Machine Learning con H2O y R

Joaquín Amat Rodrigo j.amatrodrigo@gmail.com

Julio, 2018

Tabla de contenidos

Versión PDF: Github

Introducción

En este documento se pretende mostrar cómo crear modelos de *machine learning* combinando **H2O** y el lenguaje de programación **R**.

Aunque son muchos los pasos que preceden al entrenamiento de un modelo (exploración de los datos, transformaciones, selección de predictores, etc.), para no añadir una capa extra de complejidad, se va a asumir que los datos se encuentran prácticamente preparados para que puedan ser aceptados por los algoritmos de aprendizaje. Se puede encontrar una descripción más detallada de las etapas que forman parte el ciclo de vida del modelado en el documento Machine Learning con R y caret.

¿Qué es H2O?

H2O es un producto creado por la compañía H2O.ai con el objetivo de combinar los principales algoritmos de *machine learning* y aprendizaje estadístico con el *Big Data*. Gracias a su forma de comprimir y almacenar los datos, H2O es capaz de trabajar con millones de registros en un único ordenador (emplea todos sus cores) o en un cluster de muchos ordenadores. Internamente, H2O está escrito en Java y sigue el paradigma *Key/Value* para almacenar los datos y *Map/Reduce* para sus algoritmos. Gracias a sus API, es posible acceder a todas sus funciones desde **R**, **Python** o **Scala**, así como por una interfaz web llamada **Flow**.

Aunque la principal ventaja de H2O frente a otras herramientas es su escalabilidad, sus algoritmos son igualmente útiles cuando se trabaja con un volumen de datos reducido.

Comunicación entre H2O y R

El manejo de H2O puede hacerse íntegramente desde R: iniciar el cluster, carga de datos, entrenamiento de modelos, predicción de nuevas observaciones, etc. Es importante tener en cuenta que, aunque los comandos se ejecuten desde R, los datos se encuentran en el cluster de H2O, no en memoria. Solo cuando los datos se cargan en memoria, se les pueden aplicar funciones propias de R.

Las funciones `as.data.frame()` y `as.h2o()` permiten transferir los datos de la sesión de R al cluster H2O y viceversa. Hay que tener especial precaución cuando el movimiento de datos se hace desde H2O a R, ya que esto implica cargar en RAM todos los datos y, si son muchos, pueden

ocupar toda la memoria. Para evitar este tipo de problemas, conviene realizar todos los pasos posibles (filtrado, agregaciones, cálculo de nuevas columnas...) con las funciones de H2O antes de transferir los datos.

Datos

Para los ejemplos de este documento se emplea el set de datos *Adult Data Set* disponible en UCI Machine Learning Repository. Este set de datos contiene información sobre 48842 ciudadanos de diferentes países. Para cada ciudadano, se dispone de 15 variables:

- `age`: edad.
- `workclass`: tipo de empleo.
- `fnlwgt`: importancia relativa de cada observación.
- `education`: nivel de estudios.
- `education-num`: años de estudios.
- `marital-status`: estado civil.
- `occupation`: sector de trabajo
- `relationship`: estado familiar.
- `race`: raza.
- `sex`: género.
- `capital-gain`: ganancia de capital.
- `capital-loss`: perdida de capital.
- `hours-per-week`: horas de trabajo semanal.
- `native-countr`: país de origen.
- `salario`: si el salario supera o no 50000 dólares anuales.

El objetivo del problema es crear un modelo capaz de predecir correctamente si un ciudadano tiene un salario superior o inferior a 50000 dólares anuales. Se trata de un problema de clasificación binaria supervisado.

Los datos se han modificado para eliminar observaciones con valores ausentes y para recodificar los niveles de algunas variables. El archivo resultante (*adult_custom.csv*) puede descargarse directamente del repositorio Github.

```r
# EL SIGUIENTE CÓDIGO MUESTRA COMO SE HAN PREPROCESADO LOS DATOS ORIGINALES.
# NO ES NECESARIO EJECUTARLO SI SE HA DESCARGADO EL ARCHIVO adult_custom.csv.
library(tidyverse)

# Datos de train.
datos_train <- read_csv(file = "adult_train.csv", col_names = FALSE)
colnames(datos_train) <- c("age", "workclass", "final_weight", "education",
                           "education_number", "marital_status", "occupation",
                           "relationship", "race", "sex", "capital_gain",
                           "capital_loss", "hours_per_week", "native_country",
                           "salario")
# Datos de test.
datos_test <- read_csv(file = "adult_test.csv", col_names = FALSE, skip = 1)
colnames(datos_test) <- c("age", "workclass", "final_weight", "education",
                           "education_number", "marital_status", "occupation",
                           "relationship", "race", "sex", "capital_gain",
                           "capital_loss", "hours_per_week", "native_country",
                           "salario")

# Se combinan los dos conjuntos de datos.
datos <- bind_rows(datos_train, datos_test)

# Se eliminan registros con missing values, que en este set de datos están
# codificados como ?.
remplazar <- function(x){
  x[x == "?"] <- NA
  return(x)
}
datos <- datos %>% map_df(.f = remplazar)
datos <- drop_na(datos)

# Recodificación de niveles.
datos <- datos %>% mutate(workclass = recode(datos$workclass,
                                             "Without_pay" = "desempleado",
                                             "Self_emp_inc" = "autonomo",
                                             "Self_emp_not_inc" = "autonomo",
                                             "Federal_gov" = "funcionario",
                                             "Local_gov" = "funcionario",
                                             "State_gov" = "funcionario"))

datos <- datos %>% mutate(marital_status = recode(datos$marital_status,
                                             "Married-AF-spouse"    = "casado",
                                             "Married-civ-spouse"   = "casado",
                                             "Married-spouse-absent"= "casado",
                                             "Widowed"              = "separado",
                                             "Never-married"        = "soltero",
                                             "Separated"            = "separado",
                                             "Divorced"             = "separado")
                          )
```

```
norte_america <- c("Canada", "Cuba", "Dominican-Republic", "El-Salvador",
                   "Guatemala", "Haiti", "Honduras", "Jamaica", "Mexico",
                   "Nicaragua", "Outlying-US(Guam-USVI-etc)", "Puerto-Rico",
                   "Trinadad&Tobago", "United-States")
europa        <- c("England", "France", "Germany", "Greece", "Holand-Netherlands",
                   "Hungary", "Ireland", "Italy", "Poland", "Portugal", "Scotland",
                   "Yugoslavia")
asia          <- c("Cambodia", "China", "Hong", "India", "Iran", "Japan", "Laos",
                    "Philippines", "Taiwan", "Thailand", "Vietnam")
sud_america   <- c("Columbia", "Ecuador", "Peru")
otros         <- c("South")

datos <- datos %>% mutate(native_country = case_when(
                          native_country %in% norte_america ~ "Norte America",
                          native_country %in% asia          ~ "Asia",
                          native_country %in% sud_america   ~ "Sud America",
                          native_country %in% europa        ~ "Europa",
                          native_country %in% otros         ~ "Otros",
                          TRUE ~ as.character(native_country)
                ))

primaria     <- c("Preschool", "1st-4th", "5th-6th", "7th-8th", "9th", "10th",
                  "11th","12th")
bachillerato <- c("Some-college", "Assoc-acdm", "Assoc-voc")
master       <- c("Masters", "Prof-school")

datos <- datos %>% mutate(education = case_when(
                          education %in% primaria      ~ "primaria",
                          education == "HS-grad"       ~ "secundaria",
                          education %in% bachillerato  ~ "bachillerato",
                          education == "Bachelors"     ~ "universidad",
                          education %in% master        ~ "master",
                          education == "Doctorate"     ~ "doctorado",
                          TRUE ~ as.character(education)
                ))

datos <- datos %>% mutate(salario = case_when(
                          salario == "<=50K."      ~ "<=50K",
                          salario == ">50K."       ~ ">50K",
                          TRUE ~ as.character(salario)
                ))

# Se agrupan las observaciones cuya variable occupation == "Armed-Forces" con
# "Other-service" ya que solo hay 14 observaciones (0.03% del total).
datos <- datos %>% mutate(occupation = case_when(
                          occupation == "Armed-Forces" ~ "Other-service",
                          TRUE ~ as.character(occupation)
                         ))
write_csv(x = datos, path = "adult_custom.csv", col_names = TRUE)
```

Iniciación de H2O

Una vez que H2O ha sido instalado (`install.packages("h2o")`), hay que iniciarlo, bien en todo el cluster o en un solo ordenador. Para este ejemplo, se emplea un único ordenador del que se utilizan todos sus cores en paralelo.

```
library(h2o)
# Creación de un cluster local con todos los cores disponibles.
h2o.init(ip = "localhost",
         # -1 indica que se empleen todos los cores disponibles.
         nthreads = -1,
         # Máxima memoria disponible para el cluster.
         max_mem_size = "4g")
```

```
## R is connected to the H2O cluster:
##     H2O cluster uptime:         1 hours 10 minutes
##     H2O cluster timezone:       Europe/Madrid
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.18.0.8
##     H2O cluster version age:    3 months and 2 days
##     H2O cluster name:           H2O_started_from_R_ximo_bxz811
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.19 GB
##     H2O cluster total cores:    4
##     H2O cluster allowed cores:  4
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         XGBoost, Algos, AutoML, Core V3, Core V4
##     R Version:                  R version 3.4.4 (2018-03-15)
```

```
# Se eliminan los datos del cluster por si ya había sido iniciado.
h2o.removeAll()
```

Tras iniciar el cluster (local), se muestran por pantalla sus características, entre las que están: el número de cores activados (4), la memoria total del cluster (3.56GB), el número de nodos (1 porque se está empleando un único ordenador) y el puerto con el que conectarse a la interfaz web de H2O (http://localhost:54321/flow/index.html).

Si se desea lanzar H2O en un cluster *Hadoop* ya establecido, solo es necesario especificar la dirección IP y puerto de acceso en h2O.init().

```
# Ejemplo de conexión a un cluster remoto
h2o.init(ip = "123.45.67.89", port = 54321)
```

Carga de datos

La carga de datos puede hacerse directamente al cluster H2O, o bien cargándolos primero en memoria en la sesión de R y después transfiriéndolos. La segunda opción no es aconsejable si el volumen de datos es muy grande.

```
# Carga de datos en el cluster H2O desde url.
url_path <- paste0("https://github.com/JoaquinAmatRodrigo/Estadistica-con-R/raw/",
                   "master/datos/adult_custom.csv")

datos_h2o <- h2o.importFile(path = url_path,
                            header = TRUE,
                            sep = ",",
                            destination_frame = "datos_h2o")

# Carga de datos en el cluster H2O desde local.
datos_h2o <- h2o.importFile(path = "/home/ximo/Desktop/adult_custom.csv",
                            header = TRUE,
                            sep = ",",
                            destination_frame = "datos_h2o")

# Carga de datos en R y transferencia a H2O.
url_path <- paste0("https://github.com/JoaquinAmatRodrigo/Estadistica-con-R/raw/",
                   "master/datos/adult_custom.csv")
datos_R   <- read.csv(file = url_path, header = TRUE)
datos_h2o <- as.h2o(x = datos_R, destination_frame = "datos_h2o")
```

Exploración de los datos

Si bien el set de datos empleado en este ejemplo es suficientemente pequeño para cargarlo todo en memoria y emplear directamente las magníficas posibilidades de análisis exploratorio que ofrece R, para representar mejor la forma de proceder con grandes volúmenes de datos, se emplean funciones propias de H2O.

```
# Dimensiones del set de datos
h2o.dim(datos_h2o)

## [1] 45222    15

# Nombre de las columnas
h2o.colnames(datos_h2o)
```

```
##  [1] "age"              "workclass"        "final_weight"
##  [4] "education"        "education_number" "marital_status"
##  [7] "occupation"       "relationship"     "race"
## [10] "sex"              "capital_gain"     "capital_loss"
## [13] "hours_per_week"   "native_country"   "salario"
```

La función `h2o.describe()` es muy útil para obtener un análisis rápido que muestre el tipo de datos, la cantidad de valores ausentes, el valor mínimo, máximo, media, desviación típica y el número de categorías (*Cardinality*) de cada una de las variables. H2O empla el nombre `enum` para los datos de tipo `factor` o `character`.

```
h2o.describe(datos_h2o)
```

```
##              Label Type Missing Zeros PosInf NegInf   Min     Max
## 1              age  int       0     0      0      0    17      90
## 2        workclass enum       0  1406      0      0     0       6
## 3     final_weight  int       0     0      0      0 13492 1490400
## 4        education enum       0 13365      0      0     0       5
## 5  education_number  int       0     0      0      0     1      16
## 6    marital_status enum       0 21639      0      0     0       2
## 7       occupation enum       0  5540      0      0     0      12
## 8     relationship enum       0 18666      0      0     0       5
## 9             race enum       0   435      0      0     0       4
## 10             sex enum       0 14695      0      0     0       1
## 11     capital_gain  int       0 41432      0      0     0   99999
## 12     capital_loss  int       0 43082      0      0     0    4356
## 13   hours_per_week  int       0     0      0      0     1      99
## 14   native_country enum       0   930      0      0     0       4
## 15          salario enum       0 34014      0      0     0       1
##            Mean        Sigma Cardinality
## 1  3.854794e+01 1.321787e+01          NA
## 2            NA           NA           7
## 3  1.897347e+05 1.056392e+05          NA
## 4            NA           NA           6
## 5  1.011846e+01 2.552881e+00          NA
## 6            NA           NA           3
## 7            NA           NA          13
## 8            NA           NA           6
## 9            NA           NA           5
## 10 6.750475e-01 4.683623e-01           2
## 11 1.101430e+03 7.506430e+03          NA
## 12 8.859542e+01 4.049561e+02          NA
## 13 4.093802e+01 1.200751e+01          NA
## 14           NA           NA           5
## 15 2.478440e-01 4.317655e-01           2
```

Para conocer el índice o nombre de las columnas que son de un determinado tipo, por ejemplo, numérico, se emplea la función `h2o.columns_by_type()`.

```
# Índices
indices <- h2o.columns_by_type(object = datos_h2o, coltype = "numeric")
indices

## [1]  1  3  5 11 12 13

# Nombres
h2o.colnames(x = datos_h2o)[indices]
```

```
## [1] "age"              "final_weight"     "education_number"
## [4] "capital_gain"     "capital_loss"     "hours_per_week"
```

Con la función `h2o.cor()` se calcula la correlación entre dos o más columnas numéricas.

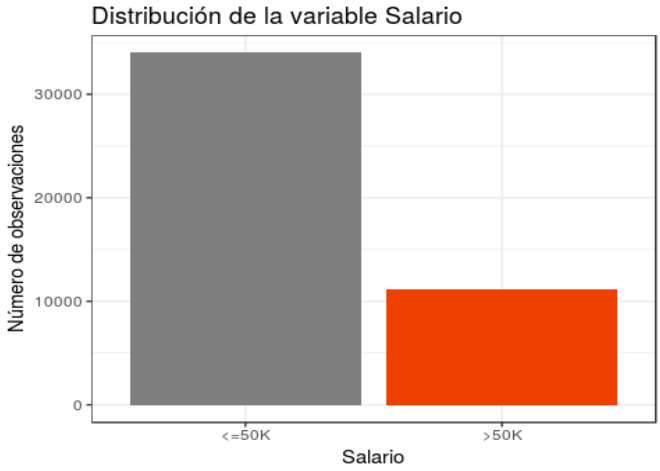
```
indices <- h2o.columns_by_type(object = datos_h2o, coltype = "numeric")
h2o.cor(x = datos_h2o[,indices], y = NULL, na.rm = TRUE)
```

```
##          age0 final_weight0 education_number0 capital_gain0 capital_loss0
## 1  1.00000000  -0.075791969        0.03762295   0.079683241   0.059350578
## 2 -0.07579197   1.000000000       -0.04199302  -0.004110483  -0.004348775
## 3  0.03762295  -0.041993021        1.00000000   0.126906802   0.081711315
## 4  0.07968324  -0.004110483        0.12690680   1.000000000  -0.032102328
## 5  0.05935058  -0.004348775        0.08171131  -0.032102328   1.000000000
## 6  0.10199224  -0.018678732        0.14620624   0.083880418   0.054194866
##   hours_per_week0
## 1      0.10199224
## 2     -0.01867873
## 3      0.14620624
## 4      0.08388042
## 5      0.05419487
## 6      1.00000000
```

Para contar el número de observaciones de cada clase en una variable categórica, como es en este caso la variable respuesta *salario*, se emplea la función `h2o.table()`.

```
# Se crea una tabla con el número de observaciones de cada tipo.
tabla_muestra <- as.data.frame(h2o.table(datos_h2o$salario))
tabla_muestra

##   salario Count
## 1   <=50K 34014
## 2    >50K 11208
```

```
# Una vez creada la tabla, se carga en el entorno de R para poder graficar.
library(tidyverse)
ggplot(data = tabla_muestra,
       aes(x = salario, y = Count, fill = salario)) +
    geom_col() +
    scale_fill_manual(values = c("gray50", "orangered2")) +
    theme_bw() +
    labs(x = "Salario", y = "Número de observaciones",
         title = "Distribución de la variable Salario") +
    theme(legend.position = "none")
```



Separación de training, validación y test

Evaluar la capacidad predictiva de un modelo consiste en comprobar cómo de próximas son sus predicciones a los verdaderos valores de la variable respuesta. Para poder cuantificar de forma correcta este error, se necesita disponer de un conjunto de observaciones, de las que se conozca la variable respuesta, pero que el modelo no haya "visto", es decir, que no hayan participado en su ajuste. Con esta finalidad, se separan los datos disponibles en un conjunto de entrenamiento, uno de validación y uno de test. El conjunto de entrenamiento se emplea para ajustar el modelo, el de validación para encontrar los mejores hiperparámetros (*model tuning*) y el de test para estimar el error que comete el modelo al exponerse a nuevos datos. El tamaño adecuado de las particiones depende en gran medida de la cantidad de datos disponibles y la seguridad que se necesite en la estimación del error, 70%-15%-15% suele dar buenos resultados.

Si el número de observaciones es limitado, crear 3 particiones puede generar grupos demasiado pequeños y variables. En estos casos, es preferible dividir los datos únicamente en un conjunto de entrenamiento y otro de test, y utilizar validación cruzada (*cross validation*) sobre el conjunto

de entrenamiento durante la optimización de los hiperparámetros del modelo. Para mostrar ambas posibilidades, en el ejemplo de modelo *GLM* se empleará validación cruzada, mientras que en el resto de modelos (*GBM*, *RF* y *Deep Learning*), se emplea un solo conjunto de validación. Es importante tener en cuenta que, la validación cruzada, requiere ajustar el modelo repetidas veces, lo que supone un coste computacional muy alto. Cuando se trabaja con *big data* no suele ser factible aplicar validación cruzada, además, al disponer de tantos datos, un conjunto de validación único es suficiente para obtener buenas estimaciones del comportamiento del modelo. Para una descripción más detallada de los diferentes métodos de validación, consultar el *Anexo4* del documento Machine learning con R y Caret.

La función `h2o.splitFrame()` realiza particiones aleatorias, pero no permite hacerlas de forma estratificada, por lo que no asegura que la distribución de clases de variable respuesta sea igual en todas particiones. Esto puede ser problemático con datos muy desbalanceados (alguno de los grupos es muy minoritario).

```
# Separación de las observaciones en conjunto de entrenamiento y test.
# En los ejemplos de GBM y deep learning se repetirá la separación, pero en
# tres conjuntos en lugar de dos.
particiones <- h2o.splitFrame(data = datos_h2o, ratios = c(0.8), seed = 123)
datos_train_h2o   <- h2o.assign(data = particiones[[1]], key = "datos_train_H2O")
datos_test_h2o    <- h2o.assign(data = particiones[[2]], key = "datos_test_H2O")
```

Número de observaciones de cada clase por partición.

```
h2o.table(datos_train_h2o$salario)
```

```
##   salario Count
## 1   <=50K 27229
## 2    >50K  9009
##
## [2 rows x 2 columns]
```

```
h2o.table(datos_test_h2o$salario)
```

```
##   salario Count
## 1   <=50K  6785
## 2    >50K  2199
##
## [2 rows x 2 columns]
```

```
# En porcentaje
h2o.table(datos_train_h2o$salario)/h2o.nrow(datos_train_h2o)
```

```
##   salario     Count
## 1     NaN 0.7513936
## 2     NaN 0.2486064
```

```
h2o.table(datos_test_h2o$salario)/h2o.nrow(datos_test_h2o)
```

```
##   salario     Count
## 1     NaN 0.7552315
## 2     NaN 0.2447685
##
## [2 rows x 2 columns]
```

La proporción de los dos grupos de la variable respuesta es muy similar en ambas particiones.

Preprocesado de datos

H2O incorpora y automatiza muchas de las transformaciones necesarias para que los datos puedan ser ingeridos por los algoritmos de *machine learning*. Esto es distinto a la mayoría de librerías de *machine learning* de otros lenguajes como Python y R, donde las etapas de preprocesado suelen ser independientes del entrenamiento del modelo. En concreto, H2O automatiza las siguientes transformaciones:

Variables categóricas en H2O

En el momento de ajustar un modelo (*GBM*, *DRF*, *Deep Learning*, *K-Means*, *Aggregator* y *XGBoost*), H2O identifica automáticamente que variables son categóricas y crea internamente las variables *dummy* correspondientes. Es altamente recomendable permitir que H2O realice este proceso en lugar de hacerlo externamente, ya que su implementación está muy optimizada. El comportamiento de la codificación de las variables categóricas puede controlarse con el argumento `categorical_encoding`, por defecto su valor es "AUTO".

Estandarización

Por defecto, H2O estandariza los predictores numéricos antes de ajustar los modelos para que todos tengan media cero y varianza uno. Este comportamiento se puede controlar con el argumento `standardize`. Es importante tener en cuenta que, para muchos modelos (*lasso*, *ridge*, *Deep Learning*...), es necesario realizar la estandarización de predictores.

Eliminación de variables con varianza cero

No se deben de incluir en un modelo predictores que contengan un único valor (varianza cero), ya que no aportan información. Los algoritmos de H2O excluyen directamente las columnas con valor constante. Este comportamiento se puede controlar mediante el argumento `ignore_const_cols`.

Balance de clases

En problemas de clasificación, puede ocurrir que los datos estén desbalanceados, es decir, que el número de observaciones pertenecientes a cada grupo sea muy dispar. Por ejemplo, que de 1000 observaciones, 800 pertenezcan a un grupo y solo 200 al otro. En estos casos, los modelos pueden tener dificultades para aprender a identificar las observaciones del grupo minoritario. Con el argumento `balance_classes` se puede indicar que antes de ajustar el modelo se equilibren las clases mediante *undersampling* u *oversampling*. Por defecto, este comportamiento está desactivado.

Todas las trasformaciones descritas anteriormente se aprenden a partir de los datos de entrenamiento en el momento del ajuste, y se aplican automáticamente cuando el modelo se emplea para predecir nuevas observaciones. De esta forma, se garantiza que no se viola la condición de que ninguna información procedente de las observaciones de test participe o influya en el ajuste del modelo.

H2O dispone de un listado de funciones para manipular y transformar los datos, sin embargo, con frecuencia no son suficientes para resolver todas las necesidades de preprocesado. En estos casos es necesario recurrir a R, si el volumen de datos es limitado, o a *Spark* si la información no se puede cargar en memoria.

Nota: en mi experiencia propia, aunque las tecnologías distribuidas como Spark son muy útiles, antes de recurrir a ellas, recomiendo reflexionar sobre la manipulación que se tiene que realizar y analizar si es necesario que se haga sobre todos los datos a la vez. Por ejemplo, el proceso de imputación se debe aprender únicamente de los datos de entrenamiento y luego aplicarlo al resto de datos. Puede que todo el dataset no se pueda cargar en memoria, pero sí el conjunto de entrenamiento. Esto permitiría aprender la imputación y después aplicarla por separado a cada conjunto.

Modelos con H2O

Modelos disponibles

H2O incorpora los siguientes algoritmos de *machine learning*. Todos ellos están implementados de forma que puedan trabajar, en la medida de lo posible, de forma distribuida y/o en paralelo.

Modelos supervisados

- Cox Proportional Hazards (CoxPH)
- Deep Learning (Neural Networks)
- Distributed Random Forest (DRF)
- Generalized Linear Model (GLM)
- Gradient Boosting Machine (GBM)
- Naïve Bayes Classifier
- Stacked Ensembles
- XGBoost

Modelos no supervisados

- Aggregator
- Generalized Low Rank Models (GLRM)
- K-Means Clustering
- Principal Component Analysis (PCA)

Otros

- Word2vec

En este documento se muestran ejemplos con *Generalized Linear Model (GLM)*, *Gradient Boosting Machine (GBM)* y *Deep Learning (Neural Networks)*.

Parada temprana

Los algoritmos de H2O están pensados para poder ser utilizados de forma eficiente con grandes volúmenes de datos. Emplear miles o millones de datos suele implicar tiempos largos de entrenamiento. H2O incorpora criterios de parada para que no se continúe mejorando un modelo si ya se ha alcanzado una solución aceptable. Dependiendo del algoritmo, el criterio de parada es distinto. En los siguientes apartados se describen con detalle.

Optimización de hiperparámetros

Muchos modelos contienen parámetros que no pueden aprenderse a partir de los datos de entrenamiento y que, por lo tanto, deben de ser establecidos por el analista. A estos se les conoce como hiperparámetros. Los resultados de un modelo pueden depender en gran medida del valor que tomen sus hiperparámetros, sin embargo, no se puede conocer de antemano cuál es el adecuado. Aunque con la práctica, los especialistas en *machine learning* ganan intuición sobre qué valores pueden funcionar mejor en cada problema, no hay reglas fijas. La forma más común de encontrar los valores óptimos es probando diferentes posibilidades, lo que comúnmente se conoce como *grid search*.

H2O emplea la función `h2o.grid()` para realizar la búsqueda de los mejores hiperparámetros, sus argumentos principales son: el nombre del algoritmo, los parámetros del algoritmo, una lista con los valores de los hiperparámetros que se quieren comparar, el tipo de búsqueda (*"Cartesian"* o *"RandomDiscrete"*) y , si es de tipo *random*, un criterio de parada. Una vez que la búsqueda ha finalizado, el objeto `grid` creado contiene todos los modelos, para acceder a ellos es necesario extraerlos.

Generalized linear models (GLMs)

Introducción

El modelo lineal generalizado surge como resultado de generalizar y unificar diferentes modelos lineales tradicionales (regresión lineal por mínimos cuadrados, regresión logística...), bajo un mismo marco.

En el modelo de regresión lineal por mínimos cuadrados, la variable respuesta y está linealmente relacionada con el o los predictores X acorde a la ecuación:

$$y = \beta_0 + X^T\beta + \epsilon$$

Para que esta aproximación sea válida, se necesita que el error se distribuya de forma normal y que la varianza sea constante. Estos requisitos hacen que el modelo no se aplicable a muchos problemas reales. EL GLM permite solucionar esta limitación haciendo una abstracción a un modelo más genérico en el que la distribución del error puede seguir distintas funciones *link* (Gaussian, Poisson, binomial...). Por ejemplo, el modelo lineal generalizado equivale al modelo lineal por mínimos cuadrados cuando la función *link* es gausiana, o a la regresión logística cuando es binomial (*logit*). Además, el modelo lineal generalizado puede incluir regularización durante su ajuste, por lo que también incluye los modelos ridge regression, lasso y elastic net.

El ajuste de todos los modelos GLM consiste en identificar el valor de los coeficientes β_0 y β que maximizan la verosimilitud (*likelihood*) de los datos, es decir, los valores de los coeficientes que dan lugar al modelo que con mayor probabilidad puede haber generado los datos observados. Cuando se emplea una función *link* gausiana (con o sin regularización *L2*), la máxima verosimilitud equivale a minimizar la suma de errores cuadrados, lo que tiene una solución analítica. Para el resto de familias o cuando se emplea la regularización *L1*, no existe una solución analítica con la que encontrar la máxima verosimilitud, por lo que se requiere un método de optmimización iterativo como el *iteratively reweighted least squares* (IRLSM), L-BFGS, método de Newton o descenso de gradiente. H2O selecciona automáticamente el método apropiado en dependiendo de la función link especificada por el usuario.

Regularización

H2O incorpora 3 tipos de regularización para los modelos GLM con el objetivo de evitar *overfitting*, reducir varianza y atenuar el efecto de la correlación entre predictores. Por lo general, aplicando regularización se consigue modelos con mayor poder predictivo (generalización).

- El modelo lasso es un modelo lineal por mínimos cuadrados que incorpora una regularización que penaliza la suma del valor absolutos de los coeficientes de regresión ($||\beta||_1 = \sum_{k=1}^{p} |\beta_k|$). Ha esta penalización se le conoce como *l1* y tiene el efecto de forzar a que los coeficientes de los predictores tiendan a cero. Dado que un predictor con coeficiente de regresión cero no influye en el modelo, *lasso* consigue seleccionar los predictores más influyentes. El grado de penalización está controlado por el hiperparámetro λ. Cuando $\lambda = 0$, el resultado es equivalente al de un modelo lineal por mínimos cuadrados ordinarios. A medida que λ aumenta, mayor es la penalización y más predictores quedan excluidos.

- El modelo ridge es un modelo lineal por mínimos cuadrados que incorpora una regularización que penaliza la suma de los coeficientes de regresión elevados al cuadrado ($||\beta||_2^2 = \sum_{k=1}^{p} \beta_k^2$). Ha esta penalización se le conoce como *l2* y tiene el efecto de reducir de forma proporcional el valor de todos los coeficientes del modelo pero sin que estos lleguen a cero. Al igual que *lasso*, el grado de penalización está controlado por el hiperparámetro λ.

La principal diferencia práctica entre *lasso* y *ridge* es que el primero consigue que algunos coeficientes sean exactamente cero, por lo que realiza selección de predictores, mientras que el segundo no llega a excluir ninguno. Esto supone una ventaja notable de *lasso* en escenarios donde no todos los predictores son importantes para el modelo y se desea que los menos influyentes queden excluidos. Por otro lado, cuando existen predictores altamente correlacionados (linealmente), *ridge* reduce la influencia de todos ellos a la vez y de forma proporcional, mientras que *lasso* tiende a seleccionar uno de ellos, dándole todo el peso y excluyendo al resto. En presencia de correlaciones, esta selección varía mucho con pequeñas perturbaciones (cambieos en los datos de entrenamiento), por lo que, las soluciones de *lasso*, son muy inestables si los predictores están altamente correlacionados.

Para conseguir un equilibrio óptimo entre estas dos propiedades, se puede emplear lo que se conoce como penalización *elastic net*, que combina ambas estrategias.

- El modelo *elastic net* incluye una regularización que combina la penalización *l1* y *l2* ($\alpha\lambda||\beta||_1 + \frac{1}{2}(1-\alpha)||\beta||_2^2$). El grado en que influye cada una de las penalizaciones está controlado por el hiperparámetro α. Su valor debe estar comprendido en el intervalo [0,1], cuando $\alpha = 0$, H2O aplica *ridge regression* y cuando $\alpha = 1$ aplica *lasso*. La combinación de ambas penalizaciones suele dar lugar a buenos resultados. Una estrategia frecuentemente

utilizada es asignarle casi todo el peso a la penalización *l1* (α muy próximo a 1) para conseguir seleccionar predictores y un poco a la *l2* para dar cierta estabilidad en el caso de que algunos predictores estén correlacionados.

Encontrar el mejor modelo GLM implica identificar los valores óptimos de los hiperparámetros de regularización α y λ. Por defecto, H2O selecciona $\alpha = 0.5$ y, con cada uno de ellos, analiza un rango de valores λ. También es posible hacer una búsqueda del valor óptimo de α mediante un *grid seach*.

Lambda Search

La búsqueda del valor óptimo de *lambda* se habilita indicando `lambda_search = TRUE`. Además, esta búsqueda puede ser configurada con los siguientes argumentos:

- `alpha`: el valor de α que se emplea en el modelo para distribuir la penalización *l1* y *l2*. Por defecto, `alpha= 0.5`.

- `validation_frame`: método de validación empleado para identificar el mejor modelo, puede ser mediante un único conjunto de validación o mediante validación cruzada (*cross-validation*). En el segundo caso, no es necesario crear un conjunto de validación, las particiones se generan a partir del conjunto de entrenamiento.

- `n_folds`: número de particiones en la validación cruzada.

- `lambda_min_ratio` y `nlambdas`: determinan la secuencia de valores *lambda* que se generan automáticamente y sobre los que se realiza la búsqueda. El rango de valores va desde el valor mínimo con el que se alcanza una penalización total (*lambda_max*), es decir, el valor de *lambda* a partir del cual el coeficiente de todos los predictores es cero (valor identificado automáticamente por H2O), hasta un valor de *lambda* igual a *lambda_min_ratio x lambda_max*. Dentro de ese rango, se generan un número de *lambdas* igual al valor especificado con nlambdas. Por defecto, se emplean `nlambda = 100` y `lambda_min_ratio = 1e-4`.

- `max_active_predictors`: establece el número máximo de predictores incluidos en el modelo (predictores con coeficientes de regresión distintos de cero). Esta opción es muy útil para evitar tiempos de computación excesivos cuando se trabaja con sets de datos que contienen cientos de predictores pero se sabe que solo unos pocos son importantes.

Método de ajuste

Todos los modelos GLM se ajustan encontrando el valor de los coeficientes β que maximizan el *likelihood*. La estimación de estos coeficientes se realiza mediante métodos de optimización iterativos, ya que, a excepción de unos pocos casos, no existe una solución analítica. H2O incorpora dos métodos (*solvers*) distintos, cuya correcta elección, que depende del tipo de datos, puede influir notablemente en el rendimiento computacional.

- *Iteratively Reweighted Least Squares Method (IRLSM)*: funciona especialmente bien cuando el set de datos empleado tiene muchas observaciones (largo) pero un número limitado de predictores (estrecho). Los autores recomiendan utilizarlo cuando el número de predictores no es mayor de 500.
- *Limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm (L-BFGS)*: es el método recomendado cuando se dispone de muchos predictores (>500).

Las anteriores son solo recomendaciones generales, en caso de duda, la mejor forma de elegir es comparando el resultado de ambos *solvers*.

Criterio de parada

Los algoritmos de H2O están pensados para poder ser aplicados a grandes volúmenes de datos, por lo que, entre sus características, incorporan criterios de parada para que no se continúe mejorando un modelo si ya se ha alcanzado una solución aceptable. En el caso de los GLM, el criterio de parada es el número máximo de predictores permitidos, que se especifica mediante el argumento `max_active_predictors`.

Modelo

El problema que se describió en la introducción sobre predecir el salario es un problema de clasificación binaria. El principal modelo lineal empleado para clasificaciones binarias es la regresión logística, que es el resultado de emplear, en un modelo lineal generalizado, la función *logit* como link. Aunque la selección se puede hacer manualmente, H2O es capaz de identificar el tipo de variable respuesta (numérica o categórica) y en función de eso seleccionar automáticamente el tipo de modelo, regresión o clasificación.

```
# Se comprueba que la variable respuesta es de tipo factor.
datos_train_h2o$salario <- h2o.asfactor(datos_train_h2o$salario)
datos_test_h2o$salario <- h2o.asfactor(datos_test_h2o$salario)
h2o.isfactor(datos_train_h2o$salario)
```

```
## [1] TRUE
```

```
# Se define la variable respuesta y los predictores.
var_respuesta <- "salario"
# Para este modelo se emplean todos los predictores disponibles.
predictores   <- setdiff(h2o.colnames(datos_h2o), var_respuesta)

# Ajuste del modelo y validación mediente 5-CV para estimar su error.
modelo_binomial <- h2o.glm(
                     y = var_respuesta,
                     x = predictores,
                     training_frame = datos_train_h2o,
                     family = "binomial",
                     link = "logit",
                     standardize = TRUE,
                     balance_classes = FALSE,
                     ignore_const_cols = TRUE,
                     # Se especifica que hacer con observaciones incompletas
                     missing_values_handling = "Skip",
                     # Se hace una búsqueda del hiperparámetro lamba.
                     lambda_search = TRUE,
                     # Selección automática del solver adecuado.
                     solver = "AUTO",
                     alpha = 0.95,
                     # Validación cruzada de 5 folds para estimar el error
                     # del modelo.
                     seed = 123,
                     nfolds = 5,
                     # Reparto estratificado de las observaciones en la creación
                     # de las particiones.
                     fold_assignment = "Stratified",
                     keep_cross_validation_predictions = FALSE,
                     model_id = "modelo_binomial"
                    )

modelo_binomial
```

```
## Model Details:
## ==============
##
## H2OBinomialModel: glm
## Model ID:  modelo_binomial
## GLM Model: summary
##     family  link                                regularization
## 1 binomial logit Elastic Net (alpha = 0.95, lambda = 5.09E-5 )
```

```
##                                                                 lambda_search
## 1 nlambda = 100, lambda.max = 0.1519, lambda.min = 5.09E-5, lambda.1se = 5.21E-4
##   number_of_predictors_total number_of_active_predictors
## 1                         53                          45
##   number_of_iterations   training_frame
## 1                  136 RTMP_sid_81b6_11
##
## Coefficients: glm coefficients
##                       names coefficients standardized_coefficients
## 1                 Intercept    -7.429849                 -2.660849
## 2    occupation.Adm-clerical     0.000000                  0.000000
## 3    occupation.Craft-repair     0.017865                  0.017865
## 4 occupation.Exec-managerial     0.695254                  0.695254
## 5 occupation.Farming-fishing    -1.067297                 -1.067297
##
## ---
##               names coefficients standardized_coefficients
## 49              age     0.025490                  0.337455
## 50     final_weight     0.000001                  0.073707
## 51 education_number     0.203311                  0.519740
## 52     capital_gain     0.000313                  2.373427
## 53     capital_loss     0.000630                  0.255137
## 54   hours_per_week     0.029095                  0.348779
##
## H2OBinomialMetrics: glm
## ** Reported on training data. **
##
## MSE:  0.1046908
## RMSE:  0.3235596
## LogLoss:  0.3269983
## Mean Per-Class Error:  0.1912179
## AUC:  0.9045175
## Gini:  0.8090349
## R^2:  0.4395605
## Residual Deviance:  23699.53
## AIC:  23791.53
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##        <=50K  >50K    Error         Rate
## <=50K  23332  3897 0.143119  =3897/27229
## >50K    2156  6853 0.239316   =2156/9009
## Totals 25488 10750 0.167035  =6053/36238
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                       max f1  0.333745 0.693659 218
## 2                       max f2  0.145824 0.792694 296
## 3                 max f0point5  0.583384 0.710063 130
## 4                 max accuracy  0.503282 0.848419 157
## 5                max precision  0.996615 0.992016   2
```

```
## 6                   max recall  0.001393 1.000000 398
## 7              max specificity  0.999827 0.999853   0
## 8             max absolute_mcc  0.400279 0.588165 193
## 9   max min_per_class_accuracy  0.277348 0.817143 241
## 10 max mean_per_class_accuracy  0.244263 0.821387 255
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLif
t(<model>, valid=<T/F>, xval=<T/F>)`
##
## H2OBinomialMetrics: glm
## ** Reported on cross-validation data. **
## ** 5-fold cross-validation on training data (Metrics computed for combined holdo
ut predictions) **
##
## MSE:  0.1051063
## RMSE:  0.324201
## LogLoss:  0.3282526
## Mean Per-Class Error:  0.1911896
## AUC:  0.9037092
## Gini:  0.8074184
## R^2:  0.4373362
## Residual Deviance:  23790.44
## AIC:  23880.44
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##        <=50K  >50K    Error         Rate
## <=50K  23261  3968 0.145727  =3968/27229
## >50K    2132  6877 0.236652   =2132/9009
## Totals 25393 10845 0.168332  =6100/36238
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                        max f1  0.330819 0.692757 220
## 2                        max f2  0.141983 0.791447 304
## 3                  max f0point5  0.566768 0.709278 136
## 4                  max accuracy  0.480893 0.847977 165
## 5                 max precision  0.997895 0.990719   1
## 6                    max recall  0.000918 1.000000 399
## 7              max specificity  0.999759 0.999853   0
## 8             max absolute_mcc  0.383599 0.585549 200
## 9   max min_per_class_accuracy  0.279008 0.816628 243
## 10 max mean_per_class_accuracy  0.226527 0.820319 266
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLif
t(<model>, valid=<T/F>, xval=<T/F>)`
## Cross-Validation Metrics Summary:
##                 mean           sd cv_1_valid cv_2_valid cv_3_valid
## accuracy   0.8370137 0.0047676703  0.8446949  0.8424983 0.82581276
## auc       0.90377855 0.0016269764 0.90619415  0.9066515 0.90286505
## err       0.16298628 0.0047676703 0.15530512  0.1575017 0.17418721
```

```
## err_count     1180.8    26.815666     1130.0     1160.0     1243.0
## f0point5   0.6677162  0.009476489  0.6725798  0.6883655 0.64861405
##           cv_4_valid cv_5_valid
## accuracy   0.8383588  0.8337038
## auc       0.90059686  0.9025852
## err        0.1616412  0.1662962
## err_count     1174.0     1197.0
## f0point5   0.6701031  0.6589185
##
## ---
##                          mean          sd cv_1_valid cv_2_valid cv_3_valid
## precision          0.65078366 0.014271369  0.6546621 0.67998004  0.6208406
## r2                  0.4371859 0.004227429 0.43763807 0.44788808 0.43729803
## recall             0.74796927 0.019931803 0.75526464 0.72408295  0.7899721
## residual_deviance   4757.921   33.133957  4699.8945  4820.0493  4707.5337
## rmse               0.32420433 0.001208666 0.32094553 0.32402903  0.3254827
## specificity         0.8664767 0.011873075  0.8731654  0.8831145  0.8378581
##                    cv_4_valid cv_5_valid
## precision          0.66078836 0.63764703
## r2                 0.43105683 0.43204856
## recall             0.71014494  0.7603816
## residual_deviance   4789.476   4772.651
## rmse               0.32529965  0.3252647
## specificity        0.88041687  0.8578287
```

Al imprimir el modelo por pantalla, se muestra toda la información disponible: tipo de modelo, coeficientes de regresión obtenidos, métricas, resultados de la validación cruzada... Esta suele ser más información de la que se necesita en un análisis convencional. Para poder acceder directamente a la información de interés, H2O incorpora una serie de funciones que extraen información concreta del modelo.

Coeficientes de regresión

Si se especifica en la creación del modelo que se estandaricen los predictores (`standardize = TRUE`), se generan dos tipos de coeficientes de regresión:

- Coeficientes estandarizados: son los obtenidos directamente por el modelo al emplear los predictores estandarizados. Es importante tener en cuenta que, al estar estandarizados, su valor no está en las mismas unidades que los predictores originales, lo que dificulta su interpretación. Sin embargo, sí que son útiles (su valor absoluto) para ordenar los predictores de mayor a menor influencia en el modelo.

- Coeficientes: son los coeficientes "normales" obtenidos a partir de los coeficientes estandarizados pero revirtiendo la estandarización. Sus unidades son las mismas que las de los predictores originales y pueden ser interpretados de forma normal.

Para obtener más información sobre la correcta interpretación de los coeficientes de regresión de un modelo de regresión logística consultar Regresión logística simple y múltiple en R.

```
# Coeficientes de correlación de cada uno de los predictores del modelo.
modelo_binomial@model$coefficients_table
```

```
## Coefficients: glm coefficients
##                         names coefficients standardized_coefficients
## 1                   Intercept    -7.429849                 -2.660849
## 2    occupation.Adm-clerical     0.000000                  0.000000
## 3    occupation.Craft-repair     0.017865                  0.017865
## 4 occupation.Exec-managerial     0.695254                  0.695254
## 5 occupation.Farming-fishing    -1.067297                 -1.067297
##
## ---
##                names coefficients standardized_coefficients
## 49               age     0.025490                  0.337455
## 50     final_weight     0.000001                  0.073707
## 51 education_number     0.203311                  0.519740
## 52     capital_gain     0.000313                  2.373427
## 53     capital_loss     0.000630                  0.255137
## 54   hours_per_week     0.029095                  0.348779
```

```
# Para mostrarlos todos.
as.data.frame(modelo_binomial@model$coefficients_table)
```

```
##                          names  coefficients standardized_coefficients
## 1                    Intercept -7.429849e+00                -2.66084924
## 2       occupation.Adm-clerical  0.000000e+00                 0.00000000
## 3       occupation.Craft-repair  1.786458e-02                 0.01786458
## 4    occupation.Exec-managerial  6.952535e-01                 0.69525354
## 5    occupation.Farming-fishing -1.067297e+00                -1.06729733
## 6  occupation.Handlers-cleaners -7.143720e-01                -0.71437197
## 7  occupation.Machine-op-inspct -3.208084e-01                -0.32080841
## 8      occupation.Other-service -9.186435e-01                -0.91864352
## 9    occupation.Priv-house-serv -9.238011e-01                -0.92380108
## 10    occupation.Prof-specialty  5.366986e-01                 0.53669864
## 11   occupation.Protective-serv  3.561468e-01                 0.35614679
## 12             occupation.Sales  2.209898e-01                 0.22098981
## 13      occupation.Tech-support  4.542207e-01                 0.45422070
## 14  occupation.Transport-moving -1.256209e-01                -0.12562088
## 15        workclass.Federal-gov  5.929299e-01                 0.59292994
## 16          workclass.Local-gov  0.000000e+00                 0.00000000
## 17            workclass.Private  1.381303e-01                 0.13813026
## 18       workclass.Self-emp-inc  3.485436e-01                 0.34854357
```

```
## 19   workclass.Self-emp-not-inc -3.916796e-01               -0.39167963
## 20          workclass.State-gov -1.724626e-01               -0.17246255
## 21       workclass.Without-pay  0.000000e+00                0.00000000
## 22      education.bachillerato -6.281495e-02               -0.06281495
## 23         education.doctorado  4.334823e-01                0.43348229
## 24            education.master  2.762277e-01                0.27622771
## 25          education.primaria -5.350141e-01               -0.53501406
## 26        education.secundaria -1.776797e-01               -0.17767966
## 27       education.universidad  1.160438e-01                0.11604375
## 28        relationship.Husband  9.352626e-01                0.93526257
## 29  relationship.Not-in-family  0.000000e+00                0.00000000
## 30 relationship.Other-relative -2.024166e-01               -0.20241657
## 31      relationship.Own-child -7.954375e-01               -0.79543755
## 32      relationship.Unmarried -2.541777e-01               -0.25417773
## 33           relationship.Wife  2.024050e+00                2.02405034
## 34     race.Amer-Indian-Eskimo -1.370965e-01               -0.13709648
## 35     race.Asian-Pac-Islander  3.481366e-01                0.34813661
## 36                  race.Black  0.000000e+00                0.00000000
## 37                  race.Other -6.805539e-02               -0.06805539
## 38                  race.White  1.592144e-01                0.15921436
## 39         native_country.Asia  0.000000e+00                0.00000000
## 40       native_country.Europa  3.423929e-01                0.34239290
## 41 native_country.Norte America  2.917642e-01                0.29176422
## 42        native_country.Otros -1.109116e+00               -1.10911609
## 43  native_country.Sud America -6.341519e-01               -0.63415188
## 44       marital_status.casado  8.241046e-01                0.82410463
## 45     marital_status.separado  0.000000e+00                0.00000000
## 46      marital_status.soltero -5.026763e-01               -0.50267633
## 47                  sex.Female -6.943990e-01               -0.69439900
## 48                    sex.Male  0.000000e+00                0.00000000
## 49                         age  2.549035e-02                0.33745520
## 50               final_weight  6.970095e-07                0.07370664
## 51           education_number  2.033113e-01                0.51974033
## 52               capital_gain  3.132278e-04                2.37342741
## 53               capital_loss  6.302363e-04                0.25513691
## 54             hours_per_week  2.909483e-02                0.34877852
```

Predictores incluidos en el modelo

Si el modelo GLM incorpora penalización *L1* (*lasso* o *elastic net*), entonces, el propio ajuste del modelo realiza selección de predictores. Los predictores incluidos en el modelo son aquellos cuyo coeficiente de regresión es distinto de cero.

```
# Predictores incluidos.
names(modelo_binomial@model$coefficients[modelo_binomial@model$coefficients > 0])
```

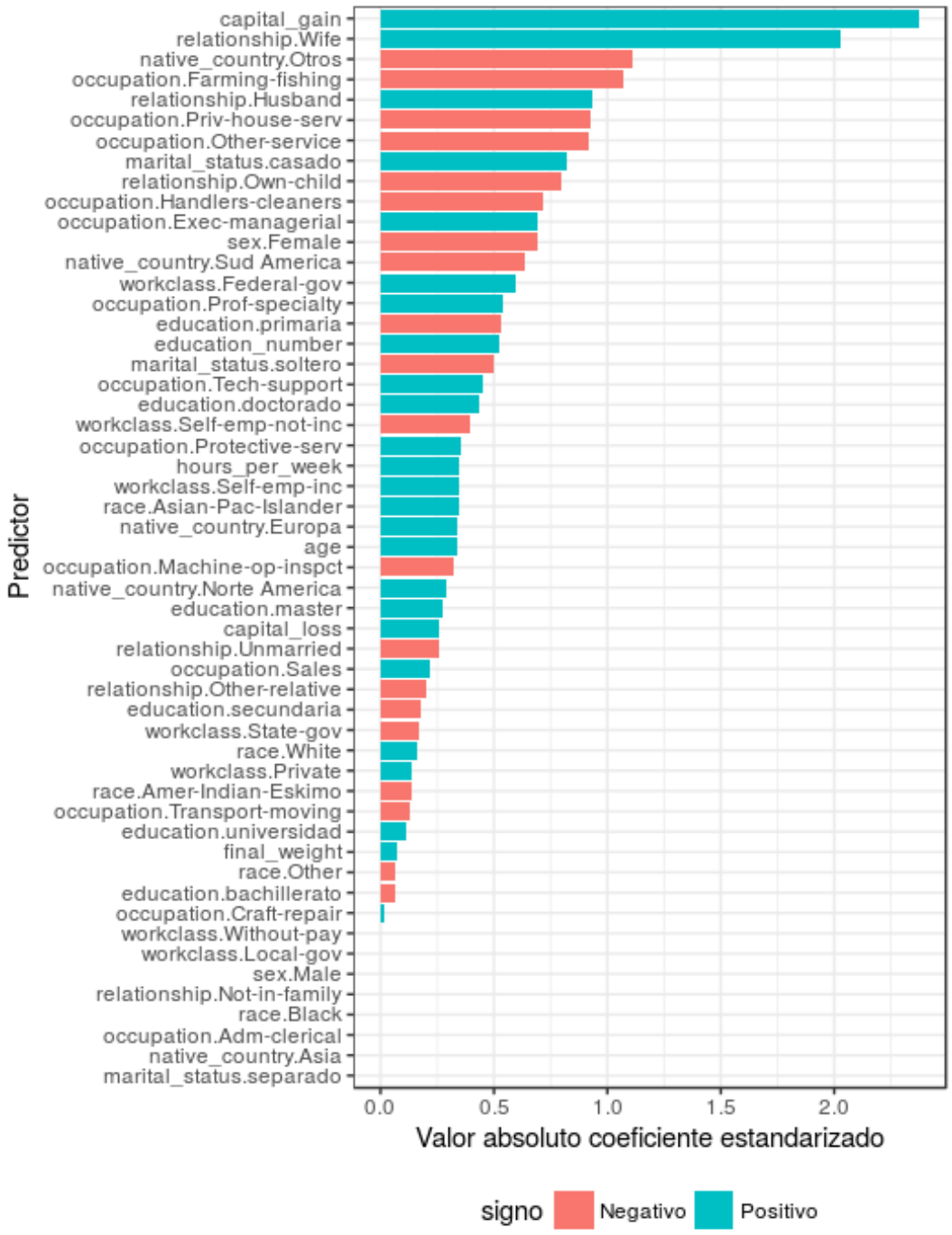
```
##  [1] "occupation.Craft-repair"      "occupation.Exec-managerial"
##  [3] "occupation.Prof-specialty"    "occupation.Protective-serv"
##  [5] "occupation.Sales"             "occupation.Tech-support"
##  [7] "workclass.Federal-gov"        "workclass.Private"
##  [9] "workclass.Self-emp-inc"       "education.doctorado"
## [11] "education.master"             "education.universidad"
## [13] "relationship.Husband"         "relationship.Wife"
## [15] "race.Asian-Pac-Islander"      "race.White"
## [17] "native_country.Europa"        "native_country.Norte America"
## [19] "marital_status.casado"        "age"
## [21] "final_weight"                 "education_number"
## [23] "capital_gain"                 "capital_loss"
## [25] "hours_per_week"
```

Importancia de los predictores

En los modelos lineales generalizados, la importancia de los predictores puede estudiarse a partir del valor absoluto de los coeficientes de regresión estandarizados.

```
coeficientes <- as.data.frame(modelo_binomial@model$coefficients_table)

# Se excluye el intercept.
coeficientes <- coeficientes %>% filter(names != "Intercept")

# Se calcula el valor absoluto.
coeficientes <- coeficientes %>%
                mutate(abs_stand_coef = abs(standardized_coefficients))

# Se añade una variable con el signo del coeficiente.
coeficientes <- coeficientes %>%
                mutate(signo = if_else(standardized_coefficients > 0,
                                       "Positivo",
                                       "Negativo"))
ggplot(data = coeficientes,
       aes(x = reorder(names, abs_stand_coef),
           y = abs_stand_coef,
           fill = signo)) +
  geom_col() +
  coord_flip() +
  labs(title = "Importancia de los predictores en el modelo GLM",
       x     = "Predictor",
       y     = "Valor absoluto coeficiente estandarizado") +
  theme_bw() +
  theme(legend.position = "bottom")
```

Importancia de los predictores en el modelo




```
# Equivalente:
# h2o.varimp(modelo_binomial)
# h2o.varimp_plot(modelo_binomial)
```

Métricas de entrenamiento

Se pueden obtener toda una serie de métricas calculadas a partir de los datos de entrenamiento. Al tratarse de métricas de entrenamiento, no son válidas como estimación del error que comete el modelo al predecir nuevos valores (son demasiado optimistas) ni tampoco para comparar modelos, para esto hay que emplear métricas de validación o de test.

```
# Área bajo la curva
h2o.auc(modelo_binomial, train = TRUE)

## [1] 0.9045175

# Mean Squared Error
h2o.mse(modelo_binomial, train = TRUE)
# R2
h2o.r2(modelo_binomial, train = TRUE)
# LogLoss
h2o.logloss(modelo_binomial, train = TRUE)
# Coeficiente de Gini
h2o.giniCoef(modelo_binomial, train = TRUE)
# Desviance del modelo nulo
h2o.null_deviance(modelo_binomial, train = TRUE)
# Desviance del modelo final
h2o.residual_deviance(modelo_binomial, train = TRUE)
# AIC
h2o.aic(modelo_binomial, train = TRUE)
```

Se pueden obtener todas a la vez con la función `h2o.performance()`.

```
h2o.performance(model = modelo_binomial, train = TRUE)
```

```
## H2OBinomialMetrics: glm
## ** Reported on training data. **
##
## MSE:  0.1046908
## RMSE:  0.3235596
## LogLoss:  0.3269983
## Mean Per-Class Error:  0.1912179
## AUC:  0.9045175
## Gini:  0.8090349
## R^2:  0.4395605
## Residual Deviance:  23699.53
## AIC:  23791.53
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##        <=50K  >50K    Error         Rate
## <=50K  23332  3897 0.143119  =3897/27229
## >50K    2156  6853 0.239316   =2156/9009
```

```
## Totals 25488 10750 0.167035  =6053/36238
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                       max f1  0.333745 0.693659 218
## 2                       max f2  0.145824 0.792694 296
## 3                 max f0point5  0.583384 0.710063 130
## 4                 max accuracy  0.503282 0.848419 157
## 5                max precision  0.996615 0.992016   2
## 6                   max recall  0.001393 1.000000 398
## 7              max specificity  0.999827 0.999853   0
## 8             max absolute_mcc  0.400279 0.588165 193
## 9   max min_per_class_accuracy  0.277348 0.817143 241
## 10 max mean_per_class_accuracy  0.244263 0.821387 255
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLif
t(<model>, valid=<T/F>, xval=<T/F>)`
```

```
# Equivalente a:
# modelo_binomial@model$training_metrics
```

Métricas de validación y validación cruzada

Al igual que para el entrenamiento, se pueden obtener las métricas obtenidas en el conjunto de validación (`train=TRUE`) o el promedio obtenido mediante validación cruzada (`xval=TRUE`).

```
# Área bajo la curva
h2o.auc(modelo_binomial, xval = TRUE)
```

```
## [1] 0.9037092
```

```
h2o.performance(model = modelo_binomial, xval = TRUE)
```

```
## H2OBinomialMetrics: glm
## ** Reported on cross-validation data. **
## ** 5-fold cross-validation on training data (Metrics computed for combined holdo
ut predictions) **
##
## MSE:  0.1051063
## RMSE:  0.324201
## LogLoss:  0.3282526
## Mean Per-Class Error:  0.1911896
## AUC:  0.9037092
## Gini:  0.8074184
## R^2:  0.4373362
## Residual Deviance:  23790.44
```

```
## AIC:  23880.44
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##        <=50K  >50K    Error         Rate
## <=50K  23261  3968 0.145727  =3968/27229
## >50K    2132  6877 0.236652   =2132/9009
## Totals 25393 10845 0.168332  =6100/36238
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                       max f1  0.330819 0.692757 220
## 2                       max f2  0.141983 0.791447 304
## 3                 max f0point5  0.566768 0.709278 136
## 4                 max accuracy  0.480893 0.847977 165
## 5                max precision  0.997895 0.990719   1
## 6                   max recall  0.000918 1.000000 399
## 7              max specificity  0.999759 0.999853   0
## 8             max absolute_mcc  0.383599 0.585549 200
## 9   max min_per_class_accuracy  0.279008 0.816628 243
## 10 max mean_per_class_accuracy  0.226527 0.820319 266
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLif
t(<model>, valid=<T/F>, xval=<T/F>)`
```

Predicciones

Una vez que el modelo ha sido entrenado, puede emplearse para predecir nuevas observaciones con la función `h2o.predict()`, que recibe como argumentos un modelo y un nuevo set de datos.

```
predicciones <- h2o.predict(object = modelo_binomial, newdata = datos_test_h2o)

predicciones
```

```
##   predict      <=50K       >50K
## 1    >50K 0.09031284 0.90968716
## 2    >50K 0.08783268 0.91216732
## 3   <=50K 0.94674951 0.05325049
## 4   <=50K 0.70345153 0.29654847
## 5   <=50K 0.92590110 0.07409890
## 6    >50K 0.42091423 0.57908577
##
## [8984 rows x 3 columns]
```

El resultado devuelto por la función `h2o.predict()` para un modelo GLM logístico es una tabla con 3 columnas, una con la clase predicha y otras dos con la probabilidad de pertenecer a cada una de las clases. El orden de las filas es el mismo que el de los datos pasados al argumento `newdata`.

Si el nuevo set de datos incluye la variable respuesta, se pueden calcular métricas que cuantifiquen el grado de acierto.

```
h2o.performance(model = modelo_binomial, newdata = datos_test_h2o)
```

```
## H2OBinomialMetrics: glm
##
## MSE:  0.1027842
## RMSE:  0.3205997
## LogLoss:  0.3209669
## Mean Per-Class Error:  0.1936068
## AUC:  0.9058081
## Gini:  0.8116161
## R^2:  0.4439796
## Residual Deviance:  5767.134
## AIC:  5859.134
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##        <=50K >50K    Error        Rate
## <=50K   5935  850 0.125276   =850/6785
## >50K     576 1623 0.261937   =576/2199
## Totals  6511 2473 0.158727  =1426/8984
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                       max f1  0.364758 0.694777 208
## 2                       max f2  0.128946 0.790420 310
## 3                 max f0point5  0.668525 0.715008 104
## 4                 max accuracy  0.472887 0.851514 169
## 5                max precision  0.999922 1.000000   0
## 6                   max recall  0.003821 1.000000 395
## 7              max specificity  0.999922 1.000000   0
## 8             max absolute_mcc  0.396981 0.590386 195
## 9   max min_per_class_accuracy  0.278579 0.817539 245
## 10 max mean_per_class_accuracy  0.258651 0.820233 253
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLif
t(<model>, valid=<T/F>, xval=<T/F>)`
```

```
# Cálculo manual de accuracy
mean(as.vector(predicciones$predict) == as.vector(datos_test_h2o$salario))
```

```
## [1] 0.8323687
```

Grid search

En el modelo anterior, se ha hecho una búsqueda del valor óptimo de *lambda* pero no de *alpha* (se ha empleado un valor fijo de `alpha=0.95`). A continuación, se repite el ajuste del modelo, pero esta vez, comparando también distintos valores de *alpha*. Para estimar la capacidad predictiva de cada modelo se emplea validación cruzada con 10 particiones.

Nota: con el número de observaciones disponible, emplear validación simple sería suficiente y los resultados se obtendrían mucho más rápido.

```
# Valores de alpha que se van a comparar.
hiperparametros <- list(alpha = c(0, 0.1, 0.5, 0.95, 1))

grid_glm <- h2o.grid(
    # Algoritmo y parámetros.
    algorithm = "glm",
    family = "binomial",
    link = "logit",
    # Variable respuesta y predictores.
    y = var_respuesta,
    x = predictores,
    # Datos de entrenamiento.
    training_frame = datos_train_h2o,
    # Preprocesado.
    standardize = TRUE,
    missing_values_handling = "Skip",
    ignore_const_cols = TRUE,
    # Hiperparámetros.
    hyper_params = hiperparametros,
    # Tipo de búsqueda.
    search_criteria = list(strategy = "Cartesian"),
    lambda_search = TRUE,
    # Selección automática del solver adecuado.
    solver = "AUTO",
    # Estrategia de validación para seleccionar el mejor modelo.
    seed = 123,
    nfolds = 10,
    # Reparto estratificado de las observaciones en la creación
    # de las particiones.
    fold_assignment = "Stratified",
    keep_cross_validation_predictions = FALSE,
    grid_id = "grid_glm")

# Se muestran los modelos ordenados de mayor a menor AUC.
resultados_grid <- h2o.getGrid(grid_id = "grid_glm", sort_by = "auc",
                               decreasing = TRUE)
print(resultados_grid)
```

```
## H2O Grid Details
## ================
##
## Grid ID: grid_glm
## Used hyper parameters:
##   -  alpha
## Number of models: 5
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by decreasing auc
##    alpha        model_ids                auc
## 1  [0.0] grid_glm_model_0 0.9036622723317056
## 2  [0.1] grid_glm_model_1 0.9036252247350709
## 3  [1.0] grid_glm_model_4 0.9036227013567348
## 4 [0.95] grid_glm_model_3 0.9036161034765463
## 5  [0.5] grid_glm_model_2 0.9036057429498247
```

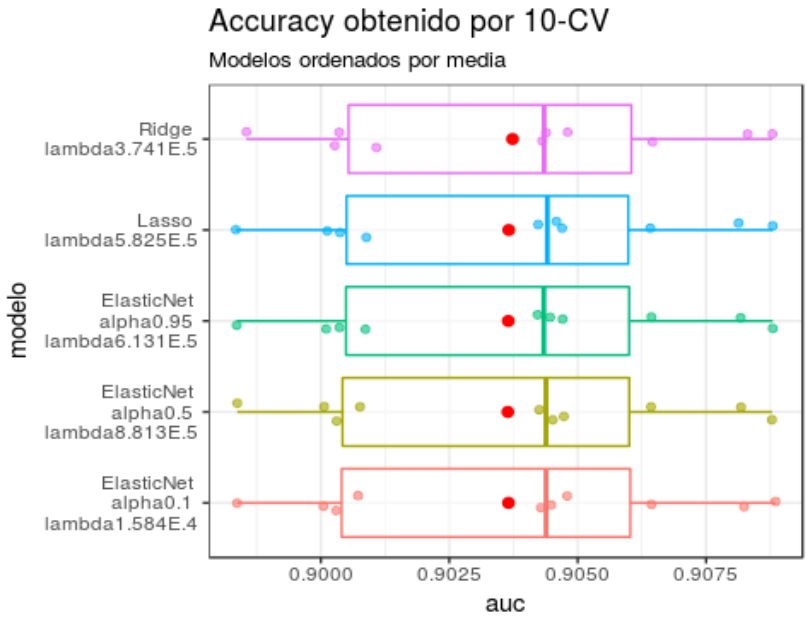
```
# resultados_grid@summary_table
```

De entre los valores comparados, el mayor *accuracy* promedio de validación cruzada se obtiene con `alpha=0.0` (modelo *ridge*), aunque para este problema, la diferencia es mínima.

Para conocer con más detalle la distribución de *accuracy* obtenido por cada modelo en cada partición de la validación cruzada, se tiene que extraer la información de cada uno de los modelos.

```
# Identificador de los modelos creados por validación cruzada.
id_modelos <- unlist(resultados_grid@model_ids)

# Se crea una lista donde se almacenarán los resultados.
auc_xvalidacion <- vector(mode = "list", length = length(id_modelos))

# Se recorre cada modelo almacenado en el grid y se extraen la métrica (auc)
# obtenida en cada partición.
for (i in seq_along(id_modelos)) {
  modelo <- h2o.getModel(resultados_grid@model_ids[[i]])
  metricas_xvalidacion_modelo <- modelo@model$cross_validation_metrics_summary
  names(auc_xvalidacion)[i]   <- modelo@model$model_summary$regularization
  auc_xvalidacion[[i]] <- as.numeric(metricas_xvalidacion_modelo["auc", -c(1,2)])
}
# Se eliminan los espacios en blanco del nombre de los modelos.
library(stringr)
names(auc_xvalidacion) <- str_remove_all(string = names(auc_xvalidacion),
                                         pattern = "[ )=]")
names(auc_xvalidacion) <- str_replace_all(string = names(auc_xvalidacion),
                                          pattern = "[(,]",
                                          replacement = "_")
```

```
# Se convierte la lista en dataframe.
auc_xvalidacion_df <- as.data.frame(auc_xvalidacion) %>%
                      mutate(resample = row_number()) %>%
                      gather(key = "modelo", value = "auc", -resample) %>%
                      mutate(modelo = str_replace_all(string = modelo,
                                                      pattern = "_" ,
                                                      replacement = " \n "))
# Gráfico
ggplot(data = auc_xvalidacion_df, aes(x = modelo, y = auc, color = modelo)) +
  geom_boxplot(alpha = 0.6, outlier.shape = NA) +
  geom_jitter(width = 0.1, alpha = 0.6) +
  stat_summary(fun.y = "mean", colour = "red", size = 2, geom = "point") +
  theme_bw() +
  labs(title = "Accuracy obtenido por 10-CV",
       subtitle = "Modelos ordenados por media") +
  coord_flip() +
  theme(legend.position = "none")
```



En este caso, los resultados de los 5 modelos son prácticamente idénticos.

Una vez identificado el mejor modelo, mediante h2o.grid(), se extrae del objeto grid y se almacena por separado.

```
# Se extrae el mejor modelo, que es el que ocupa la primera posición tras haber
# ordenado los resultados de mayor a menor AUC (métrica utilizada en este caso).
modelo_glm_final <- h2o.getModel(resultados_grid@model_ids[[1]])
```

Gradient Boosting Machin (GBM)

Introducción

Boosting Machine (BM) es un tipo de modelo obtenido al combinar (*ensembling*) múltiples modelos sencillos (de regresión o clasificación) también conocidos como *weak learners*. Está combinación se realiza de forma secuencial, de forma que cada nuevo modelo que se incorpora al conjunto intenta corregir los errores de los anteriores modelos. Como resultado de la combinación de múltiples modelos, *Boosting Machine* consigue aprender relaciones no lineales entre la variable respuesta y los predictores. Si bien los modelos combinados pueden ser muy variados, H2O, al igual que la mayoría de librerías, utiliza como *weak learners* modelos de árboles simples.

Gradient Boosting Machine (GBM) es una generalización del modelo de *Boosting Machine* que permite aplicar el método de descenso de gradiente para optimizar cualquier función de coste durante el ajuste del modelo.

El valor predicho por un modelo GBM es la agregación (normalmente la moda en problemas de clasificación y la media en problemas de regresión) de las predicciones de todos los modelos individuales que forman el *ensemble*.

Dado que el ajuste de los *weak learners* que forman un GBM se hace de forma secuencial (el árbol *i* se ajusta a partir de los resultados del árbol *i-1)*, las posibilidades de paralelizar el algoritmo son limitadas. H2O consigue acelerar el proceso en cierta medida paralelizando el ajuste de los árboles individuales.

Para información más detallada sobre métodos *boosting* consultar Árboles de predicción .

Detención temprana del entrenamiento (Early Stopping)

Una de las características de GBM es que, con el número suficiente de *weak learners*, el modelo final tiende a ajustarse perfectamente a los datos de entrenamiento causando *overfitting*. Este comportamiento implica que el analista tiene que encontrar el número adecuado de árboles y, para ello, suele tener que entrenar el modelo con cientos o miles de árboles hasta identificar el momento en el que empieza el *overfitting*. Esto suele ser poco eficiente en términos de tiempo, ya que, posiblemente, se estén ajustando muchos árboles innecesarios.

De nuevo, la experiencia de los creadores de H2O queda reflejada en su herramienta, esta vez incluyendo toda una serie de estrategias para detener el proceso de ajuste de un modelo GBM a

partir del momento en el que este deja de mejorar. Por defecto, la métrica empleada se calcula con los datos de validación. Cuatro argumentos controlan la estrategia de parada:

- `stopping_metric`: métrica empleada para cuantificar cuánto mejora el modelo.
- `score_tree_interval`: número de árboles tras los que se evalúa el modelo. Por ejemplo, si el valor es 10, entonces, tras cada 10 nuevos árboles que se añaden al modelo, se calcula la `stopping_metric`. Si bien la evaluación del modelo puede hacerse tras cada nuevo árbol que se añade, esto puede ralentizar el entrenamiento.
- `stopping_tolerance`: porcentaje mínimo de mejora entre dos mediciones consecutivas (`score_tree_interval`) por debajo del cual se considera que el modelo no ha mejorado.
- `stopping_rounds`: número de mediciones consecutivas en las que no se debe superar el `stopping_tolerance` para que el algoritmo se detenga.

Algunos ejemplos de criterios de parada son:

- Detener el entrenamiento de un modelo cuando el error de clasificación no se reduce más de un 1% durante dos mediciones consecutivas: `stopping_rounds=1`, `stopping_tolerance=0.01` y `stopping_metric="misclassification"`.
- Detener el entrenamiento de un modelo cuando el *logloss* no se reduce nada durante 3 mediciones consecutivas: `stopping_rounds=3`, `stopping_tolerance=0` y `stopping_metric="logloss"`.
- Detener el entrenamiento de un modelo cuando la media de *AUC* no aumenta más de un 0.1% durante cinco mediciones consecutivas: `stopping_rounds=5`, `stopping_tolerance=0.001` y `stopping_metric="AUC"`.
- Para detener el modelo aunque la métrica converja, se tiene que deshabilitar este comportamiento de parada automática indicando `stopping_rounds=0`. En este caso, el algoritmo se detendrá cuando se alcance otra condición de parada, por ejemplo, el número máximo de árboles que pueden formar el modelo.

También es posible detener el entrenamiento del modelo cuando se supera un tiempo máximo especificado con el argumento `max_runtimesecs > 0`.

En la práctica, es conveniente establecer un número máximo de árboles muy elevado (potencialmente por encima de la cantidad necesaria) y activar la parada temprana.

Comportamiento estocástico

Stochastic Gradient Boosting es el resultado de emplear, en el entrenamiento de cada árbol, únicamente una selección aleatoria de observaciones y predictores del conjunto de entrenamiento. Esta modificación suele reducir el *overfitting* del modelo y mejorar su generalización. Se puede especificar el % de datos aleatorios seleccionados en cada ajuste con los argumentos:

- sample_rate: porcentaje de observaciones aleatorias empleadas para el ajuste de cada árbol del *ensemble*.
- sample_rate_per_class: cuando el set de datos con el que se trabaja está desbalanceado (el % de observaciones de cada grupo es muy distinto), al hacer una selección aleatoria de observaciones para el entrenamiento, puede ocurrir que, en algunos árboles, la selección excluya a alguno de los grupos. Para evitarlo, con el argumento sample_rate_per_class se indica que la selección debe hacerse de forma estratificada, asegurando que todos los grupos queden representados.
- col_sample_rate: porcentaje de predictores (columnas) seleccionados de forma aleatoria para el ajuste de cada árbol del *ensemble*.

Los tres parámetros deben tener un valor entre 0 y 1. Por defecto, el valor empleado es 1, es decir, se emplean todos los datos y no hay comportamiento estocástico.

Distribución y función de coste

Dependiendo el tipo de modelo (regresión, clasificación binaria, clasificación múltiple…) se emplea una distribución y una función de coste distinta. Indicando la distribución con el argumento distribution, H2O selecciona automáticamente la función de coste correspondiente.

Las distribuciones disponibles son: *bernoulli*, *quasibinomial*, *multinomial*, *poisson*, *laplace*, *tweedie*, *gaussian*, *huber*, *gamma* y *quantile*. De entre todas ellas, las más empleadas suelen ser: *gaussian* para regresión, *bernoulli* para clasificación binaria y *multinomial* para clasificación multiclase.

Otros hiperparámetros

Además de los ya descritos (criterio de parada, comportamiento estocástico, distribución), el modelo GBM de H2O incorpora muchos otros parámetros e hiperparámetros con los que se puede controlar el comportamiento del algoritmo. A continuación se describen algunos de los que tienen más influencia en el modelo final. Se puede encontrar un listado completo en la documentación.

Complejidad de los árboles

- ntrees: número de árboles que forman el *ensemble*.
- max_depth: profundidad máxima que pueden tener los árboles (número máximo de nodos).
- min_rows: número mínimo de observaciones que debe tener cada nodo.
- min_split_improvement: reducción mínima en el error cuadrático que debe de conseguir la división de un nodo para que se lleve a cabo.

Velocidad de aprendizaje

- learn_rate: determina cuanto influye cada árbol individual en el conjunto del *ensemble*. Cuanto más pequeño es el valor de *learning rate*, más lentamente aprende el modelo y más árboles se necesitan, sin embargo, al aprender poco a poco, se reduce el riesgo de *overfitting*. Suelen emplearse valores inferiores a 0.1.
- learn_rate_annealing: emplear un valor de learn_rate muy pequeño puede conllevar que el modelo necesite varios miles de árboles y, por lo tanto, el entrenamiento tarde mucho tiempo. En lugar de utilizar un valor muy bajo desde el principio, se puede iniciar el proceso con un valor moderadamente alto e ir reduciéndolo tras cada nuevo árbol que se incorpora al modelo. El learn_rate_annealing determina el factor por el que se reduce el learn_rate en cada iteración. Para *N* árboles, GBM empieza con learn_rate y termina con learn_rate * learn_rate_annealing^N. Por ejemplo, en lugar de emplear earn_rate=0.01, se puede emplear un learn_rate=0.05 y un learn_rate_annealing=0.99. Con la segunda opción, el algoritmo converge mucho más rápido con aproximadamente la misma capacidad predictiva final.

Para conseguir maximizar la capacidad predictiva de un modelo *GBM*, es imprescindible entender bien cómo influye cada uno de los parámetros e hiperparámetros. Algunos de ellos pueden encontrarse descritos con más detalle en el documento Boosting.

Modelo

A continuación, se ajusta un modelo *GBM* que prediga la variable respuesta *salario*. En esta ocasión, se emplea validación simple en lugar de validación cruzada, por lo que dividen de nuevo los datos, esta vez, en un conjunto de entrenamiento (60%), validación (20%) y test (20%).

```
particiones <- h2o.splitFrame(data = datos_h2o, ratios = c(0.6, 0.20), seed = 123)
datos_train_h20 <- h2o.assign(data = particiones[[1]], key = "datos_train_H2O")
datos_val_h20   <- h2o.assign(data = particiones[[2]], key = "datos_val_H2O")
datos_test_h20  <- h2o.assign(data = particiones[[3]], key = "datos_test_H2O")
```

Se entrena un modelo inicial en el que únicamente se especifican el número máximo de árboles, la complejidad de los mismos y la estrategia de parada temprana, el resto de valores se dejan por defecto.

```
modelo_gbm <- h2o.gbm(
                      # Tipo de distribución (clasificación binaria)
                      distribution = "bernoulli",
                      # Variable respuesta y predictores.
                      y = var_respuesta,
                      x = predictores,
                      # Datos de entrenamiento.
                      training_frame = datos_train_h20,
                      # Datos de validación para estimar el error.
                      validation_frame = datos_val_h20,
                      # Número de árboles.
                      ntrees = 500,
                      # Complejidad de los árboles
                      max_depth = 3,
                      min_rows = 10,
                      # Aprendizaje
                      learn_rate = 0.01,
                      # Detención temprana
                      score_tree_interval = 5,
                      stopping_rounds = 3,
                      stopping_metric = "AUC",
                      stopping_tolerance = 0.001,
                      model_id = "modelo_gbm",
                      seed = 123)

modelo_gbm
```

```
## Model Details:
## ==============
##
## H2OBinomialModel: gbm
## Model ID:  modelo_gbm
## Model Summary:
##   number_of_trees number_of_internal_trees model_size_in_bytes min_depth
## 1              95                       95               15471         3
##   max_depth mean_depth min_leaves max_leaves mean_leaves
## 1         3    3.00000          8          8     8.00000
##
##
## H2OBinomialMetrics: gbm
## ** Reported on training data. **
##
## MSE:  0.1244416
## RMSE:  0.3527628
## LogLoss:  0.4020413
## Mean Per-Class Error:  0.2194013
## AUC:  0.8896691
## Gini:  0.7793381
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##        <=50K >50K    Error         Rate
## <=50K  18247 2157 0.105715  =2157/20404
## >50K    2263 4531 0.333088   =2263/6794
## Totals 20510 6688 0.162512  =4420/27198
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                        max f1  0.340967 0.672155 117
## 2                        max f2  0.151212 0.773090 175
## 3                  max f0point5  0.469619 0.711687  60
## 4                  max accuracy  0.380419 0.846717  85
## 5                 max precision  0.683568 0.995885   1
## 6                    max recall  0.118008 1.000000 181
## 7               max specificity  0.700388 0.999951   0
## 8              max absolute_mcc  0.367128 0.566731  93
## 9   max min_per_class_accuracy  0.263324 0.784074 156
## 10 max mean_per_class_accuracy  0.278211 0.798695 149
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLif
t(<model>, valid=<T/F>, xval=<T/F>)`
## H2OBinomialMetrics: gbm
## ** Reported on validation data. **
##
## MSE:  0.1206073
## RMSE:  0.3472855
## LogLoss:  0.3929575
## Mean Per-Class Error:  0.2051175
```

```
## AUC:  0.898275
## Gini:  0.7965499
##
## Confusion Matrix (vertical: actual; across: predicted) for F1-optimal threshold:
##        <=50K >50K    Error        Rate
## <=50K   6105  720 0.105495   =720/6825
## >50K     675 1540 0.304740   =675/2215
## Totals  6780 2260 0.154314  =1395/9040
##
## Maximum Metrics: Maximum metrics at their respective thresholds
##                         metric threshold    value idx
## 1                       max f1  0.339734 0.688268  89
## 2                       max f2  0.157774 0.773203 136
## 3                 max f0point5  0.379802 0.727403  64
## 4                 max accuracy  0.379802 0.857190  64
## 5                max precision  0.700388 1.000000   0
## 6                   max recall  0.118008 1.000000 143
## 7              max specificity  0.700388 1.000000   0
## 8             max absolute_mcc  0.379802 0.589602  64
## 9   max min_per_class_accuracy  0.265986 0.801806 119
## 10 max mean_per_class_accuracy  0.281448 0.809153 114
##
## Gains/Lift Table: Extract with `h2o.gainsLift(<model>, <data>)` or `h2o.gainsLif
t(<model>, valid=<T/F>, xval=<T/F>)`
```

Evolución del modelo

Tras ajustar un modelo GBM, es fundamental evaluar la como aprende el modelo a medida que se añaden nuevos árboles al *ensemble*. Una forma de hacerlo es representar la evolución de una métrica (en este caso el área bajo la curva roc, *AUC*) de entrenamiento y validación.

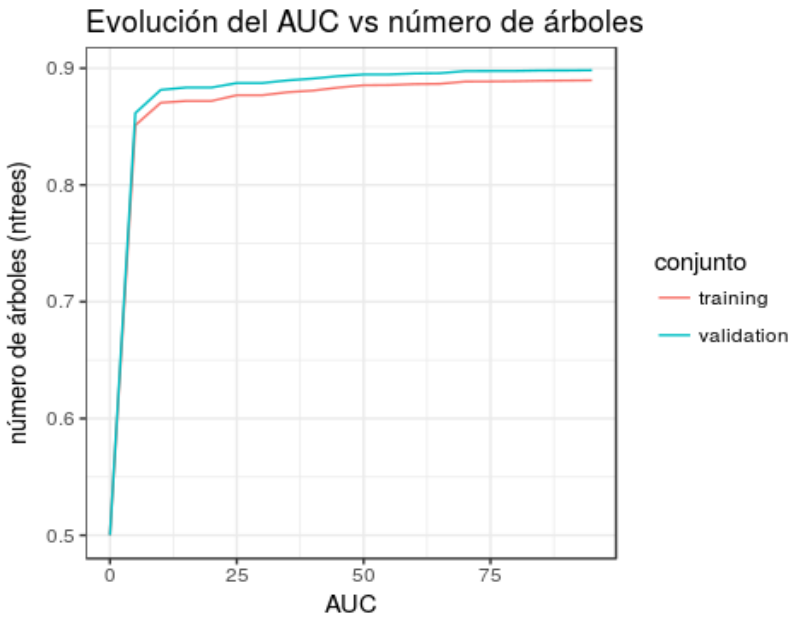
```
# H2O almacena las métricas de entrenamiento y test bajo el nombre de scoring.
# Los valores se encuentran almacenados dentro del modelo.
scoring <- as.data.frame(modelo_gbm@model$scoring_history)
head(scoring)
```

```
##             timestamp   duration number_of_trees training_rmse
## 1 2018-07-21 19:50:24  0.020 sec               0     0.4328959
## 2 2018-07-21 19:50:24  0.112 sec               5     0.4250790
## 3 2018-07-21 19:50:24  0.195 sec              10     0.4178684
## 4 2018-07-21 19:50:24  0.282 sec              15     0.4111956
## 5 2018-07-21 19:50:24  0.373 sec              20     0.4050231
## 6 2018-07-21 19:50:25  0.459 sec              25     0.3993307
##   training_logloss training_auc training_lift
## 1        0.5621129    0.5000000      1.000000
## 2        0.5445709    0.8511884      3.996064
```

```
## 3        0.5291262    0.8704359      3.950908
## 4        0.5153104    0.8719763      3.959439
## 5        0.5028586    0.8719872      3.959439
## 6        0.4916276    0.8769029      3.959439
##   training_classification_error validation_rmse validation_logloss
## 1                     0.7502022       0.4301268          0.5568611
## 2                     0.2759026       0.4220834          0.5389323
## 3                     0.1729539       0.4146631          0.5231557
## 4                     0.1793147       0.4077691          0.5089963
## 5                     0.1793147       0.4013842          0.4962254
## 6                     0.1793147       0.3954994          0.4847174
##   validation_auc validation_lift validation_classification_error
## 1      0.5000000        1.000000                       0.7549779
## 2      0.8615840        4.061643                       0.1536504
## 3      0.8814851        4.006379                       0.1659292
## 4      0.8834946        3.975941                       0.1715708
## 5      0.8835067        3.975941                       0.1715708
## 6      0.8873262        3.975941                       0.1715708
```

Tal y como se ha indicado en el argumento (`score_tree_interval=5`), cada 5 árboles se han recalculado las métricas de rendimiento en el conjunto de entrenamiento y validación. El argumento `stopping_metric="AUC"` indica que se emplee el área bajo la curva como criterio para decidir si el algoritmo debe detenerse de forma temprana, pero se calculan siempre todas (*AUC*, *logloss*, *lift*, *rmse*, *classification_error*).

```
scoring <- scoring %>%
           select(-timestamp, -duration) %>%
           gather(key = "metrica", value = "valor", -number_of_trees) %>%
           separate(col = metrica, into = c("conjunto", "metrica"), sep = "_")


scoring %>% filter(metrica == "auc") %>%
  ggplot(aes(x = number_of_trees, y = valor, color = conjunto)) +
  geom_line() +
  labs(x = "AUC", y = "número de árboles (ntrees)",
           title = "Evolución del AUC vs número de árboles") +
  theme_bw()
```




```
# Equivalente
# plot(x = modelo_gbm, metric = "auc")
```

En el gráfico puede observarse que no se han alcanzado los 500 árboles indicados en el modelo, esto es así porque, alcanzados los 95 árboles, el valor de *AUC* se ha estabilizado lo suficiente como para que se ejecute la detención temprana.

Importancia de los predictores

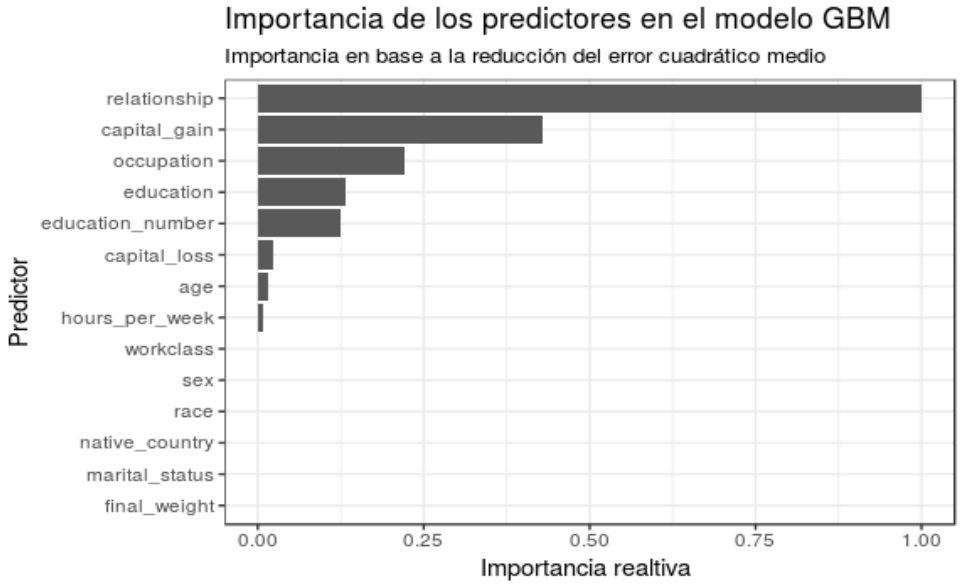
En los modelos GBM de H2O, se puede estudiar la influencia de los predictores cuantificando la reducción total de error cuadrático que ha conseguido cada predictor en el conjunto de todos los árboles que forman el modelo.

```
# Se extraen los valores de importancia
importancia <- as.data.frame(modelo_gbm@model$variable_importances)
importancia
```

```
##            variable relative_importance scaled_importance  percentage
## 1      relationship          43854.7891       1.000000000 0.511678326
## 2      capital_gain          18832.6973       0.429433083 0.219731601
## 3        occupation           9717.4492       0.221582396 0.113378909
## 4         education           5749.3257       0.131099153 0.067080595
## 5  education_number           5419.2427       0.123572426 0.063229332
## 6      capital_loss           1062.5126       0.024227971 0.012396928
```

```
## 7              age          682.1815       0.015555461 0.007959392
## 8    hours_per_week          389.5345       0.008882371 0.004544917
## 9        workclass            0.0000       0.000000000 0.000000000
## 10     final_weight           0.0000       0.000000000 0.000000000
## 11   marital_status           0.0000       0.000000000 0.000000000
## 12             race           0.0000       0.000000000 0.000000000
## 13              sex           0.0000       0.000000000 0.000000000
## 14   native_country           0.0000       0.000000000 0.000000000
```

```
ggplot(data = importancia,
       aes(x = reorder(variable, scaled_importance),
           y = scaled_importance)) +
  geom_col() +
  coord_flip() +
  labs(title = "Importancia de los predictores en el modelo GBM",
       subtitle = "Importancia en base a la reducción del error cuadrático medio",
       x = "Predictor",
       y = "Importancia relativa") +
  theme_bw()
```




```
# Equivalente:
# h2o.varimp_plot(modelo_gbm)
```

Predicciones y error

Una vez ajustado el modelo, se pueden predecir nuevas observaciones y estimar el error promedio.

```
predicciones <- h2o.predict(object = modelo_gbm, newdata = datos_test_h2o)

predicciones
```

```
##   predict     <=50K      >50K
## 1    >50K 0.2996122 0.7003878
## 2    >50K 0.3634380 0.6365620
## 3   <=50K 0.7586597 0.2413403
## 4   <=50K 0.7398186 0.2601814
## 5   <=50K 0.7586597 0.2413403
## 6    >50K 0.5307913 0.4692087
##
## [8984 rows x 3 columns]
```

```
# AUC de test
h2o.performance(model = modelo_gbm, newdata = datos_test_h2o)@metrics$AUC
```

```
## [1] 0.8928113
```

Grid search

Empleando los valores por defecto de los hiperparámetros, se ha conseguido un error de test de 0.893. A continuación, se intenta encontrar los valores óptimos de los hiperparámetros con los que se consiga mejorar aún más el *AUC*.

Dada la gran cantidad de hiperparámetros que tiene el algoritmo *GBM* y, aunque todos interaccionan entre sí, no es posible explorar todo el espacio de posibles valores a la vez. Una estrategia que suele funcionar bien es establecer primero los hiperparámetros más influyentes (número de árboles, *learning rate*, complejidad de los árboles, % de observaciones empleadas en cada ajuste) y luego el resto.

- Número de árboles (`ntree`): incorporar tantos árboles como sea necesario hasta que el error de validación se estabilice o empiece a aumentar. Dado que H2O incorpora estrategias de parada temprana, no hay problema en indicar un número elevado porque, de no ser necesario, no se ajustarán.

- Tasa de aprendizaje (`learning_rate`): por lo general, valores pequeños (<0.01) resultan en mejores modelos, pero requieren más árboles para aprender. Se recomienda reducir el valor tanto como permitan las limitaciones de tiempo/computación.
- Profundidad de los árboles (`max_depth`): es la principal forma de controlar la complejidad de los árboles que forman el *ensemble*. Al tratarse de *weak learners*, la complejidad de los árboles debe ser baja, aunque depende mucho de los datos, el valor óptimo raramente está fuera del rango [1, 20].
- Comportamiento estocástico (`sample_rate`, `col_sample_rate`): la finalidad de emplear solo un subconjunto de observaciones y/o columnas en el entrenamiento de cada árbol es conseguir mejor generalización, sin embargo, puede tener el resultado contrario cuando se dispone de muy pocos datos. El valor óptimo suele estar dentro del rango de [0.7-0.9]. En situaciones en los que la variable respuesta es cualitativa y los grupos están desbalanceados, es conveniente realizar una selección aleatoria estratificada, este comportamiento puede activarse con `sample_rate_per_class`.

```
# Hiperparámetros que se quieren comparar.
hiperparametros <- list(learn_rate  = c(0.001, 0.05, 0.1),
                        max_depth   = c(5, 10, 15, 20),
                        sample_rate = c(0.8, 1))

grid_gbm <- h2o.grid(
    # Algoritmo.
    algorithm = "gbm",
    distribution = "bernoulli",
    # Variable respuesta y predictores.
    y = var_respuesta,
    x = predictores,
    # Datos de entrenamiento.
    training_frame = datos_train_h2o,
    # Datos de validación.
    validation_frame = datos_val_h20,
    # Preprocesado.
    ignore_const_cols = TRUE,
    # Detención temprana.
    score_tree_interval = 10,
    stopping_rounds = 3,
    stopping_metric = "AUC",
    stopping_tolerance = 0.001,
    # Hiperparámetros fijados.
    ntrees = 5000,
    min_rows = 10,
    # Hiperparámetros optimizados.
    hyper_params = hiperparametros,
```

```
    # Tipo de búsqueda.
    search_criteria = list(strategy = "Cartesian"),
    seed = 123,
    grid_id = "grid_gbm")

# Se muestran los modelos ordenados de mayor a menor AUC.
resultados_grid <- h2o.getGrid(grid_id = "grid_gbm",
                               sort_by = "auc",
                               decreasing = TRUE)

data.frame(resultados_grid@summary_table) %>% select(-model_ids)
```

```
##    learn_rate max_depth sample_rate               auc
## 1         0.1        20         0.8 0.9999934181694905
## 2         0.1        20         1.0 0.9999675208162793
## 3         0.1        15         0.8 0.9999339501732278
## 4        0.05        20         0.8 0.9998481217804017
## 5        0.05        20         1.0 0.9997803190037953
## 6        0.05        15         0.8 0.9996054870637263
## 7         0.1        15         1.0  0.999129610795525
## 8        0.05        15         1.0 0.9977724307295414
## 9         0.1        10         0.8 0.9846072482821918
## 10       0.05        10         0.8 0.9839516119696706
## 11        0.1        10         1.0 0.9788428215877426
## 12       0.05        10         1.0 0.9753048065553709
## 13      0.001        20         0.8 0.9628418293519874
## 14      0.001        20         1.0 0.9541996874457371
## 15      0.001        15         0.8 0.9538756893971341
## 16        0.1         5         0.8 0.9516128626828402
## 17        0.1         5         1.0 0.9506168233572296
## 18       0.05         5         0.8  0.949382349779641
## 19       0.05         5         1.0 0.9491593944054441
## 20      0.001        15         1.0 0.9466584972589488
## 21      0.001        10         0.8 0.9338122524578507
## 22      0.001        10         1.0 0.9296325916371063
## 23      0.001         5         0.8 0.9070215563217822
## 24      0.001         5         1.0 0.9015677986422908
```

Los 3 modelos con mayor *AUC* de validación tienen todos un *learning rate* de 0.1 y un *max depth* de 15 y 20, el *sample rate* parece no influir de forma sustancial. Teniendo en cuenta estos resultados, pude concluirse que los valores óptimos son: `learn_rate=0.1`, `max_depth=20` y `sample_rate=0.8`.

```
modelo_gbm_final <- h2o.getModel(resultados_grid@model_ids[[1]])

# AUC de test
h2o.performance(model = modelo_gbm_final, newdata = datos_test_h2o)@metrics$AUC
```

```
## [1] 0.9140022
```

Optimizando los hiperparámetros más influyentes, se ha conseguido mejorar el *AUC* de test de 0.893 a 0.914. Por último, se intenta encontrar el valor óptimo del resto de hiperparámetros, esta vez mediante una búsqueda aleatoria.

```
# Hiperparámetros que se quieren optimizar mediante búsqueda aleatoria.
# Para realizar esta búsqueda se tiene que pasar un vector de posibles valores
# de cada hiperparámetro, entre los que se escoge aleatoriamente.
hiperparametros <- list(
  min_rows = seq(from = 5, to = 50, by = 10),
  nbins = 2^seq(4, 10, 1),
  nbins_cats = 2^seq(4, 12, 1),
  min_split_improvement = c(0, 1e-8, 1e-6, 1e-4),
  histogram_type = c("UniformAdaptive","QuantilesGlobal","RoundRobin")
)

# Al ser una búsqueda aleatoria, hay que indicar criterios de parada.
search_criteria <- list(
  strategy = "RandomDiscrete",
  # Tiempo máximo de búsqueda (5 minutos).
  max_runtime_secs = 5*60,
  # Número máximo de modelos.
  max_models = 100,
  # Reproducible.
  seed = 1234
)

grid_gbm2 <- h2o.grid(
    # Algoritmo.
    algorithm = "gbm",
    distribution = "bernoulli",
    # Variable respuesta y predictores.
    y = var_respuesta,
    x = predictores,
    # Datos de entrenamiento.
    training_frame = datos_train_h2o,
    # Datos de validación.
    validation_frame = datos_val_h20,
    # Preprocesado.
    ignore_const_cols = TRUE,
    # Detención temprana
    score_tree_interval = 10,
    stopping_rounds = 3,
    stopping_metric = "AUC",
    stopping_tolerance = 0.001,
    # Hiperparámetros fijados
    ntrees = 5000,
    learn_rate = 0.1,
    max_depth = 20,
    sample_rate = 0.8,
```

```
    # Hiperparámetros optimizados.
    hyper_params = hiperparametros,
    # Tipo de búsqueda.
    search_criteria = search_criteria,
    seed = 123,
    grid_id = "grid_gbm2")

# Se muestran los modelos ordenados de mayor a menor AUC.
resultados_grid <- h2o.getGrid(grid_id = "grid_gbm2",
                               sort_by = "auc",
                               decreasing = TRUE)
data.frame(resultados_grid@summary_table) %>% select(-model_ids)
```

```
##     histogram_type min_rows min_split_improvement nbins nbins_cats
## 1  QuantilesGlobal      5.0                1.0E-8   512         16
## 2       RoundRobin      5.0                1.0E-8    16         32
## 3       RoundRobin     15.0                   0.0   512       4096
## 4  QuantilesGlobal     15.0                1.0E-6  1024       2048
## 5       RoundRobin     25.0                   0.0    16       4096
## 6  UniformAdaptive     35.0                   0.0   512        256
## 7  UniformAdaptive      5.0                1.0E-4    32        512
## 8  UniformAdaptive     45.0                1.0E-8   512        512
## 9  UniformAdaptive     45.0                1.0E-6   128        512
## 10 QuantilesGlobal     45.0                   0.0    16       1024
## 11 UniformAdaptive     35.0                1.0E-4   512        256
## 12 QuantilesGlobal     15.0                1.0E-4   128       1024
## 13 QuantilesGlobal     35.0                1.0E-4   128        512
##                   auc
## 1  0.9999996692547483
## 2  0.9999994377330721
## 3  0.9999829335450103
## 4  0.9999767486088028
## 5  0.9998937976996668
## 6  0.9998057863881791
## 7   0.999703586105392
## 8  0.9996343611242031
## 9   0.999565103068489
## 10 0.9993649691166622
## 11 0.9758039011402442
## 12 0.9727742084852694
## 13 0.9279392420972556
```

```
modelo_gbm_final <- h2o.getModel(resultados_grid@model_ids[[1]])


# AUC de test
h2o.performance(model = modelo_gbm_final, newdata = datos_test_h2o)@metrics$AUC
```

```
## [1] 0.9147773
```

El *AUC* obtenido es prácticamente idéntico al del anterior modelo.

Deep Learning (Neural Networks)

Introducción

El término *deep learning* engloba a todo un conjunto de modelos basados en redes neuronales artificiales (*artificial neural networks*) que contienen múltiples capas intermedias (ocultas). En concreto, H2O incorpora redes neuronales de tipo *Multi-layer - feedforward - neural networks*. Estas redes se caracterizan por:

- Pueden tener una o múltiples capas intermedias, también conocidas como capas ocultas. Son las capas de neuronas que hay entre la capa de entrada y la de salida.
- La estructura es *full conected*, lo que significa que cada neurona está conectada con todas las neuronas de la capa siguiente.
- El peso (participación de cada neurona en la red) se aprende mediante el proceso *feed-forward*. A grandes rasgos, la estrategia de aprendizaje consiste en un proceso iterativo. En cada iteración (época), la red predice las observaciones de entrenamiento, se calcula el error de estas predicciones comparando los resultados frente al valor real y se reajustan los pesos de las neuronas con el objetivo de minimizar el error (*backpropagation* y *descenso de gradiente*).

Estas redes son unas de las más empleadas, pero existen otros tipos, por ejemplo, las *Convolutional Neural Networks (CNNs)* son las que mejores resultados dan cuando se trabaja con imágenes.

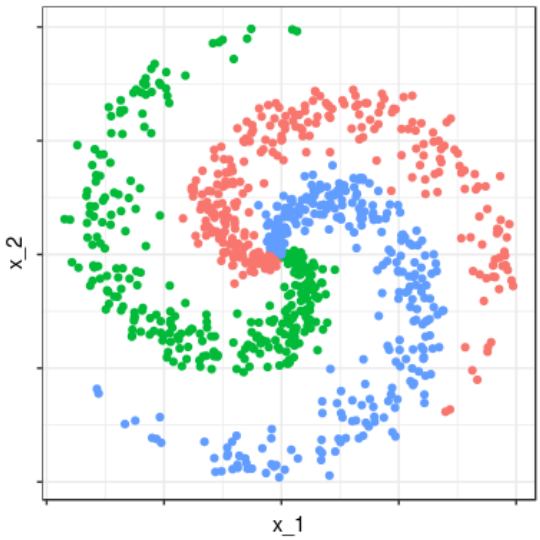
Comprender los fundamentos de las redes neuronales y *deep learning* requiere de un estudio notable, puede encontrarse mucha de la información necesaria en el curso gratuito de Stanford Convolutional Neural Networks for Visual Recognition y en el libro Deep Learning An MIT Press book.

Flexibilidad de las redes neuronales

Los modelos basados en redes neuronales tienen la característica de ser potencialmente capaces de aprender cualquier función. La velocidad con la que aprenden está influenciada principalmente (aunque hay muchos otros hiperparámetros) por el número de observaciones, el número de neuronas y el número de iteraciones de aprendizaje (comúnmente llamadas *epoch*). A continuación, se muestra un ejemplo sencillo que ilustra este concepto.

Supóngase las siguientes observaciones generadas a partir de un modelo no lineal de espiral.

```
# Número de observaciones por clase
N = 300
# Número de dimensiones
D = 2
# Número clases
K = 3
# Matriz para almacenar las observaciones
x_1 = vector(mode = "numeric")
x_2 = vector(mode = "numeric")
y   = vector(mode = "numeric")

# Simulación de los datos
for(i in 1:K){
  set.seed(123)
  r = seq(from = 0, to = 1, length.out = N)
  t = seq(from =  i*4, to = (i+1)*4, length.out = N) + rnorm(n = N) * 0.35
  x_1 <- c(x_1, r * sin(t))
  x_2 <- c(x_2, r*cos(t))
  y   <- c(y, rep(letters[i], N))
}
datos_espiral <- data.frame(y, x_1, x_2)
ggplot(data = datos_espiral, aes(x = x_1, y = x_2, color = y)) +
  geom_point() +
  theme_bw() +
  theme(legend.position = "none",
        axis.text = element_blank())
```

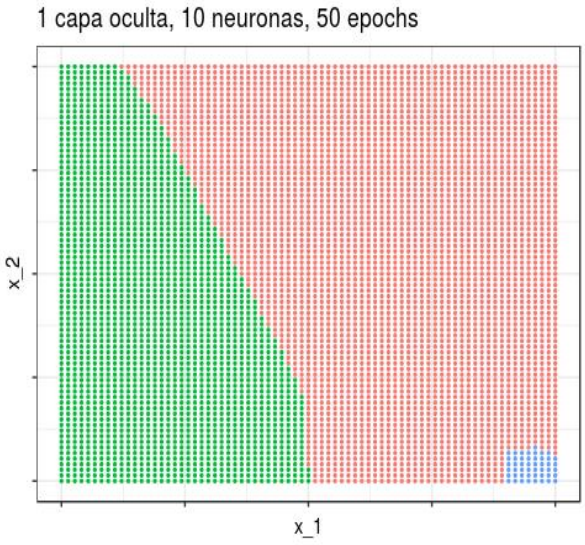


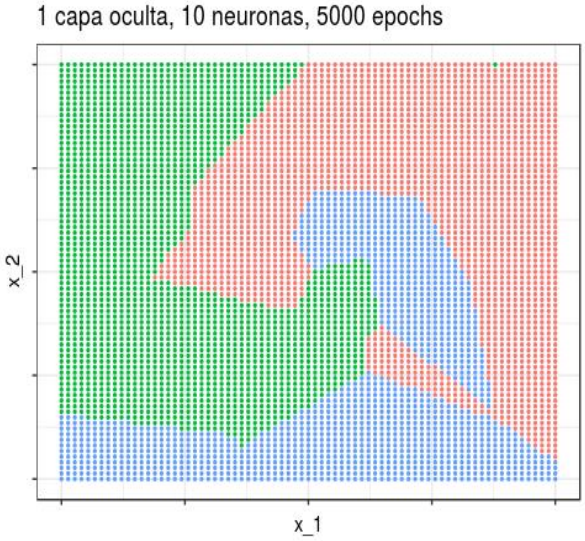
Se comparan 3 redes neuronales: 10 neuronas y 50 *epochs*, 10 neuronas y 5000 *epochs*, 100 neuronas y 50 *epochs*.

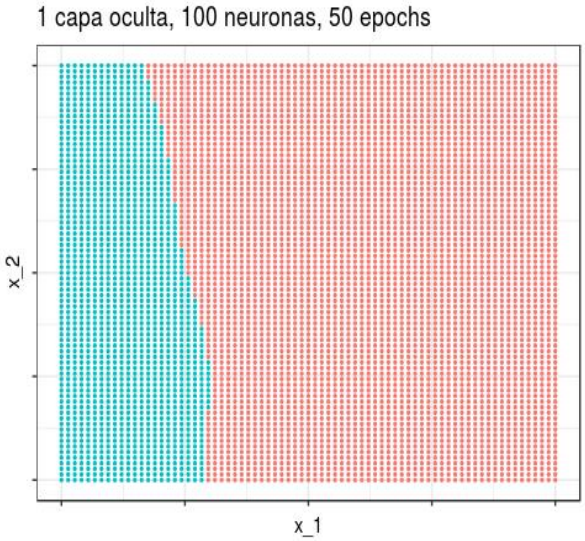
```
datos_espiral_h2o <- as.h2o(datos_espiral)

modelo_dl_10 <- h2o.deeplearning(x = c("x_1", "x_2"),
                                 y = "y",
                                 distribution = "multinomial",
                                 training_frame = datos_espiral_h2o,
                                 standardize = TRUE,
                                 activation = "Rectifier",
                                 hidden = 10,
                                 stopping_rounds = 0,
                                 epochs = 50,
                                 seed = 123,
                                 model_id = "modelo_dl_10"
                                 )

modelo_dl_10_5k <- h2o.deeplearning(x = c("x_1", "x_2"),
                                 y = "y",
                                 distribution = "multinomial",
                                 training_frame = datos_espiral_h2o,
                                 standardize = TRUE,
                                 activation = "Rectifier",
                                 hidden = 10,
                                  stopping_rounds = 0,
                                 epochs = 5000,
                                 seed = 123,
                                 model_id = "modelo_dl_10_5k"
                                 )

modelo_dl_100 <- h2o.deeplearning(x = c("x_1", "x_2"),
                                 y = "y",
                                 distribution = "multinomial",
                                 training_frame = datos_espiral_h2o,
                                 standardize = TRUE,
                                 activation = "Rectifier",
                                 hidden = 100,
                                  stopping_rounds = 0,
                                 epochs = 100,
                                 seed = 123,
                                 model_id = "modelo_dl_100"
                                 )
```

Una vez entrenados los modelos, se representan sus fronteras de clasificación.

```
grid_predicciones <- expand.grid(x_1 = seq(from = -1, to = 1, length = 75),
                                 x_2 = seq(from = -1, to = 1, length = 75))
grid_predicciones_h2o <- as.h2o(grid_predicciones)

predicciones_10 <- h2o.predict(object = modelo_dl_10,
                               newdata = grid_predicciones_h2o)
grid_predicciones$y_10 <- as.vector(predicciones_10$predict)

predicciones_10_5k <- h2o.predict(object = modelo_dl_10_5k,
                               newdata = grid_predicciones_h2o)
grid_predicciones$y_10_5k <- as.vector(predicciones_10_5k$predict)

predicciones_100 <- h2o.predict(object = modelo_dl_100,
                                newdata = grid_predicciones_h2o)
grid_predicciones$y_100 <- as.vector(predicciones_100$predict)

ggplot(data = grid_predicciones, aes(x = x_1, y = x_2, color = y_10)) +
  geom_point(size = 0.5) +
  theme_bw() +
  labs(title = "1 capa oculta, 10 neuronas, 50 epochs") +
  theme(legend.position = "none",
        axis.text = element_blank())

ggplot(data = grid_predicciones, aes(x = x_1, y = x_2, color = y_10_5k)) +
  geom_point(size = 0.5) +
  labs(title = "1 capa oculta, 10 neuronas, 5000 epochs") +
  theme_bw() +
  theme(legend.position = "none",
        axis.text = element_blank())

ggplot(data = grid_predicciones, aes(x = x_1, y = x_2, color = y_100)) +
  geom_point(size = 0.5) +
  labs(title = "1 capa oculta, 100 neuronas, 50 epochs") +
  theme_bw() +
  theme(legend.position = "none",
        axis.text = element_blank())
```


1 capa oculta, 10 neuronas, 50 epochs
x_2
x_1


1 capa oculta, 10 neuronas, 5000 epochs
x_2
x_1


1 capa oculta, 100 neuronas, 50 epochs
x_2
x_1

Este ejemplo debe servir como reflexión sobre la alta capacidad que tienen los modelos basados en redes para adaptarse a los datos de entrenamiento, por lo tanto, cuando la complejidad de los datos no es muy grande o se dispone de pocas observaciones, se debe tener muy presente el riesgo de *overfitting*.

Parámetros e hiperparámetros

Los modelos de *Deep Learning* ofrecidos por H2O tienen un número muy elevado de parámetros configurables (listado). Para la gran mayoría de casos, los valores por defecto dan buenos resultados, sin embargo, es conveniente conocer, al menos, los más influyentes.

Arquitectura

- `hidden`: número de capas ocultas y número de neuronas por capa. La estructura se define mediante un vector de números enteros. Por ejemplo, una red con una única capa de 100 neuronas se crea con `hidden=c(100)`, y una red con dos capas ocultas de 10 neuronas cada una con `hidden=c(10,10)`. Las capas de entrada y salida se crean automáticamente. La capa de entrada tiene tantas neuronas como predictores (tras codificar las variables categóricas). La capa de salida tiene una única neurona en problemas de regresión, y tantas neuronas como clases en los problemas de clasificación.

Preprocesado

- `standardize`: por defecto, se estandarizan los valores para que tengan media cero y varianza uno. Los modelos basados en redes no son independientes de la escala de los predictores, por lo que es fundamental estandarizarlos.
- `missing_values_handling`: las redes neuronales no aceptan observaciones con valores ausentes. H2O ofrece dos opciones: excluirlos `Skip` o imputarlos con la media `MeanImputation`.
- `shuffle_training_data`: mezclar aleatoriamente las observaciones antes de entrenar el modelo.

Aprendizaje

- `activation`: función de activación de las neuronas de las capas intermedias (*Tahn*, *Tahn with dropout*, *Rectifier*, *Rectifier with dropout*, *Maxout*, *Maxout with dropout*). Las funcines *Rectifier* y *Rectifier with dropout* suelen dar buenos resultados para un abanico amplio de situaciones. La función de activación de la capa de salida se selecciona automáticamente dependiendo de si es un problema de regresión o clasificación.

- `loss`: función de coste que se intenta minimizar durante el entrenamiento de la red. Esta es la forma en que se cuantifica el error que comete la red y en función de ello aprende, por lo tanto, es importante escoger la función de coste más adecuada para el problema en cuestión (esto es así para todos los algoritmos, no solo para redes). Por ejemplo, en problemas de regresión, las dos funciones más comunes son el error absoluto y el error cuadrático. Con el primero, el modelo resultante prioriza predecir bien la mayoría de observaciones, aunque para unas pocas se equivoque por mucho. Con el segundo, como los errores se elevan al cuadrado, el modelo resultante tiende a evitar grandes errores a cambio de aumentar un poco el error en la mayoría de observaciones.

- `epochs`: número de iteraciones de aprendizaje durante el entrenamiento de la red. Encontrar el valor óptimo de épocas es muy importante ya que, si son muy pocas, la red puede no aprender lo suficiente, pero si son demasiadas, se produce *overfiting*. Normalmente, se puede tener una idea aproximada del valor correcto en función de la complejidad de la red (cuanto más compleja más pesos hay que aprender) y el número de observaciones de entrenamiento. Gracias al sistema de *early stopping* implementado en H2O, se puede indicar un número muy elevado de épocas sin riesgo de *overfitting*, ya que el entrenamiento se detendrá cuando la métrica de validación elegida no mejore. Aun así, siempre es recomendable representar la evolución del error de entrenamiento y validación en función del número de épocas.

- `adaptive_rate`: si se indica esta opción (activada por defecto), se emplea el algoritmo (*ADADELTA*) como algoritmo de aprendizaje. Con él, se consigue que *learning rate* se adapte automáticamente (reduciéndose) a medida que el entrenamiento avanza. Es recomendable probar en primer lugar esta opción, ya que consigue un balance entre velocidad de aprendizaje y resultados bastante bueno. Este algoritmo está controlado por dos parámetros:

  - `rho`: controla el ratio en el que se va reduciendo el *learning rate* en cada iteración de aprendizaje.

  - `epsilon`: corrector para evitar divisiones entre cero.

Si se desactiva el `adaptive_rate`, entonces, el usuario debe especificar el valor de *learning rate* empleado por la red.

- `rate`: Cuanto menor sea este valor, más estable es el modelo final pero más tarda en aprender (llegar a convergencia).

- `nesterov_accelerated_gradient`: activa el método de descenso de gradiente *Nesterov Accelerated Gradient*.

Regularización

- `input_dropout_ratio`: porcentaje de *dropout* en la capa de entrada. Este tipo de regularización controla que ningún predictor influya en exceso en la red. Se recomiendan valores de 0.1 o 0.2.
- `hidden_dropout_ratios`: porcentaje de *dropout* en las capas de intermedias. Solo es aplicable si se selecciona como función de activación *TanhWithDropout*, *RectifierWithDropout*, o *MaxoutWithDropout*. Por defecto, el valor es 0.5.
- `l1`: penalización *l1*. Ver apartado de regularización GLM para más información.
- `l2`: penalización *l2*. Ver apartado de regularización GLM para más información.

El siguiente listado contiene el valor por defecto de cada uno de los argumentos de la función `h2o.deeplearning()`.

- `nfolds` = 0
- `keep_cross_validation_predictions` = FALSE
- `keep_cross_validation_fold_assignment` = FALSE
- `fold_assignment` = c("AUTO", "Random", "Modulo", "Stratified")
- `fold_column` = NULL
- `ignore_const_cols` = TRUE
- `score_each_iteration` = FALSE
- `weights_column` = NULL
- `offset_column` = NULL
- `balance_classes` = FALSE
- `class_sampling_factors` = NULL
- `max_after_balance_size` = 5
- `max_hit_ratio_k` = 0
- `checkpoint` = NULL
- `pretrained_autoencoder` = NULL
- `overwrite_with_best_model` = TRUE
- `use_all_factor_levels` = TRUE
- `standardize` = TRUE
- `activation` = c("Tanh", "TanhWithDropout", "Rectifier", "RectifierWithDropout", "Maxout", "MaxoutWithDropout")

- `hidden` = c(200, 200)
- `epochs` = 10
- `train_samples_per_iteration` = -2
- `target_ratio_comm_to_comp` = 0.05
- `seed` = -1
- `adaptive_rate` = TRUE
- `rho` = 0.99
- `epsilon` = 1e-08
- `rate` = 0.005
- `rate_annealing` = 1e-06
- `rate_decay` = 1
- `momentum_start` = 0
- `momentum_ramp` = 1e+06
- `momentum_stable` = 0
- `nesterov_accelerated_gradient` = TRUE
- `input_dropout_ratio` = 0
- `hidden_dropout_ratios` = NULL
- `l1` = 0
- `l2` = 0
- `max_w2` = 3.4028235e+38
- `initial_weight_distribution` = c("UniformAdaptive", "Uniform", "Normal")
- `initial_weight_scale` = 1
- `initial_weights` = NULL
- `initial_biases` = NULL
- `loss` = c("Automatic", "CrossEntropy", "Quadratic", "Huber", "Absolute", "Quantile")
- `distribution` = c("AUTO", "bernoulli", "multinomial", "gaussian", "poisson", "gamma", "tweedie", "laplace", "quantile", "huber")
- `quantile_alpha` = 0.5
- `tweedie_power` = 1.5
- `huber_alpha` = 0.9
- `score_interval` = 5
- `score_training_samples` = 10000
- `score_validation_samples` = 0
- `score_duty_cycle` = 0.1

- `classification_stop` = 0
- `regression_stop` = 1e-06
- `stopping_rounds` = 5
- `stopping_metric` = c("AUTO", "deviance", "logloss", "MSE", "RMSE", "MAE", "RMSLE", "AUC", "lift_top_group", "misclassification", "mean_per_class_error")
- `stopping_tolerance` = 0
- `max_runtime_secs` = 0
- `score_validation_sampling` = c("Uniform", "Stratified")
- `diagnostics` = TRUE
- `fast_mode` = TRUE
- `force_load_balance` = TRUE
- `variable_importances` = TRUE
- `replicate_training_data` = TRUE
- `single_node_mode` = FALSE
- `shuffle_training_data` = FALSE
- `missing_values_handling` = c("MeanImputation", "Skip")
- `quiet_mode` = FALSE
- `autoencoder` = FALSE
- `sparse` = FALSE
- `col_major` = FALSE
- `average_activation` = 0
- `sparsity_beta` = 0
- `max_categorical_features` = 2147483647
- `reproducible` = FALSE
- `export_weights_and_biases` = FALSE
- `mini_batch_size` = 1
- `categorical_encoding` = c("AUTO", "Enum", "OneHotInternal", "OneHotExplicit", "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited")
- `elastic_averaging` = FALSE
- `elastic_averaging_moving_rate` = 0.9
- `elastic_averaging_regularization` = 0.001
- `verbose` = FALSE

Importancia de los predictores

Para modelos de redes neuronales, una forma de cuantificar la importancia de los predictores es mediante el método *Gedeon*. Por defecto, esta opción está desactivada debido a que puede ser muy lento.

Modelo

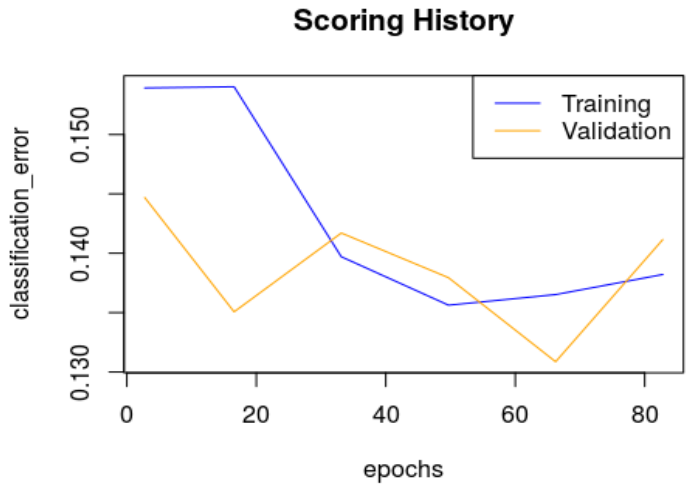
```
# Hiperparámetros que se quieren comparar.
hiperparametros <- list(hidden = list(c(64), c(128), c(256), c(512), c(1024),
                                      c(64,64), c(128,128), c(256,256),
                                      c(512, 512)))
grid_dl <- h2o.grid(
    # Algoritmo.
    algorithm = "deeplearning",
    activation = "RectifierWithDropout",
    epochs = 500,
    # Variable respuesta y predictores.
    y = var_respuesta,
    x = predictores,
    # Datos de entrenamiento.
    training_frame = datos_train_h2o,
    shuffle_training_data = FALSE,
    # Datos de validación.
    validation_frame = datos_val_h20,
    # Preprocesado.
    standardize = TRUE,
    missing_values_handling = "Skip",
    # Detención temprana.
    stopping_rounds = 3,
    stopping_metric = "AUC",
    stopping_tolerance = 0.01,
    # Hiperparámetros optimizados.
    hyper_params = hiperparametros,
    # Regularización
    l1 = 1e-5,
    l2 = 1e-5,
    # Tipo de búsqueda.
    search_criteria = list(strategy = "Cartesian"),
    seed = 123,
    grid_id = "grid_dl"
          )
```

```
# Se muestran los modelos ordenados de mayor a menor AUC.
resultados_grid <- h2o.getGrid(grid_id = "grid_dl",
                               sort_by = "auc",
                               decreasing = TRUE)

data.frame(resultados_grid@summary_table) %>% select(-model_ids)
```

```
##       hidden                auc
## 1      [128] 0.9278353550136846
## 2      [512] 0.9270888629805109
## 3     [1024] 0.9266682211693498
## 4      [256]  0.925782452310669
## 5 [512, 512] 0.9256794251647524
## 6       [64] 0.9251884669130719
## 7   [64, 64] 0.9240360843069647
## 8 [128, 128] 0.9232712359123195
## 9 [256, 256] 0.9231293461993235
```

El modelo que consigue mayor *AUC* de validación es el que tiene una arquitectura de una sola capa oculta con 128 neuronas.

```
modelo_dl_final <- h2o.getModel(resultados_grid@model_ids[[1]])
plot(modelo_dl_final, timestep = "epochs", metric = "classification_error")
```




```
# AUC de test
h2o.performance(model = modelo_dl_final, newdata = datos_test_h2o)@metrics$AUC
```

```
## [1] 0.9128816
```

Guardar un modelo H2O

Para guardar un modelo H2O ya entrenado se emplean la función `h2o.saveModel()`.

```
# Se guarda el modelo en el directorio actual
h2o.saveModel(object = modelo_dl_final, path = getwd(), force = TRUE)
```

Para cargar un modelo previamente guardado se emplea la función `h2o.loadModel()`.

```
modelo <- h2o.loadModel(path = "./grid_dl_model_8")
```

```
# Se apaga el cluster H2O.
h2o.shutdown(prompt = FALSE)
```

Bibliografía

Nykodym, T., Kraljevic, T., Hussami, N., Rao, A., and Wang, A. (Nov 2017). Generalized Linear Modeling with H2O.

Gradient Boosting Machine with H2O by Michal Malohlava & Arno Candel with assitance from Cliff Click, Hank Roark, & Viraj Parmar.

Machine Learning with R and H2O by Mark Landry with assistance from Spencer Aiello, Eric Eckstrand, Anqi Fu, & Patrick Aboyoun Edited by: Angela Bartz

Dua, D. and Karra Taniskidou, E. (2017). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Top 10 Deep Learning Tips and Tricks - Arno Candel

This work by Joaquín Amat Rodrigo is licensed under a Creative Commons Attribution 4.0 International License.

This work is licensed under a Creative Commons Attribution 4.0 International License.