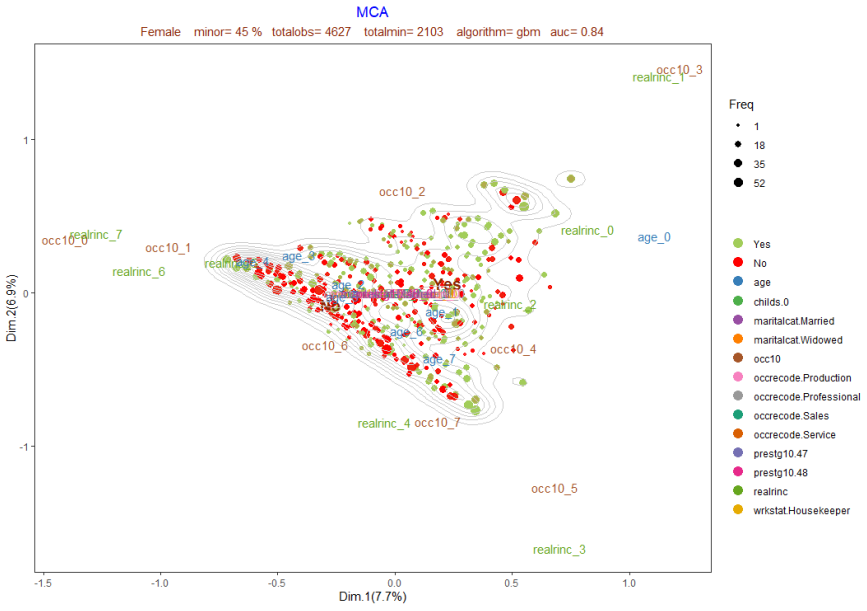

Figure 32: MCA

Ambas visualizaciones se han realizado utilizando el algorítmo GBM, el cual había salido ganador en nuestros análisis. Gracias a estas visualizaciones se pueden identificar rápidamente valores outliers y obtener una clasificación, igual que se ha realizado en pasos previos, pero esta vez de forma visual, y permite entender a simple vista como interaccionan los diferentes parámetros con los modelos.

6 Anexo

6.1 Código preparación de medias

```
set.seed(123)

vardep <- "Female"
listconti <- c("wrkstat.Housekeeper",
               "occrecode.Office and Administrative Support",
               "occrecode.Service", "occ10", "realrinc",
               "wrkstat.Part-Time", "maritalcat.Widowed",
               "prestg10.47", "occrecode.Professional",
               "childs.0", "occrecode.Sales",
               "occrecode.Business/Finance", "occrecode.Production",
               "prestg10.48", "age", "maritalcat.Married",
               "maritalcat.Never Married")
listclass <- c("")
grupos <- 4
sinicio <- 1234
repe <- 5


set.seed(123)

medias1 <- cruzadalogistica(data = data, vardep = vardep, listconti = listconti,
                            listclass = listclass, grupos = grupos,
                            sinicio = sinicio, repe = repe)
medias1bis <- as.data.frame(medias1[1])
medias1bis$modelo <- "01 Logistica"
predi1 <- as.data.frame(medias1[2])
predi1$logi <- predi1$Yes



medias5 <- cruzadaavnnetbin(data = data, vardep = vardep, listconti = listconti,
                            listclass = listclass, grupos = grupos,
                            sinicio = sinicio, repe = repe, itera=100,
                            size = c(10), decay = c(0.1))
medias5bis <- as.data.frame(medias5[1])
medias5bis$modelo <- "05 Redes"
predi5 <- as.data.frame(medias5[2])
predi5$Redes <- predi5$Yes


medias8 <- cruzadagbmbin(data = data, vardep = vardep, listconti = listconti,
                         listclass = listclass, grupos = grupos,
                         sinicio = sinicio, repe = repe,
                         shrinkage = 0.05,
                         n.minobsinnode = 20, n.trees = 5000,
                         interaction.depth = 2)
medias8bis <- as.data.frame(medias8[1])
medias8bis$modelo <- "08 GBM"
```

```
predi8 <- as.data.frame(medias8[2])
predi8$GBM <- predi8$Yes


medias9 <- cruzadaSVMbin(data = data, vardep = vardep, listconti = listconti,
                         listclass = listclass, grupos = grupos,
                         sinicio = sinicio, repe = repe, C = 5)
medias9bis <- as.data.frame(medias9[1])
medias9bis$modelo <- "09 SVM lin"
predi9 <- as.data.frame(medias9[2])
predi9$SVMlin <- predi9$Yes


medias10 <- cruzadaSVMbinPoly(data = data, vardep = vardep,
                              listconti = listconti, listclass = listclass,
                              grupos = grupos,sinicio = sinicio,
                              repe = repe, C = 10, degree = 2,
                              scale = 0.5)
medias10bis <- as.data.frame(medias10[1])
medias10bis$modelo <- "10 SVM poli"
predi10 <- as.data.frame(medias10[2])
predi10$SVMpoli <- predi10$Yes


medias11 <- cruzadaSVMbinRBF(data = data, vardep = vardep, listconti = listconti,
                             listclass = listclass, grupos = grupos,
                             sinicio = sinicio, repe = repe, C = 10,
                             sigma = 0.1)
medias11bis <- as.data.frame(medias11[1])
medias11bis$modelo <- "11 SVM RBF"
predi11 <- as.data.frame(medias11[2])
predi11$SVMrbf <- predi11$Yes


unionEns <- rbind(medias1bis, medias5bis, medias8bis, medias9bis, medias10bis, medias11bis)
```

6.2 Código Tasa de Fallos y AUC

```
listado <- c("logi","Redes","GBM","SVMlin","SVMpoli","SVMrbf","predi1med_2",
             "predi2med_2", "predi3med_2", "predi4med_2", "predi5med_2", "predi6med_2",
             "predi7med_2", "predi8med_2", "predi9med_2", "predi10med_2",
             "predi11med_2", "predi12med_2", "predi13med_2", "predi14med_2",
             "predi15med_2", "predi1med_3", "predi2med_3", "predi3med_3",
             "predi4med_3", "predi5med_3", "predi6med_3", "predi7med_3", "predi8med_3",
             "predi9med_3", "predi10med_3", "predi11med_3", "predi12med_3",
             "predi13med_3", "predi14med_3", "predi15med_3", "predi16med_3",
             "predi17med_3", "predi18med_3", "predi19med_3", "predi20med_3",
             "predi1med_4", "predi2med_4", "predi3med_4", "predi4med_4", "predi5med_4",
             "predi6med_4", "predi7med_4", "predi8med_4", "predi9med_4", "predi10med_4",
             "predi11med_4", "predi12med_4", "predi13med_4", "predi14med_4",
             "predi15med_4")


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)
}

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$modelo <- prediccion
    auc <- auc(archi$obs, ordered(archi$proba))
    t$auc <- auc
    medias0<-rbind(medias0,t)

  }
```

```
}


medias0$modelo <- with(medias0, reorder(modelo, tasa, mean))
par(cex.axis = 0.7, las = 2)
boxplot(data = medias0, tasa~modelo,main = "Tasa Fallos")


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