Práctica Machine Learning

Esteban Joaquín Jiménez Párraga

Introducción

Para esta práctica de R trataremos de aplicar varios algoritmos de Machine Learning a una clasificación binaria en una base de datos que hemos extraído de https://vincentarelbundock.github.io/Rdatasets/datasets.html.

El proceso que vamos a seguir será primero comprender y depurar los datos de nuestra base de datos. Haremos una selección de variables a partir de la que construiremos un modelo de regresión logística que nos servirá de base para futuros modelos. A partir de aquí pasamos a los algoritmos de Machine Learning propiamente dichos: en primer lugar con algoritmos derivados de árboles de decisión, luego redes neuronales y por último máquinas de vector soporte.

Finalmente, practicaremos dos métodos de ensamblado (bagging y stacking) y haremos un análisis de nuestro modelo ganador.

Preparación de datos

Comenzamos cargando todas las librerías que vamos a utilizar

Con esto ya podemos presentar nuestra base de datos. La base de datos escogida recopila información sobre comportamientos de riesgo en jovenes con variables que indican información en sí sobre el individuo (edad, género, altura, peso...) y otras variables que nos sirven para categorizar su comportamiento (si lleva casco al montar en bici, si hace deporte regularmente...). A continuación definimos las variables detalladamente:

- age: edad (Categorica)
- gender: Género (Categorica)
- grade: Año escolar en el que se encuentra el sujeto (Categorica)
- hispanic: Si el sujeto es hispano o no (Categorica)
- race: raza del sujeto (Categorica)
- height: altura (numerica)
- weight: peso(numerica)
- helmet_12m: Frecuencia con la que el sujeto a usado casco al ir en bicicleta (Categorica)
- text_while_driving_3d: Cuantos días ha escrito mensajes mientras conducía el sujeto en el último mes (Categórica)
- physically_active_7d: Cuantos días ha sido activo (actividad fisica de +60 mins) el sujeto en la última semana (Categórica)
- hours_tv_per_school_day: Cuantas horas ve la televisión el sujeto un día de escuela en la noche (Categórica)
- strength_training_7d: Cuantos días ha hecho entrenamiento de fuerza en la última semana (Categórica)
- school_night_hours_sleep: Cuantas horas duerme por las noches durante los días de escuela (Categórica)

En esta práctica tomaremos la variable gender y trataremos de hacer predicciones sobre ella.

```
#Cargamos nuestra base de datos
youth_risk <-
  read.csv("~/Documents/Max 8/MasterBD/Practica Machine Learning con R/yrbss.csv")
#borramos la columna X que es el número de observación
youth_risk <- youth_risk[ , -1]
```

Vamos a generar un dataset depurado con nuevas categorías si lo consideramos necesario y sin datos perdidos tras realizar imputación sobre los datos. Comenzamos estableciendo una tipología de factor para las variables categóricas para poder trabajar con ellas adecuadamente.

```
#Para tener una tipología correcta de las variables
#empezamos poniendo las v. categóricas como factor
cols_to_convert <- !is.factor(youth_risk) & !names(youth_risk) %in% c("height",
                                                                      "weight")
youth_risk[cols_to_convert] <- lapply(youth_risk[cols_to_convert], as_factor)

str(youth_risk)
```

```
## 'data.frame':    13583 obs. of  13 variables:
##  $ age                   : Factor w/ 7 levels "12","13","14",..: 3 3 4 4 4 4 4 3 4 4 ...
##  $ gender                : Factor w/ 2 levels "female","male": 1 1 1 1 1 1 1 2 2 2 ...
##  $ grade                 : Factor w/ 5 levels "9","10","12",..: 1 1 1 1 1 1 1 1 1 2 ...
##  $ hispanic              : Factor w/ 2 levels "not","hispanic": 1 1 2 1 1 1 1 1 1 1 ...
```

```
##  $ race                    : Factor w/ 5 levels "Black or African American",..: 1 1 2 1 1 1 1 1 1 1
##  $ height                  : num  NA NA 1.73 1.6 1.5 1.57 1.65 1.88 1.75 1.37 ...
##  $ weight                  : num  NA NA 84.4 55.8 46.7 ...
##  $ helmet_12m              : Factor w/ 6 levels "never","did not ride",..: 1 1 1 1 2 2 2 1 1 2 ...
##  $ text_while_driving_30d  : Factor w/ 8 levels "0","30","did not drive",..: 1 NA 2 1 3 3 NA NA NA NA
##  $ physically_active_7d    : Factor w/ 8 levels "0","1","2","3",..: 5 3 8 1 3 2 5 5 6 1 ...
##  $ hours_tv_per_school_day : Factor w/ 7 levels "5+","2","3","do not watch",..: 1 1 1 2 3 1 1 1 1 4
##  $ strength_training_7d    : Factor w/ 8 levels "0","1","2","3",..: 1 1 1 1 2 1 3 1 4 1 ...
##  $ school_night_hours_sleep: Factor w/ 7 levels "8","6","<5","9",..: 1 2 3 2 4 1 4 2 3 3 ...
```

```r
#Veamos un summary de los datos
summary(youth_risk)
```

```
##       age          gender       grade          hispanic
##  17     :3473   female:6621   9    :3588   not     :9928
##  16     :3203   male  :6950   10   :3152   hispanic:3424
##  15     :3098   NA's  :  12   12   :3557   NA's    : 231
##  18     :2320                  11   :3184
##  14     :1368                  other:  23
##  (Other):  44                  NA's :  79
##  NA's   :  77
##                                        race          height
##  Black or African American                :3229   Min.   :1.270
##  Native Hawaiian or Other Pacific Islander: 258   1st Qu.:1.600
##  American Indian or Alaska Native         : 323   Median :1.680
##  White                                    :6416   Mean   :1.691
##  Asian                                    : 552   3rd Qu.:1.780
##  NA's                                     :2805   Max.   :2.110
##                                                   NA's   :1004
##      weight              helmet_12m     text_while_driving_30d
##  Min.   : 29.94   never       :6977   0            :4792
##  1st Qu.: 56.25   did not ride:4549   did not drive:4646
##  Median : 64.41   sometimes   : 341   1-2          : 925
##  Mean   : 67.91   always      : 399   30           : 827
##  3rd Qu.: 76.20   rarely      : 713   3-5          : 493
##  Max.   :180.99   most of time: 293   (Other)      : 982
##  NA's   :1004     NA's        : 311   NA's         : 918
##  physically_active_7d hours_tv_per_school_day strength_training_7d
##  7      :3622         2           :2705       0      :3632
##  0      :2172         <1          :2168       7      :2085
##  5      :1728         3           :2139       3      :1468
##  3      :1451         do not watch:1840       5      :1333
##  2      :1270         1           :1750       2      :1305
##  (Other):3067         (Other)     :2643       (Other):2584
##  NA's   : 273         NA's        : 338       NA's   :1176
##  school_night_hours_sleep
##  7      :3461
##  8      :2692
##  6      :2658
##  5      :1480
##  <5     : 965
##  (Other):1079
##  NA's   :1248
```

Observamos que en algunas variables tenemos bastantes datos missing. Veamos las categorías que tenemos en cada variable.

```
unique(youth_risk$grade)
```

```
## [1] 9     10    12    11    <NA>  other
## Levels: 9 10 12 11 other
```

```
unique(youth_risk$age)
```

```
## [1] 14   15   16   <NA> 18   17   12   13
## Levels: 12 13 14 15 16 17 18
```

```
unique(youth_risk$helmet_12m)
```

```
## [1] never        did not ride <NA>         sometimes    always
## [6] rarely       most of time
## Levels: never did not ride sometimes always rarely most of time
```

```
unique(youth_risk$text_while_driving_30d)
```

```
## [1] 0             <NA>          30            did not drive 1-2
## [6] 3-5           20-29         10-19         6-9
## Levels: 0 30 did not drive 1-2 3-5 20-29 10-19 6-9
```

```
unique(youth_risk$physically_active_7d)
```

```
## [1] 4    2    7    0    1    5    3    <NA> 6
## Levels: 0 1 2 3 4 5 6 7
```

No observamos ningún dato, a priori, que pudiera ser incorrecto. Ninguna categoría inadecuada para las categóricas y en los valores continuos si tenemos algún dato atípico respecto al peso y la altura que trataremos a continuación. Lo que sí que podemos ver es que hay algunas categorías poco representadas que recategorizaremos a clases más genericas:

- En helmet_12m vamos a juntar "always" con "most of time" y "rarely" con "sometimes"
- En school_night_hours_sleep vamos a juntar "5" con "<5"
- En race las clases que no son "White" o "Black or African American" parecen infrarepresentadas, así que las ponemos en "other"

```
youth_risk <- youth_risk %>%
  mutate(helmet_12m = recode(helmet_12m, "always"="most of time",
                             "rarely"="sometimes"))

youth_risk <- youth_risk %>%
  mutate(school_night_hours_sleep = recode(school_night_hours_sleep,
                                           "5"="5-", "<5"="5-"))

youth_risk <- youth_risk %>%
  mutate(race = recode(race,
                       "Asian"="other",
                       "Native Hawaiian or Other Pacific Islander"="other",
                       "American Indian or Alaska Native" = "other"))
```

Ahora eliminaremos del conjunto los datos con valores NA en gender ya que no nos servirán para hacer las predicciones y entrenar nuestro modelo.

```
youth_risk <- youth_risk[complete.cases(youth_risk$gender),]
```

Pasamos ahora a la imputación de datos missing con el dataframe que nos ha quedado. Para ello usaremos las siguientes funciones que rescatamos del módulo de minería de datos. Imputaremos con valores aleatorios.

```
# Imputación variables cuantitativas
ImputacionCuant<-function(vv,tipo){#tipo debe tomar los valores media, mediana o aleatorio
  if (tipo=="media"){
    vv[is.na(vv)]<-round(mean(vv,na.rm=T),4)
  } else if (tipo=="mediana"){
    vv[is.na(vv)]<-round(median(vv,na.rm=T),4)
  } else if (tipo=="aleatorio"){
    dd<-density(vv,na.rm=T,from=min(vv,na.rm = T),to=max(vv,na.rm = T))
    vv[is.na(vv)]<-round(approx(cumsum(dd$y)/sum(dd$y),
                                dd$x,runif(sum(is.na(vv))))$y,4)
  }
  vv
}

# Imputación variables cualitativas
ImputacionCuali<-function(vv,tipo){#tipo debe tomar los valores moda o aleatorio
  if (tipo=="moda"){
    vv[is.na(vv)]<-names(sort(table(vv),decreasing = T))[1]
  } else if (tipo=="aleatorio"){
    vv[is.na(vv)]<-sample(vv[!is.na(vv)],sum(is.na(vv)),replace = T)
  }
  vv
}
#Guardamos cada una

youth_risk[,as.vector(which(sapply(youth_risk, class)=="numeric"))]<-
  sapply(Filter(is.numeric,youth_risk),
         function(x) ImputacionCuant(x,"aleatorio"))

youth_risk[,as.vector(which(sapply(youth_risk, class)=="factor"))]<-
  sapply(Filter(is.factor, youth_risk),
         function(x) ImputacionCuali(x,"aleatorio"))

#Para asegurarnos de que quedan como factor
youth_risk[,as.vector(which(sapply(youth_risk, class)=="character"))]<-
  lapply(youth_risk[,as.vector(which(sapply(youth_risk, class)=="character"))],
         factor)

#Lo guardamos ya depurado

youth_risk_dep <- youth_risk
```

Veamos que ha quedado nuestro nuevo dataframe ya listo para aplicar los algoritmos.

```
summary(youth_risk_dep)
```

```
##  age          gender       grade          hispanic
##  12:  26   female:6621   10   :3165   hispanic: 3492
##  13:  18   male  :6950   11   :3203   not     :10079
##  14:1379                 12   :3576
##  15:3111                 9    :3604
##  16:3218                 other:  23
##  17:3489
##  18:2330
##                          race          height          weight
##  Black or African American:4022   Min.   :1.300   Min.   : 29.94
##  other                     :1462   1st Qu.:1.600   1st Qu.: 56.25
##  White                     :8087   Median :1.680   Median : 64.86
##                                    Mean   :1.691   Mean   : 67.91
##                                    3rd Qu.:1.778   3rd Qu.: 76.20
##                                    Max.   :2.110   Max.   :163.30
##
##        helmet_12m     text_while_driving_30d physically_active_7d
##  did not ride:4662   0            :5140       7      :3682
##  most of time: 709   did not drive:4970       0      :2213
##  never       :7129   1-2          : 995       5      :1775
##  sometimes   :1071   30           : 885       3      :1477
##                      3-5          : 529       2      :1297
##                      10-19        : 399       4      :1295
##                      (Other)      : 653       (Other):1832
##  hours_tv_per_school_day strength_training_7d school_night_hours_sleep
##  <1          :2225       0      :3991         10+: 347
##  1           :1782       7      :2268         5- :2702
##  2           :2783       3      :1620         6  :2931
##  3           :2194       5      :1449         7  :3800
##  4           :1081       2      :1409         8  :2941
##  5+          :1633       4      :1161         9  : 850
##  do not watch:1873       (Other):1673
```

Selección de variables

Empezamos fijando la semilla de aleatoriedad que vamos a usar durante toda la práctica.

```
set.seed(99)
```

Empecemos preparando la selección de variables. Definimos nuestra variable objetivo y nuestro conjunto de datos sobre el que inferir.

```
varObj <- youth_risk_dep$gender
datos <- youth_risk_dep[,c(-2)]
datos <- cbind(datos,varObj)

#Creamos una partición para hacer train test

trainIndex_bin <- createDataPartition(datos$varObj, p=0.8, list=FALSE)
data_train_bin <- datos[trainIndex_bin,]
data_test_bin <- datos[-trainIndex_bin,]
```

Hacemos la selección de variables.

```
#Hacemos selección de variables

null_bin<-glm(varObj~1, data=data_train_bin, family = binomial)#Modelo minimo
full_bin<-glm(varObj~., data=data_train_bin, family = binomial)
#Modelo maximo

#AIC
#Dirección Forward
modeloStepForwardAIC_bin<-step(null_bin,
                               scope=list(lower=null_bin, upper=full_bin),
                               direction="forward")
#Dirección backward
modeloBackAIC_bin<-step(full_bin,
                        scope=list(lower=null_bin, upper=full_bin),
                        direction="backward")
#Dirección stepwise
modeloStepAIC_bin<-step(null_bin,
                        scope=list(lower=null_bin, upper=full_bin),
                        direction="both")
#Veamos según el SBC
modeloStepForwardBIC_bin<-
  step(null_bin, scope=
         list(lower=null_bin, upper=full_bin),direction="forward",
       k=log(nrow(data_train_bin)))

modeloStepBIC_bin<-
  step(null_bin, scope=list(lower=null_bin, upper=full_bin),
       direction="both",k=log(nrow(data_train_bin)))
```

```
modeloBackBIC_bin<-
  step(full_bin, scope=list(lower=null_bin, upper=full_bin),
       direction="backward",k=log(nrow(data_train_bin)))

#Obtenemos dos modelos según AIC y BIC

modelo1 <- modeloStepAIC_bin
modelo2 <- modeloStepForwardBIC_bin
formula1 <- modeloStepForwardAIC_bin$formula
formula2 <- modeloStepForwardBIC_bin$formula
```

Comparemos el área bajo la curva roc que genera cada modelo:

```
roc1 <-roc(data_train_bin$varObj,
           predict(modelo1,data_train_bin,type = "response"))
roc2 <- roc(data_train_bin$varObj,
            predict(modelo2,data_train_bin,type = "response"))
```

Obtenemos que roc1 = 0.8797 y roc2 = 0.8774. Es ligeramente mejor el modelo1. Consideraremos entonces las variables proporcionadas por él que son «gender = height + physically_active_7d + helmet_12m + hispanic + strength_training_7d + weight + race + grade».

#Regresión Logística

Pasamos a hacer la regresión logística a partir de esta selección.

```
#Usaremos caret

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

logi<- train(gender ~ height + physically_active_7d + helmet_12m + hispanic +
               strength_training_7d + weight + race + grade,data=youth_risk_dep,
             method="glm",trControl=control)

sal<-logi$pred
salconfu<-confusionMatrix(sal$pred,sal$obs)
salconfu
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction female male
##     female   5279 1357
##     male     1342 5593
##
##                Accuracy : 0.8011
##                  95% CI : (0.7943, 0.8078)
##     No Information Rate : 0.5121
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.602
##
```

```
##  Mcnemar's Test P-Value : 0.7876
##
##             Sensitivity : 0.7973
##             Specificity : 0.8047
##          Pos Pred Value : 0.7955
##          Neg Pred Value : 0.8065
##              Prevalence : 0.4879
##          Detection Rate : 0.3890
##    Detection Prevalence : 0.4890
##       Balanced Accuracy : 0.8010
##
##        'Positive' Class : female
##
```

```
curvaroc<-roc(response=sal$obs,predictor=sal$male)
```

```
## Setting levels: control = female, case = male
```

```
## Setting direction: controls < cases
```

```
auc<-curvaroc$auc
auc
```

```
## Area under the curve: 0.8768
```

Esta será la regresión logística que tomaremos como referencia para modelos posteriores.

Algoritmos basados en árboles

Árbol de clasificación

Este modelo se construye a partir de un conjunto de datos de entrenamiento y se divide en ramas, donde cada rama representa una posible decisión que se puede tomar. Estas ramas se generan mediante funciones matemáticas que determinan los mejores cortes posibles en las variables que mejoran nuestra decisión.

Una de las mayores ventajas que presenta es su explicabilidad respecto a otros modelos de Machine Learning, un factor a tener en cuenta a la hora de elegir nuestro modelo.

Probemos con un primer árbol tomando nuestra selección de variables y estableciendo una última capa con un tamaño de 30 observaciones como hiperparámetro.

```
arbolgrid <- expand.grid(cp=c(0))
arbolcaret<- train(gender ~ height + physically_active_7d +
                     helmet_12m + hispanic +
                     strength_training_7d + weight + race + grade,
                   data=youth_risk_dep, method="rpart",
                   minbucket=30,trControl=control,tuneGrid=arbolgrid)
sal<-arbolcaret$pred
salconfu<-confusionMatrix(sal$pred,sal$obs)
curvaroc<-roc(response=sal$obs,predictor=sal$male)
```

```
## Setting levels: control = female, case = male
```

```
## Setting direction: controls < cases
```

```
auc<-curvaroc$auc
```

Obtenemos la mejor versión para un minbucket = 50 con un auc de 0.8601, pero accuracy de 0.783.

Haremos lo mismo con un árbol con todas las variables posibles.

```
arbolcaret_total<- train(gender ~ .,
                   data=youth_risk_dep, method="rpart",minbucket=30,
                   trControl=control,tuneGrid=arbolgrid)
```

```
arbolcaret_total
```

```
## CART
##
## 13571 samples
##    12 predictor
##     2 classes: 'female', 'male'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 10177, 10179, 10179, 10178
## Resampling results:
##
##   Accuracy   Kappa
##   0.7662686  0.5324274
##
## Tuning parameter 'cp' was held constant at a value of 0
```

Obtenemos que es mejor el que hemos hecho mediante selección de variables. Tunearemos ahora el parámetro del minbucket a partir de nuestro mejor modelo para ver cuál es la mejor aproximación.

```
mbucket = list()
auc_mbucket = list()

for (minbu in seq(from=30, to=105, by=10)){
  mbucket <- append(mbucket, minbu)
  cat("/n")

  arbolgrid <-  expand.grid(cp=c(0))

  arbolcaret<- train(gender ~ height + physically_active_7d + helmet_12m +
                       hispanic + strength_training_7d + weight +
                       race + grade,data=youth_risk_dep,
                     method="rpart",minbucket = minbu, trControl=control,
                     tuneGrid=arbolgrid)


  sal<-arbolcaret$pred

  curvaroc<-roc(response=sal$obs,predictor=sal$male)
  auc<-curvaroc$auc

  auc_mbucket <- append(auc_mbucket, auc)
}
result_minbucket <- Map(c, mbucket, auc_mbucket)
```

```
result_minbucket
```

```
## [[1]]
## [1] 30.0000000  0.8568334
##
## [[2]]
## [1] 40.0000000  0.8590352
##
## [[3]]
## [1] 50.0000000  0.8601009
##
## [[4]]
## [1] 60.0000000  0.8569025
##
## [[5]]
## [1] 70.0000000  0.8586047
##
## [[6]]
## [1] 80.0000000  0.8600778
##
## [[7]]
## [1] 90.0000000  0.8583794
##
## [[8]]
## [1] 100.0000000   0.8586294
```

Obtenemos la mejor versión para un minbucket = 50 con un auc de 0.8601, pero accuracy de 0.768.

Bagging

El bagging es una técnica utilizada en el árbol de decisión para mejorar la precisión del modelo. Consiste en generar múltiples árboles de decisión a partir de diferentes subconjuntos aleatorios del conjunto de datos de entrenamiento. El bagging ayuda a reducir la varianza y el sobreajuste del modelo, ya que cada árbol de decisión se entrena con un subconjunto aleatorio del conjunto de datos y, por lo tanto, tiene una visión limitada del conjunto completo.

```
rfgrid_baggin_reg_log<-expand.grid(mtry=c(8))
baggin <- train(data=youth_risk_dep,
           gender ~ height + physically_active_7d + helmet_12m + hispanic +
             strength_training_7d + weight + race + grade,
           method="rf",trControl=control,tuneGrid=rfgrid_baggin_reg_log,
           linout = FALSE,ntree=5000,sampsize=200,nodesize=100,replace=TRUE)
```

```
baggin
```

```
## Random Forest
##
## 13571 samples
##     8 predictor
##     2 classes: 'female', 'male'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 10178, 10179, 10179, 10177
## Resampling results:
##
##   Accuracy   Kappa
##   0.7918357  0.5838169
##
## Tuning parameter 'mtry' was held constant at a value of 8
```

Ha mejorado bastante accuracy a 0.79. Hagamos ahora otro con todas las variables.

```
rfgrid_baggin_total<-expand.grid(mtry=c(12))

baggin_total<- train(data=youth_risk_dep,
               gender ~ .,
               method="rf",trControl=control,tuneGrid=rfgrid_baggin_total,
               linout = FALSE,ntree=5000,sampsize=200,nodesize=100,replace=TRUE)
```

```
baggin_total
```

```
## Random Forest
##
## 13571 samples
##    12 predictor
```

```
##     2 classes: 'female', 'male'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 10178, 10178, 10178, 10179
## Resampling results:
##
##   Accuracy   Kappa
##   0.7910985  0.5823731
##
## Tuning parameter 'mtry' was held constant at a value of 12
```

Es muy similar con un accuracy de 0.791 y kappa de 0.528.

Random Forest

Al igual que el bagging, este es un modelo que busca mejorar el algoritmo de árbol clásico. Aquí además de trabajar con subconjuntos aleatorios dentro de nuestro conjunto completo, hacemos lo mismo con variables; generando árboles con subconjuntos de estas y así podemos reducir la influencia de variables muy significativas y obtener información de otras que si no pasarían desapercibidas. Cabe destacar que visto de esta forma el bagging es una variante del random forest donde tomamos la totalidad de las variables en cada generación.

Probamos un tuneo con distintos números de variables que tomamos en el subconjunto y nos quedamos con el mejor.

```
rfgrid<-expand.grid(mtry=c(8,9,10,11,12))

rf<- train(gender~.,data=youth_risk_dep, method="rf",trControl=control,
           tuneGrid=rfgrid, linout = FALSE, ntree=100,nodesize=100,replace=TRUE,
           importance=TRUE)
```

```
rf$results
```

```
##   mtry  Accuracy     Kappa  AccuracySD     KappaSD
## 1    8 0.7966256 0.5934281 0.004146252 0.008294723
## 2    9 0.7973623 0.5949130 0.003117513 0.006160726
## 3   10 0.7982463 0.5966657 0.003137697 0.006212565
## 4   11 0.7985413 0.5972257 0.002491963 0.004927581
## 5   12 0.7965518 0.5931859 0.002600102 0.005171312
```

El mejor es el de 11 variables con Accuracy de 0.7985413 y Kappa de 0.5972257. Es el mejor hasta el momento.

Gradient Boosting

Este es una versión mejorada del Random Forest, pero el Gradient Boosting entrena cada árbol de decisión para mejorar los errores del modelo anterior. Esto lo hace siguiendo el la función que minimiza el descenso de gradiente.

Aquí el parámetro más importante será shrinkage (parámetro de regularización, mide la velocidad de ajuste, a menor valor, máslento y necesita más iteraciones, pero es más fino en el ajuste).

```
gbmgrid<- expand.grid(shrinkage=c(0.1,0.05),
                      n.minobsinnode=c(100,200),
                      n.trees=c(1000,500),
                      interaction.depth=c(2))

gbm<- train(gender~.,data=youth_risk_dep,
            method="gbm",trControl=control,tuneGrid=gbmgrid,
            distribution="bernoulli", bag.fraction=1,verbose=FALSE)
```

```
gbm
```

```
## Stochastic Gradient Boosting
##
## 13571 samples
##    12 predictor
##     2 classes: 'female', 'male'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 10178, 10178, 10178, 10179
## Resampling results across tuning parameters:
##
##   shrinkage  n.minobsinnode  n.trees  Accuracy   Kappa
##   0.05       100              500     0.8028878  0.6057656
##   0.05       100             1000     0.8022247  0.6044407
##   0.05       200              500     0.8029615  0.6059166
##   0.05       200             1000     0.8029615  0.6059115
##   0.10       100              500     0.8035511  0.6070931
##   0.10       100             1000     0.8014880  0.6029560
##   0.10       200              500     0.8028877  0.6057606
##   0.10       200             1000     0.8027408  0.6054395
##
## 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 = 500, interaction.depth =
##  2, shrinkage = 0.1 and n.minobsinnode = 100.
```

Los mejores obetenidos sonn.trees = 500, interaction.depth = 2, shrinkage = 0.1 y n.minobsinnode = 100. Obteniendo accuracy de 0.8035511 y kappa de 0.6070931.

XGBoost

El XGBoosting mejora el Gradient Boosting al utilizar una estrategia de optimización de gradiente más eficiente y un regularizador de penalización para evitar el sobreajuste. El problema con este método es que nació totalmente de la práctica, por lo que no hay mucha base teórica respecto a el y también tiene bastantes más parámetros a tener en cuenta que en otros algoritmos de clasificación con árboles.

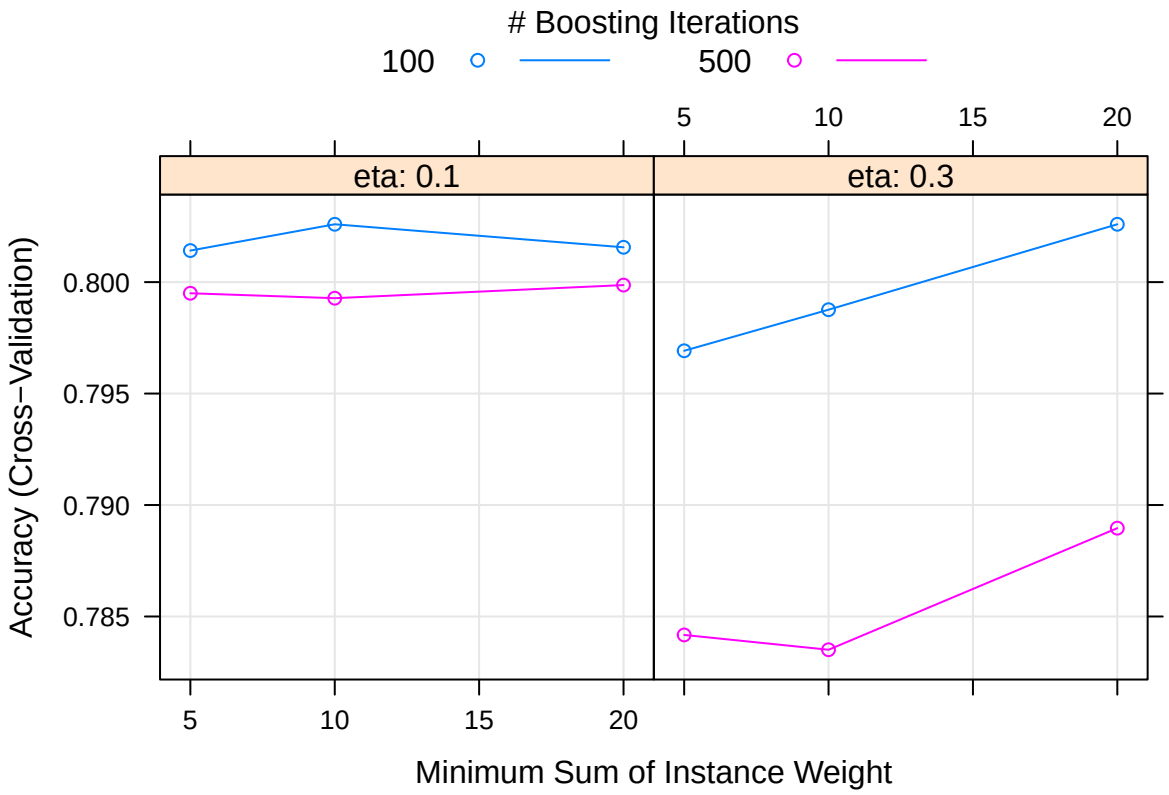
```
xgbmgrid<-expand.grid(min_child_weight=c(5,10,20),
                      eta=c(0.3,0.1), nrounds=c(100,500),
                      max_depth=6,gamma=0,colsample_bytree=1,subsample=1)
```

```
xgbm<- train(gender~.,data=youth_risk_dep,
             method="xgbTree",trControl=control,tuneGrid=xgbmgrid,verbose=FALSE)
```

```
xgbm
```

```
## eXtreme Gradient Boosting
##
## 13571 samples
##    12 predictor
##     2 classes: 'female', 'male'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 10178, 10178, 10178, 10179
## Resampling results across tuning parameters:
##
##   eta  min_child_weight  nrounds  Accuracy   Kappa
##   0.1   5                100      0.8014146  0.6028394
##   0.1   5                500      0.7994988  0.5988748
##   0.1  10                100      0.8025936  0.6051938
##   0.1  10                500      0.7992777  0.5985032
##   0.1  20                100      0.8015621  0.6031572
##   0.1  20                500      0.7998672  0.5996716
##   0.3   5                100      0.7969193  0.5937741
##   0.3   5                500      0.7841718  0.5681772
##   0.3  10                100      0.7987617  0.5974911
##   0.3  10                500      0.7835084  0.5668197
##   0.3  20                100      0.8025936  0.6051451
##   0.3  20                500      0.7889614  0.5778035
##
## 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
##  = 0.3, gamma = 0, colsample_bytree = 1, min_child_weight = 20 and subsample
##  = 1.
```

```
plot(xgbm)
```



Los mejores modelos obtenidos del xgboost son nrounds = 100, max_depth = 6, eta = 0.3, gamma = 0, colsample_bytree= 1, min_child_weight = 20 and subsample = 1. Obtenemos accuracy de 0.8025936 y kappa de 0.6051451. En el plot se aprecia efectivamente como mejoran los parámetros y se llega al punto óptimo.

Redes Neuronales

Para una red neuronal, una vez tenemos depurados los datos necesitamos ajustar los valores de las variables continuas. Tenemos que hacer una transformación que puede ser un escalado o una normalización. Optaremos por la normalización en este caso.

```
var_cont <- c("height", "weight")


means <-apply(youth_risk_dep[,var_cont],2,mean,na.rm=TRUE)
sds<-apply(youth_risk_dep[,var_cont],2,sd,na.rm=TRUE)

df_auxiliar <-scale(youth_risk_dep[,var_cont], center = means, scale = sds)
youth_risk_norm <- data.frame(cbind(df_auxiliar,
              youth_risk_dep[,-which(colnames(youth_risk_dep) %in% var_cont)]))
```

Una red neuronal está inspirada en la estructura y el funcionamiento del cerebro humano. Se compone de múltiples capas de nodos interconectados (neuronas) que procesan y transforman la información de entrada para producir una salida. Aquí los principales parámetros a tunear son la tasa de aprendizaje de las neuronas en cada iteración y el número de nodos que colocamos.

Probemos con una red con todas las variables, con tasa de aprendizaje 0.1 y 3 neuronas en la capa oculta.

```
control<-trainControl(method = "LGOCV",p=0.8,number=1,savePredictions = "all")
nnetgrid <-expand.grid(size=c(3),decay=c(0.1))

rednnet<- train(gender~.,
                data=youth_risk_norm, method="nnet",
                linout = FALSE, trControl=control, tuneGrid=nnetgrid)
```

```
rednnet
```

```
## Neural Network
##
## 13571 samples
##    12 predictor
##     2 classes: 'female', 'male'
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (1 reps, 80%)
## Summary of sample sizes: 10857
## Resampling results:
##
##   Accuracy   Kappa
##   0.8043478  0.6086115
##
## Tuning parameter 'size' was held constant at a value of 3
## Tuning
##  parameter 'decay' was held constant at a value of 0.1
```

Obtenemos Accuracy de 0.8043478 y Kappa de 0.6086115.

Juguemos un poco con la arquitectura (numero de capas ocultas) para ver si podemos mejorar esta predicción

Estamos ante un dataframe con muchas variables categóricas por lo que aparentemente los datos guarden ciertas relaciones complejas latentes. Probaremos a aumentar el número de nodos.

```
nnetgrid_nodos <-expand.grid(size=c(3:20),decay=c(0.1,0.01))

rednnet_nodos<- train(gender~.,
                data=youth_risk_norm, method="nnet",
                linout = FALSE, trControl=control, tuneGrid=nnetgrid_nodos)
```

```
rednnet_nodos
```

```
## Neural Network
##
## 13571 samples
##    12 predictor
##     2 classes: 'female', 'male'
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (1 reps, 80%)
## Summary of sample sizes: 10857
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##    3    0.01   0.8235077  0.6470408
##    3    0.10   0.8227708  0.6457067
##    4    0.01   0.8124539  0.6246388
##    4    0.10   0.8135593  0.6271521
##    5    0.01   0.8095063  0.6187802
##    5    0.10   0.8091378  0.6182277
##    6    0.01   0.8131909  0.6265022
##    6    0.10   0.8106116  0.6212708
##    7    0.01   0.8072955  0.6144666
##    7    0.10   0.8109801  0.6218518
##    8    0.01   0.8061901  0.6119978
##    8    0.10   0.8139278  0.6276420
##    9    0.01   0.8036109  0.6072359
##    9    0.10   0.8142962  0.6284260
##   10    0.01   0.7881356  0.5761952
##   10    0.10   0.8036109  0.6071232
##   11    0.01   0.7958732  0.5916367
##   11    0.10   0.8014001  0.6027149
##   12    0.01   0.7925571  0.5850399
##   12    0.10   0.7995578  0.5989360
##   13    0.01   0.7840825  0.5680799
##   13    0.10   0.8076640  0.6151969
##   14    0.01   0.7877671  0.5755723
##   14    0.10   0.8050847  0.6100577
##   15    0.01   0.7833456  0.5666058
##   15    0.10   0.8010317  0.6019421
##   16    0.01   0.7855564  0.5710282
##   16    0.10   0.7962417  0.5923227
##   17    0.01   0.7881356  0.5759368
##   17    0.10   0.8065586  0.6128813
```

```
##   18    0.01   0.7826087  0.5648821
##   18    0.10   0.7958732  0.5915928
##   19    0.01   0.7759764  0.5515751
##   19    0.10   0.7892410  0.5783988
##   20    0.01         NaN        NaN
##   20    0.10         NaN        NaN
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 3 and decay = 0.01.
```

Parece que encontramos el mejor resultado para size = 3 y decay = 0.01. obteniendo accuracy y kappa de 0.8227708 y 0.6457067 respectivamente. De momento es nuestra mejor opción.

Probaremos ahora una red con la selección de variables hecha anteriormente.

```
rednnet_nodos_seleccion<- train(gender ~ height + physically_active_7d + helmet_12m + hispanic +
                                  strength_training_7d + weight + race + grade,
                      data=youth_risk_norm, method="nnet",
                      linout = FALSE, trControl=control, tuneGrid=nnetgrid_nodos)
```

```
rednnet_nodos_seleccion
```

```
## Neural Network
##
## 13571 samples
##     8 predictor
##     2 classes: 'female', 'male'
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (1 reps, 80%)
## Summary of sample sizes: 10857
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##    3    0.01   0.8065586  0.6129785
##    3    0.10   0.8095063  0.6189716
##    4    0.01   0.8076640  0.6152797
##    4    0.10   0.8025055  0.6050889
##    5    0.01   0.8043478  0.6086115
##    5    0.10   0.8054532  0.6109970
##    6    0.01   0.7947679  0.5895946
##    6    0.10   0.8069270  0.6137780
##    7    0.01   0.8002948  0.6004965
##    7    0.10   0.8006632  0.6013693
##    8    0.01   0.7932940  0.5866918
##    8    0.10   0.8043478  0.6086395
##    9    0.01   0.8006632  0.6014122
##    9    0.10   0.8072955  0.6145910
##   10    0.01   0.7951363  0.5903387
##   10    0.10   0.8002948  0.6005108
##   11    0.01   0.7969786  0.5941173
##   11    0.10   0.7958732  0.5919001
##   12    0.01   0.7962417  0.5924980
##   12    0.10   0.8028740  0.6059175
```

```
##   13    0.01   0.7877671  0.5753440
##   13    0.10   0.7955048  0.5910389
##   14    0.01   0.7877671  0.5756331
##   14    0.10   0.7958732  0.5916221
##   15    0.01   0.7907148  0.5815421
##   15    0.10   0.7921887  0.5846081
##   16    0.01   0.7789241  0.5579195
##   16    0.10   0.7977155  0.5954021
##   17    0.01   0.7785556  0.5572224
##   17    0.10   0.7910833  0.5819418
##   18    0.01   0.7870302  0.5740833
##   18    0.10   0.7862933  0.5725636
##   19    0.01   0.7833456  0.5667456
##   19    0.10   0.7914517  0.5829408
##   20    0.01   0.7748710  0.5497905
##   20    0.10   0.7870302  0.5743579
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 3 and decay = 0.1.
```

Obtenemos el caso size = 3 y decay = 0.1 con accuracy y kappa de 0.8095063 y 0.6189716 respectivamente. Es mejor el modelo con todas las variables. Esa será nuestra red neuronal a considerar.

Máquinas de Vector Soporte

Las Máquinas de Vector Soporte (SVM) Se basa en encontrar un hiperplano que maximice la distancia entre las diferentes clases de datos en un espacio de alta dimensión. Para esto se usan técnicas de operación matemática basadas en el algoritmo del Simplex.

Una de las claves en las SVM es el uso de distintas funciones para comprender como actúan los vectores del espacio sobre el que tratamos de hacer nuestra clasificación. A veces nos funcionará usar una función lineal, pero otras veces será mejor usar una polinómica que se adapte más a relaciones no lineales entre los datos. Con estructuras más complejas se pueden llegar a buscar kernel más complicados que recurran a relaciones que solo se aprecian en dimensiones superiores para hacer una mejor clasificación de nuestro conjunto.

Para esta práctica probaremos con tres kernels distintos para ver cuál clasifica mejor.

```
control<-trainControl(method = "cv",number=5,savePredictions = "all",
                      classProbs = TRUE)

SVMgrid<-expand.grid(C=c(0.01,0.1))

#Lineal

SVM_lin<- train(data=youth_risk_dep,gender ~ height + physically_active_7d +
              helmet_12m + hispanic +
              strength_training_7d + weight + race + grade,
            method="svmLinear",trControl=control,
            tuneGrid=SVMgrid,verbose=FALSE)
```

```
SVM_lin$results
```

```
##      C  Accuracy     Kappa  AccuracySD    KappaSD
## 1 0.01 0.8029623 0.6056955 0.008927001 0.01779755
## 2 0.10 0.8021519 0.6040514 0.008763654 0.01750312
```

Gana para con una tasa de aprendizaje de 0.01 con accuracy y kappa de 0.8029623 y 0.6056955. Vamos con la polinómica de grado 2:

```
SVMgrid_pol<-expand.grid(C=c(0.01,0.1),
                     degree=c(2),scale=c(0.5))

SVM_pol<- train(data=youth_risk_dep,gender ~ height + physically_active_7d +
                  helmet_12m + hispanic +
                  strength_training_7d + weight + race + grade,
                method="svmPoly",trControl=control,
                tuneGrid=SVMgrid_pol,verbose=FALSE)
```

```
SVM_pol$results
```

```
##      C degree scale  Accuracy     Kappa  AccuracySD    KappaSD
## 1 0.01      2   0.5 0.8078105 0.6157015 0.006710636 0.01336825
## 2 0.10      2   0.5 0.8036109 0.6076725 0.005575809 0.01110432
```

El mejor esultados es para una tasa de 0.01 con accuracy y kappa de 0.8078105 0.6157015. Por último vamos con la no lineal, hemos elegido hacerla con una radial.

```
SVMgrid_rad<-expand.grid(C=c(0.01,0.1),
                         sigma=c(0.5,1,2))

SVM_rad<- train(data=youth_risk_dep,gender ~ height + physically_active_7d +
                  helmet_12m + hispanic +
                  strength_training_7d + weight + race + grade,
                method="svmRadial",trControl=control,
                tuneGrid=SVMgrid_rad,verbose=FALSE)
```

```
SVM_rad$results
```

```
##      C sigma  Accuracy     Kappa  AccuracySD     KappaSD
## 1 0.01   0.5 0.6716536 0.3514503 0.007242066 0.014157950
## 4 0.10   0.5 0.7870456 0.5733178 0.010288937 0.020579617
## 2 0.01   1.0 0.6654642 0.3393417 0.010199236 0.020023588
## 5 0.10   1.0 0.7785713 0.5561552 0.008979823 0.018050018
## 3 0.01   2.0 0.6555899 0.3200760 0.007909952 0.015502014
## 6 0.10   2.0 0.7065067 0.4180090 0.003431266 0.006834888
```

Esto empeora bastante lo anterior. Esto se debe a que probablemente los datos guarden relaciones lineales más sencillas y no requerimos de mayor artificio.

Métodos de Ensamblado

Bagging (ensamblado)

Comparte nombre con el bagging de árboles porque en esencia es lo mismo. Es una filosofía similar pero en vez de generar árboles hacemos las submuestras ejecutadas sobre distintos modelos que luego, siguiendo un criterio, juntamos. En este caso optaremos por escoger unir nuestras distintas predicciones con la media.

Vamos a hacer este bagging con la regresión logística como clasificador y lo comparamos con una regresión logística al uso.

```
sample <- sample.int(n = nrow(youth_risk_dep),
                     size = floor(0.75*nrow(youth_risk_dep)), replace = F)

train <- youth_risk_dep[sample, ]
test  <- youth_risk_dep[-sample, ]

#Generamos las 4 muestras
s1 <- sample.int(n = nrow(train),
                 size = floor(0.45*nrow(train)), replace = T)

train_s1 <- train[s1,]

s2 <- sample.int(n = nrow(train),
                 size = floor(0.45*nrow(train)), replace = T)
train_s2 <- train[s2,]

s3 <- sample.int(n = nrow(train),
                 size = floor(0.45*nrow(train)), replace = T)

train_s3 <- train[s3,]

s4 <- sample.int(n = nrow(train),
                 size = floor(0.45*nrow(train)), replace = T)

train_s4 <- train[s4,]

#Guardamos los 4 modelos

logi<-glm(data=train,
          gender ~ height + physically_active_7d +
            helmet_12m + hispanic +
            strength_training_7d + weight + race + grade,
          family=binomial(link = "logit"))


logi_s1<-glm(data=train_s1,
             gender ~ height + physically_active_7d +
               helmet_12m + hispanic +
               strength_training_7d + weight + race + grade,
             family=binomial(link = "logit"))


logi_s2<-glm(data=train_s2,
```

```
              gender ~ height + physically_active_7d +
                helmet_12m + hispanic +
                strength_training_7d + weight + race + grade,
              family=binomial(link = "logit"))


logi_s3<-glm(data=train_s3,
              gender ~ height + physically_active_7d +
                helmet_12m + hispanic +
                strength_training_7d + weight + race + grade,
              family=binomial(link = "logit"))


logi_s4<-glm(data=train_s4,
              gender ~ height + physically_active_7d +
                helmet_12m + hispanic +
                strength_training_7d + weight + race + grade,
              family=binomial(link = "logit"))

#Guardamos las predicciones

preditestglm<-predict(logi,newdata=test,type = "response")

preditestglm_1<-predict(logi_s1,newdata=test,type = "response")
preditestglm_2<-predict(logi_s2,newdata=test,type = "response")
preditestglm_3<-predict(logi_s3,newdata=test,type = "response")
preditestglm_4<-predict(logi_s4,newdata=test,type = "response")

preditestglm<-as.data.frame(preditestglm)
preditestglm_1<-as.data.frame(preditestglm_1)
preditestglm_2<-as.data.frame(preditestglm_2)
preditestglm_3<-as.data.frame(preditestglm_3)
preditestglm_4<-as.data.frame(preditestglm_4)

#Las agregamos por media

predi_test_aver <- (preditestglm_1 + preditestglm_2 + preditestglm_3 + preditestglm_4)/4

estima_bag <- ifelse(predi_test_aver>0.50,"Male","Female")
estima_glm <-ifelse(preditestglm>0.50,"Male","Female")

result_bagging <- as.data.frame(cbind(gender = test$gender,estima_bag,estima_glm))

confusion_baggin <- table(result_bagging$gender,result_bagging$preditestglm_1)

confusion_baggin_glm <- table(result_bagging$gender,result_bagging$preditestglm)

accuracy_baggin <- sum(diag(confusion_baggin)) / sum(confusion_baggin)

accuracy_baggin_glm <- sum(diag(confusion_baggin_glm)) / sum(confusion_baggin_glm)
```

Obtenemos los siguientes resultados comparandolos con la regresión logística.

```
confusion_baggin
```

```
##
##     Female Male
##   1   1328  317
##   2    353 1395
```

```
confusion_baggin_glm
```

```
##
##     Female Male
##   1   1334  311
##   2    357 1391
```

```
accuracy_baggin
```

```
## [1] 0.8025346
```

```
accuracy_baggin_glm
```

```
## [1] 0.8031241
```

El cambio es ínfimos. Con el modelo resultante del bagging quizá solo estemos reduciendo un poco el sobreajuste pero el cambio es muy poco significativo.

Stacking

El método de stacking es muy similar al bagging pero usando distintos algoritmos en la combinación y combinamos con otro algoritmo en este caso vamos a combinar un árbol y una regresión logística y hacemos el stacking en sí con otra regresión como método de unión entre ambos modelos.

```
inTrain <- createDataPartition(y = youth_risk_dep$gender, p = .75, list = FALSE)
training <- youth_risk_dep[ inTrain,]
testing <- youth_risk_dep[-inTrain,]

my_control <- trainControl(
  method="glm",
  number=25,
  savePredictions="final",
  classProbs=TRUE,
  index=createResample(training$gender, 25),
  summaryFunction=twoClassSummary
)

model_list <- caretList(
  gender~., data=youth_risk_dep,
  trControl=my_control,
  methodList=c("glm", "rpart")
)
```

```
glm_ensemble <- caretStack(
  model_list,
  method="glm",
  metric="ROC",
  trControl=trainControl(
    method="boot",
    number=10,
    savePredictions="final",
    classProbs=TRUE,
    summaryFunction=twoClassSummary
  )
)

ensemble <- predict(glm_ensemble, newdata=testing, type="raw")

comparativa <- as.data.frame(cbind(testing$gender, ensemble))
confusion_stacking <- table(comparativa$V1,comparativa$ensemble)

accuracy_stacking <- sum(diag(confusion_stacking)) / sum(confusion_stacking)
```

```
accuracy_stacking
```

```
## [1] 0.8001179
```

El stacking nos deja con accuracy 0.8001179.

Modelo ganador

De entre todos los algoritmos probados hemos obtenido que el que mayor accuracy ha obtenido ha sido la red neuronal entrenada con todas las variables y con parámetros de 3 nodos y decay = 0.01 como tasa de aprendizaje, obteniendo accuracy y kappa de 0.8227708 y 0.6457067.

```
rednnet_nodos
```

```
## Neural Network
##
## 13571 samples
##    12 predictor
##     2 classes: 'female', 'male'
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (1 reps, 80%)
## Summary of sample sizes: 10857
## Resampling results across tuning parameters:
##
##   size  decay  Accuracy   Kappa
##    3    0.01   0.8235077  0.6470408
##    3    0.10   0.8227708  0.6457067
##    4    0.01   0.8124539  0.6246388
##    4    0.10   0.8135593  0.6271521
##    5    0.01   0.8095063  0.6187802
##    5    0.10   0.8091378  0.6182277
##    6    0.01   0.8131909  0.6265022
##    6    0.10   0.8106116  0.6212708
##    7    0.01   0.8072955  0.6144666
##    7    0.10   0.8109801  0.6218518
##    8    0.01   0.8061901  0.6119978
##    8    0.10   0.8139278  0.6276420
##    9    0.01   0.8036109  0.6072359
##    9    0.10   0.8142962  0.6284260
##   10    0.01   0.7881356  0.5761952
##   10    0.10   0.8036109  0.6071232
##   11    0.01   0.7958732  0.5916367
##   11    0.10   0.8014001  0.6027149
##   12    0.01   0.7925571  0.5850399
##   12    0.10   0.7995578  0.5989360
##   13    0.01   0.7840825  0.5680799
##   13    0.10   0.8076640  0.6151969
##   14    0.01   0.7877671  0.5755723
##   14    0.10   0.8050847  0.6100577
##   15    0.01   0.7833456  0.5666058
##   15    0.10   0.8010317  0.6019421
##   16    0.01   0.7855564  0.5710282
##   16    0.10   0.7962417  0.5923227
##   17    0.01   0.7881356  0.5759368
##   17    0.10   0.8065586  0.6128813
##   18    0.01   0.7826087  0.5648821
##   18    0.10   0.7958732  0.5915928
##   19    0.01   0.7759764  0.5515751
##   19    0.10   0.7892410  0.5783988
```

```
##   20    0.01         NaN        NaN
##   20    0.10         NaN        NaN
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 3 and decay = 0.01.
```

El accuracy aumenta en 0.02 respecto al resto de modelos que se mnovían generalmente en torno 0.8. Lo que puede ser muy significativo es el kappa de 0.64, bastante más elevados que en el resto de modelos y que puede ser una buena señal de calidad.

Aunque este modelo es el que mejor resultados ha obtenido, hemos de tener en cuenta que las redes neuronales son un algoritmo de caja negra que presenta problemas respecto a la explicabilidad del modelo. Teniendo esto en cuenta y viendo que nuestros datos presentan algunas variables que pueden ser conflictivas (genero, raza...) podríamos, dependiendo del contexto en el que estemos trabajando, optar por un algoritmo que también sea competitivo pero que nos permita explicar mejor por qué hacemos la discriminación. En este caso lo mejor sería optar por uno de los árboles generados.