Debate

Maria Plaza

04 Mai, 2020

Contents

Introducción

El objetivo de este documento es mostrar de una forma sencilla y directa cómo crear modelos de machine learning combinando lenguaje de programación R y el paquete H2O (Aiello et al. 2016). H2O ofrece un paquete R que se puede instalar desde CRAN y un paquete de Python que se puede instalar desde PyPI.H2O también se puede descargar directamente desde http://h2o.ai/download.

H2O es un producto creado por la compañía **H2O.ai** con el fin de combinar los principales algoritmos de machine learning y aprendizaje estadístico con el Big Data. Usando técnicas de compresión en memoria, H2O puede manejar miles de millones de filas de datos en memoria, incluso con un clúster bastante pequeño. H2O es de código abierto y aprendizaje automático. Son muchas las emperesas que lo emplean para obtener predicciones precisas. H2O implementa casi todos los algoritmos comunes de aprendizaje automático, como el modelado lineal generalizado (regresión lineal, regresión logística, etc.), Naïve Bayes, análisis de componentes principales, series de tiempo, agrupación de k-medias y otros. H2O también implementa los mejores algoritmos de su clase, como Random Forest, Gradient Boosting y Deep Learning a escala. Los algoritmos avanzados,

están integrados para ayudar a los diseñadores de aplicaciones a crear aplicaciones más inteligentes a través de API elegantes (Candel et al. 2016).

Antes de proceder al entrenamiento de un modelo, hay determinados pasos que se deben llevar cabo, como son: exploración de los datos, transformaciones, selección de predictores, etc. Sin embargo, vamos a considerar que los datos ya están listos para ser empleados por los algoritmos del aprendizaje y vamos a centrarnos por tanto en este último paso. Existen multiples tutoriales y documentos que muestran de forma detallada cada una de las etapas que forman parte del modelado por ejemplo en los siguientes links: https://machinelearningmastery.com/machine-learning-in-r-step-by-step/ https://lgatto.github.io/IntroMachineLearningWithR/an-introduction-to-machine-learning-with-r.html

Nos vamos a centrar en este caso en el lenguaje R con el empleo de Rstudio. Es importante considerar 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. Debemos tener cuidado cuando pasamos los datos desde H2O a R, ya que implica cargar todos los datos y, si son demasiados pueden ocupar toda la memoria. Para evitar este tipo de problemas, se recomiend realizar todos los pasos posibles (filtrado, agregaciones, cálculo de nuevas columnas...) con las funciones de H2O antes de transferir los datos.

Instalación H2O

Si hay que instalarlo o actualizarlo, es mejor hacerlo desde su cuenta de AWS según el siguiente código.

O podemos instalarlo directamente desde r. Una vez que H2O ha sido instalado, 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.

```
> # install.packages("h2o")
> library(h2o)
```

Iniciamos un servidor H2O de 1 nodo en nuestro PC local y permitamos que use todos los núcleos de CPU y hasta 2 GB de memoria:

```
> h2o.init(nthreads=-1, max_mem_size="2G") # Creación de un cluster local.
```

```
 Connection successful!

R is connected to the H2O cluster:
    H2O cluster uptime:         3 hours 32 minutes
    H2O cluster timezone:       Europe/Berlin
    H2O data parsing timezone:  UTC
    H2O cluster version:        3.30.0.1
    H2O cluster version age:    1 month
    H2O cluster name:           H2O_started_from_R_plazagma_aoc952
    H2O cluster total nodes:    1
    H2O cluster total memory:   1.46 GB
    H2O cluster total cores:    8
    H2O cluster allowed cores:  8
    H2O cluster healthy:        TRUE
    H2O Connection ip:          localhost
    H2O Connection port:        54321
    H2O Connection proxy:       NA
    H2O Internal Security:      FALSE
    H2O API Extensions:         Amazon S3, Algos, AutoML, Core V3, TargetEncoder, Core V4
    R Version:                  R version 4.0.0 (2020-04-24)
```

```
> # -1 indica que se empleen todos los cores disponibles
> # Máxima memoria disponible para el cluster.
> h2o.removeAll() ## clean slate - En caso de que haya algo disponible en memoria
```

Tras iniciar el cluster (local), se muestran por pantalla sus características, entre las que están: el número de cores activados (8), la memoria total del cluster (1.78GB), 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().

La función *h2o.deeplearning* se ajusta a los modelos de aprendizaje profundo de H2O desde dentro de R. Podemos ejecutar el ejemplo desde la página del manual utilizando la examplefunción, o ejecutar una demostración más larga desde el h2opaquete utilizando la demofunción.

```
> args(h2o.deeplearning)
> help(h2o.deeplearning)
> example(h2o.deeplearning)
> #demo(h2o.deeplearning)  #requires user interaction
>
> # Para que no se muestre la barra de progreso.
> h2o.no_progress()
```

Los datos

Para aprender a usar este paquete, lo mejor es utilizando un set de datos. Para los ejemplos de este documento se emplean dos set de datos, el primero se corresponde con *Contraceptive Method Choice Data Set* disponible en UCI Machine Learning Repository. Este set de datos contiene información sobre un subconjunto de la Encuesta Nacional de Prevalencia de Anticonceptivos de Indonesia de 1987. (http://archive.ics.uci.edu/ml/datasets/Contraceptive+Method+Choice)

Este conjunto de datos es un subconjunto de la Encuesta nacional de prevalencia de anticonceptivos de Indonesia de 1987. Las muestras son mujeres casadas que no estaban embarazadas o no saben si estaban en el momento de la entrevista. El problema es predecir la elección actual del método anticonceptivo (sin uso, métodos a largo plazo o métodos a corto plazo) de una mujer en función de sus características demográficas y socioeconómicas.

1. Edad de la esposa (numérica)
2. Educación de la esposa (categórica) 1 = baja, 2, 3, 4 = alta
3. Educación del esposo (categórica) 1 = baja, 2, 3, 4 = alta
4. Número de hijos nacidos (numérico)
5. La religión de la esposa (binaria) 0 = No islam, 1 = Islam
6. ¿La esposa ahora está trabajando? (binario) 0 = Sí, 1 = No
7. Ocupación del esposo (categórico) 1, 2, 3, 4
8. Índice de nivel de vida (categórico) 1 = bajo, 2, 3, 4 = alto
9. Exposición a los medios (binario) 0 = Bueno, 1 = No bueno
10. Método anticonceptivo utilizado (atributo de clase) 1 = Sin uso, 2 = A largo plazo, 3 = A corto plazo

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.

El segundo set de datos se denomina *Covtype*, un conjunto de gran tamaño y con un estructura algo más complicada, que emplearemos en el modelo de Deep Learning. Contiene 581012 filas con 54 atributos.Los datos se encuentran disponibles en el enlace: http://archive.ics.uci.edu/ml/datasets/Covertype

Se utiliza para predecir el tipo de cubierta forestal solo a partir de variables cartográficas (sin datos de

detección remota). Los datos están en forma cruda (sin escala) y contienen columnas binarias (0 o 1) de datos para variables cualitativas independientes (áreas silvestres y tipos de suelo).

Se proporciona el nombre del atributo, el tipo de atributo, la unidad de medida y una breve descripción. El tipo de cubierta forestal es el problema de clasificación. El orden de este listado corresponde al orden de los números a lo largo de las filas de la base de datos.

Nombre / Tipo de datos / Medición / Descripción Elevación / cuantitativo / metros / Elevación en metros Aspecto / cuantitativo / acimut / Aspecto en grados azimut Pendiente / cuantitativo / grados / Pendiente en grados Horizontal_Distance_To_Hydrology / quantitative / meters / Horz Dist a las características de agua superficial más cercanas Vertical_Distance_To_Hydrology / cuantitativo / metros / Dist. vertical a las características de agua superficial más cercanas Horizontal_Distance_To_Roadways / cuantitativo / metros / Dist Horz a la carretera más cercana Hillshade_9am / quantitative / 0 a 255 index / Hillshade index a las 9am, solsticio de verano Hillshade_Noon / quantitative / 0 a 255 index / Hillshade index al mediodía, soltice de verano Hillshade_3pm / quantitative / 0 a 255 index / Hillshade index a las 3pm, summer solstice Horizontal_Distance_To_Fire_Points / cuantitativo / metros / Dist Horz a los puntos de ignición de incendios forestales más cercanos Wilderness_Area (4 columnas binarias) / cualitativo / 0 (ausencia) o 1 (presencia) / Designación de área silvestre Soil_Type (40 columnas binarias) / cualitativo / 0 (ausencia) o 1 (presencia) / Designación de tipo de suelo Cover_Type (7 tipos) / entero / 1 a 7 / Designación de tipo de cubierta forestal

```
> # Carga de datos en el cluster H2O desde url.
> library(R.utils)
> # datos_h2o <- h2o.importFile(path   = url, header = TRUE,
> # sep    = ",", destination_frame = "datos_h2o")
>
> # SET DATA 1
> # Carga de datos en R y transferencia a H2O.
> # url <- "http://archive.ics.uci.edu/ml/machine-learning-databases/cmc/cmc.data"
> # destino <- "./cmc.data"
> # download.file(url, destino)
> # gunzip("cmc.data", remove=FALSE)
>
> datos_r <- read.csv(file = "./cmc.data", header = TRUE)
> datos_h2o <- as.h2o(x = datos_r, destination_frame = "datos_h2o")
```

```
  |                                                                      |
```

```
> # SET DATA 2
> # Carga de datos en R y transferencia a H2O.
> # url1 <- "http://archive.ics.uci.edu/ml/machine-learning-databases/covtype/covtype.data.gz"
> # destino1 <- "./covtype.data"
> # download.file(url1, destino1)
> # gunzip("covtype.data", remove=FALSE)
>
> datos_r1 <- read.csv(file = "./covtype.data", header = TRUE)
> datos_h2o1 <- as.h2o(x = datos_r1, destination_frame = "datos_h2o1")
```

```
  |                                                                      |
```

Todos los datos, asi como el codigo R empoleado se encuentran en el siguiente repositorio GitHUb:

https://github.com/mariaplaza/Debate_MachineLearning

Exploración de los datos

Aunque el conjunto de datos seleccionado para este ejemplo es lo suficientemente pequeño para cargarlo en memoria y emplear las funciones de r, vamos a emplear funciones propias de H2O.

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

```
[1] 1472   10
```

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

```
 [1] "X24"  "X2"   "X3"   "X3.1" "X1"   "X1.1" "X2.1" "X3.2" "X0"   "X1.2"
```

Renombramos las columnas, para entender mejor las variables:

```
> colnames(datos_h2o) <- c("age", "wife_education", "husband_education",
+                          "children","religion", "work_situation",
+                          "husband_occupation","standard_living", "exposition",
+                          "Contraceptive_Method")
```

La función h2o.describe() nos da 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 emplea el nombre *enum* para los datos de tipo factor o character.

```
> h2o.describe(datos_h2o)[,1:10]
```

Label	Type	Missing	Zeros	PosInf	NegInf	Min	Max	Mean	Sigma
age	int	0	0	0	0	16	49	32.5441576	8.2270272
wife_education	int	0	0	0	0	1	4	2.9592391	1.0150313
husband_education	int	0	0	0	0	1	4	3.4300272	0.8165491
children	int	0	97	0	0	0	16	3.2615489	2.3593406
religion	int	0	220	0	0	0	1	0.8505435	0.3566591
work_situation	int	0	369	0	0	0	1	0.7493207	0.4335515
husband_occupation	int	0	0	0	0	1	4	2.1379076	0.8651437
standard_living	int	0	0	0	0	1	4	3.1338315	0.9764860
exposition	int	0	1363	0	0	0	1	0.0740489	0.2619395
Contraceptive_Method	int	0	0	0	0	1	3	1.9205163	0.8763454

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()*. De esta forma, con la función *h2o.cor()* podemos caluclar 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, method = "Pearson", na.rm = TRUE)[,1:7]
```

age0	wife_education0	husband_education0	children0	religion0	work_situation0	husband_occupation0
1.0000000	-0.0487103	-0.0532747	0.5402478	-0.1393066	-0.0394917	-0.2027199
-0.0487103	1.0000000	0.6182764	-0.1939694	-0.2327888	-0.0618542	-0.3961473
-0.0532747	0.6182764	1.0000000	-0.1875716	-0.1783246	0.0013058	-0.3370943
0.5402478	-0.1939694	-0.1875716	1.0000000	0.0739534	0.0973706	-0.0230118
-0.1393066	-0.2327888	-0.1783246	0.0739534	1.0000000	0.0696847	0.0844684
-0.0394917	-0.0618542	0.0013058	0.0973706	0.0696847	1.0000000	0.0142962
-0.2027199	-0.3961473	-0.3370943	-0.0230118	0.0844684	0.0142962	1.0000000
0.1844570	0.3614740	0.3574769	-0.0060564	-0.1962821	-0.0764601	-0.2938499
0.1131511	-0.3363730	-0.2888277	0.1336412	0.0603292	0.0019398	0.1138981
-0.1638145	0.1492043	0.1028990	0.0827245	-0.0249829	0.0548773	0.0180543

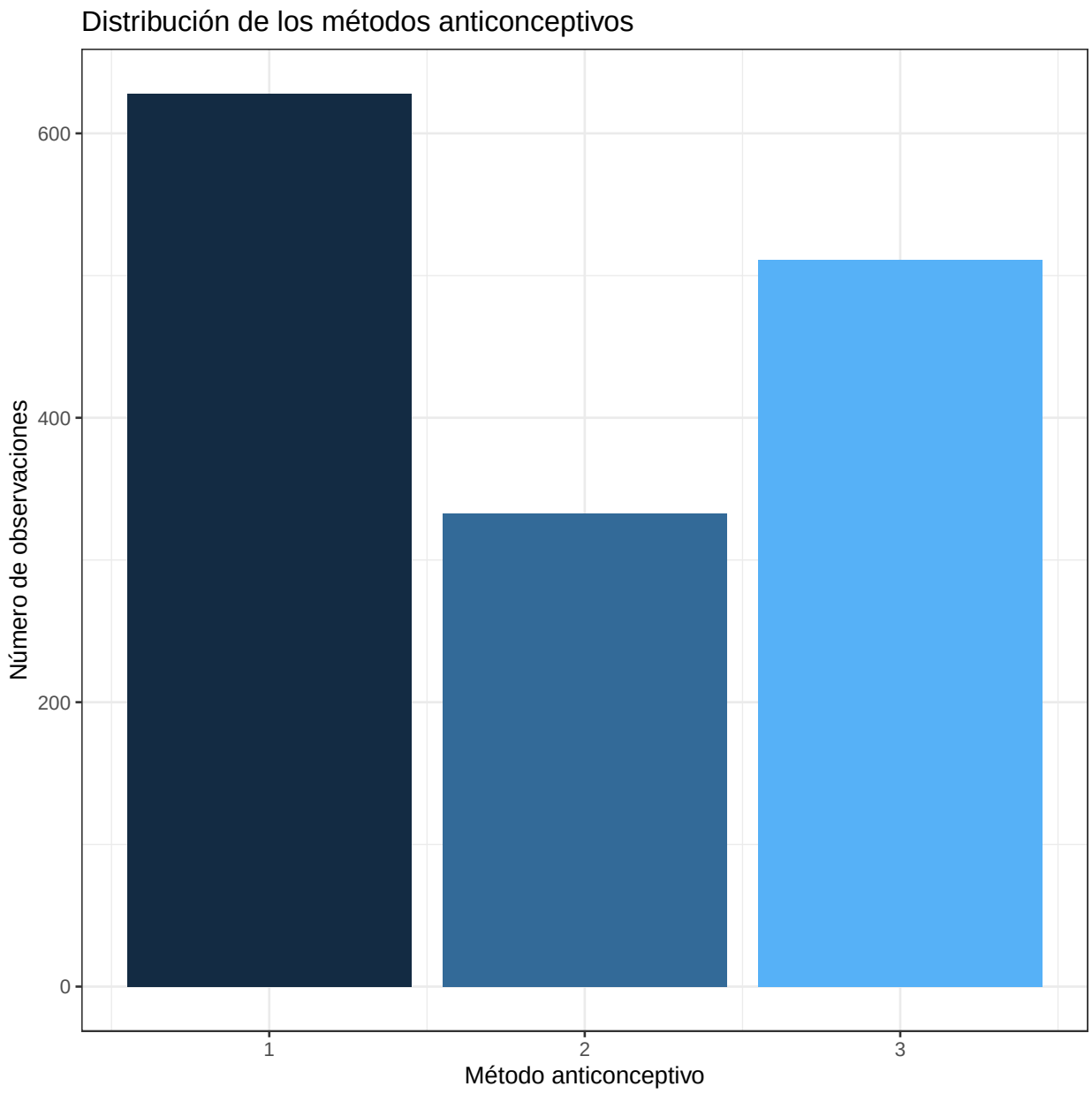
Para contar el número de observaciones de cada clase en una variable categórica, como es en este caso la variable respuesta *método anticonceptivo*, se emplea la función h2o.table().

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

Contraceptive_Method	Count
1	628
2	333
3	511

Y podemos realizar una grafica para tner una idea representativa:

```
> library(ggplot2)
> ggplot(
+   data = metodo,
+   aes(x = Contraceptive_Method, y = Count, fill = Contraceptive_Method)) +
+ geom_col() +
+ theme_bw() +
+ labs(
+   x = "Método anticonceptivo", y = "Número de observaciones",
+   title = "Distribución de los métodos anticonceptivos") +
+ theme(legend.position = "none")
```



Especificando el número de muestras de entrenamiento

El objetivo del problema es crear un modelo capaz de predecir correctamente el tipo de métdo anticonceptivo empleado.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 predecir nuevos datos.

Si el número de observaciones es limitado, como es el caso en el data set *Contraceptive* 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. Sin embargo en el data set *Covtype* podremos hacer tres subconjuntos.

Mostraremos ambas posibilidades. En el ejemplo de modelo GLM se emplearemos validación cruzada con

el data set *Contraceptive*, mientras que en el otro modelo (Deep Learning), se emplea un solo conjunto de validación; además cuando se trabaja con un gran conjunto de datos, no suele ser factible aplicar validación cruzada.

La función h2o.splitFrame() realiza particiones aleatorias:

```
> # 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.
> separaciones     <- h2o.splitFrame(data = datos_h2o, ratios = c(0.7), seed = 123)
> datos_train_h2o <- h2o.assign(data = separaciones[[1]], key = "datos_train_H2O")
> datos_test_h2o  <- h2o.assign(data = separaciones[[2]], key = "datos_test_H2O")
```

Veamos el número de observaciones de cada clase, en numero y porcentaje

```
> h2o.table(datos_train_h2o$Contraceptive_Method)
```

```
  Contraceptive_Method Count
1                    1   432
2                    2   238
3                    3   370

[3 rows x 2 columns]
```

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

```
  Contraceptive_Method     Count
1         0.0009615385 0.4153846
2         0.0019230769 0.2288462
3         0.0028846154 0.3557692

[3 rows x 2 columns]
```

```
> h2o.table(datos_test_h2o$Contraceptive_Method)
```

```
  Contraceptive_Method Count
1                    1   196
2                    2    95
3                    3   141

[3 rows x 2 columns]
```

```
> # En porcentaje
> h2o.table(datos_test_h2o$Contraceptive_Method)/h2o.nrow(datos_test_h2o)
```

```
  Contraceptive_Method     Count
1          0.002314815 0.4537037
2          0.004629630 0.2199074
3          0.006944444 0.3263889

[3 rows x 2 columns]
```

Preprocesado de datos

Una de las funciones fundamentales de H2O es que incorpora y automatiza gran parte de las transformaciones necesarias para que los datos puedan ser ingeridos por los algoritmos de machine learning. Estas funciones se aplican automáticamente cuando el modelo se emplea para predecir nuevas observaciones. garantizando

que no se viola la condición de que ninguna información procedente de las observaciones de test participe o influya en el ajuste del modelo.

- identifica automáticamente que variables son categóricas y crea internamente las variables dummy correspondientes
- estandariza los predictores numéricos antes de ajustar los modelos para que todos tengan media cero y varianza uno.
- excluye las columnas con valor constante, ya que no se deben de incluir en un modelo predictores que contengan un único valor (varianza cero), al no aportar información.
- Balance de clases: con el argumento *balance_classes* se puede indicar que antes de ajustar el modelo se equilibren las clases, si es necesario, indicando *undersampling* u *oversampling*.

Modelos

H2O incorpora varios algoritmos de machine learning, pudiendo trabajar con ellos de forma distribuida y/o en paralelo. Algunos de ellos son:

- 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

En este documento se muestran ejemplos con Generalized Linear Model (GLM) y Deep Learning (Neural Networks).

Optimización de hiperparámetros

Los parametros que incluyen algunos modelos deben ser introducidos manualmente por el analista, son los hiperparámetros. sin embargo, no se puede conocer de antemano cuál es el adecuado. La forma más común de encontrar los valores óptimos es probando diferentes posibilidades, lo que se conoce como *tunning*.

H2O posee 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.

Generalized Linear Model (GLM)

Se comprueba que la variable respuesta es de tipo factor y se define la variable respuesta y los predictores, en este caso, tendremos en cuenta todos los predictores disponibles.

```
> datos_train_h2o$Contraceptive_Method <- h2o.asfactor(datos_train_h2o$Contraceptive_Method)
> datos_test_h2o$Contraceptive_Method  <- h2o.asfactor(datos_test_h2o$Contraceptive_Method)
> h2o.isfactor(datos_train_h2o$Contraceptive_Method)
```

```
[1] TRUE
```

```
> var_respuesta <- "Contraceptive_Method"
> predictores   <- setdiff(h2o.colnames(datos_h2o), var_respuesta)
```

El siguiente paso es crear el modelo y lo validamos mediente 5-CV para estimar el error:

```
> modelo_glm <- h2o.glm(
+                 y = var_respuesta,
+                 x = predictores,
+                 training_frame = datos_train_h2o,
+                 family = "multinomial",
+                 link   = "family_default",
+                 standardize  = TRUE,
+                 balance_classes   = FALSE,
+                 ignore_const_cols = TRUE,
+                 # Especificamos que hacer con observaciones incompletas o missing values
+                 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_glm"
+               )
  |                                                                                                 |
> summary(modelo_glm)

Model Details:
==============

H2OMultinomialModel: glm
Model Key:  modelo_glm
GLM Model: summary
       family        link                                regularization
1 multinomial multinomial Elastic Net (alpha = 0.95, lambda = 0.00758 )
                                                                  lambda_search
1 nlambda = 100, lambda.max = 0.1126, lambda.min = 0.00758, lambda.1se = 0.02788
  number_of_predictors_total number_of_active_predictors number_of_iterations
1                         30                          16                    50
    training_frame
1 RTMP_sid_8aa0_14

H2OMultinomialMetrics: glm
** Reported on training data. **

Training Set Metrics:
=====================

Extract training frame with `h2o.getFrame("RTMP_sid_8aa0_14")`
MSE: (Extract with `h2o.mse`) 0.3631632
RMSE: (Extract with `h2o.rmse`) 0.6026302
Logloss: (Extract with `h2o.logloss`) 0.9527777
```

```
Mean Per-Class Error: 0.5046235
Null Deviance: (Extract with `h2o.nulldeviance`) 2225.797
Residual Deviance: (Extract with `h2o.residual_deviance`) 1981.778
R^2: (Extract with `h2o.r2`) 0.5268848
AIC: (Extract with `h2o.aic`) NaN
Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
========================================================================
Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
         1   2   3  Error          Rate
1      276  27 129 0.3611 =   156 / 432
2       72  82  84 0.6555 =   156 / 238
3      133  51 186 0.4973 =   184 / 370
Totals 481 160 399 0.4769 = 496 / 1,040

Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
========================================================================
Top-3 Hit Ratios:
  k hit_ratio
1 1  0.523077
2 2  0.819231
3 3  1.000000



H2OMultinomialMetrics: glm
** Reported on cross-validation data. **
** 5-fold cross-validation on training data (Metrics computed for combined holdout predictions) **

Cross-Validation Set Metrics:
=====================

Extract cross-validation frame with `h2o.getFrame("RTMP_sid_8aa0_14")`
MSE: (Extract with `h2o.mse`) 0.3719226
RMSE: (Extract with `h2o.rmse`) 0.6098546
Logloss: (Extract with `h2o.logloss`) 0.9736146
Mean Per-Class Error: 0.5183606
Null Deviance: (Extract with `h2o.nulldeviance`) 2229.716
Residual Deviance: (Extract with `h2o.residual_deviance`) 2025.118
R^2: (Extract with `h2o.r2`) 0.5154733
AIC: (Extract with `h2o.aic`) NaN
Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,xval = TRUE)`
========================================================================
Top-3 Hit Ratios:
  k hit_ratio
1 1  0.508654
2 2  0.801923
3 3  1.000000


Cross-Validation Metrics Summary:
                            mean          sd cv_1_valid cv_2_valid cv_3_valid
accuracy                0.5080403  0.04188426  0.5595855  0.5050505        0.5
err                     0.4919597  0.04188426 0.44041452  0.4949495        0.5
err_count                   102.2   10.917875       85.0       98.0      108.0
```

```
logloss                  0.9715609  0.03368678  0.9285955 0.97379977 0.96178883
max_per_class_error      0.6882268   0.0725429  0.6136364  0.6944444  0.6862745
mean_per_class_accuracy 0.47919658 0.042023573  0.5323797 0.47059423 0.47121236
mean_per_class_error    0.52080345 0.042023573  0.4676203  0.5294058  0.5287876
mse                     0.37210798 0.016712738 0.34995595 0.37490472 0.36566347
null_deviance            445.94327    41.22159  412.38373  419.35666  464.59827
r2                        0.512867 0.036284026  0.5431267  0.5412683  0.5202229
residual_deviance        404.11713   38.625362  358.43787   385.6247   415.4928
rmse                     0.6098837 0.013684281 0.59157073  0.6122946 0.60470116
                        cv_4_valid cv_5_valid
accuracy                0.44615385  0.5294118
err                      0.5538462 0.47058824
err_count                    108.0      112.0
logloss                  1.0224774 0.97114307
max_per_class_error      0.8039216 0.64285713
mean_per_class_accuracy 0.41949734 0.50229925
mean_per_class_error     0.5805027 0.49770075
mse                     0.39606187 0.37395388
null_deviance            423.35864   510.0191
r2                      0.45425957  0.5054575
residual_deviance        398.76617   462.2641
rmse                    0.62933445 0.61151767

Scoring History:
            timestamp   duration iteration lambda predictors deviance_train
1 2020-05-04 18:28:14  0.000 sec         1  ,11E0          3          2.140
2 2020-05-04 18:28:14  0.007 sec         3   ,1E0          5          2.131
3 2020-05-04 18:28:14  0.015 sec         5 ,93E-1          6          2.117
4 2020-05-04 18:28:14  0.022 sec         7 ,85E-1          7          2.100
5 2020-05-04 18:28:14  0.028 sec         9 ,78E-1          7          2.076
  deviance_test deviance_xval deviance_se
1            NA         2.140       0.010
2            NA         2.140       0.010
3            NA         2.141       0.009
4            NA         2.137       0.009
5            NA         2.128       0.009

---
            timestamp   duration iteration lambda predictors deviance_train
27 2020-05-04 18:28:14  0.145 sec        45  ,1E-1         18          1.909
28 2020-05-04 18:28:14  0.148 sec        46 ,91E-2         18          1.907
29 2020-05-04 18:28:14  0.151 sec        47 ,83E-2         19          1.906
30 2020-05-04 18:28:14  0.154 sec        48 ,76E-2         19          1.906
31 2020-05-04 18:28:14  0.156 sec        49 ,69E-2         19          1.905
32 2020-05-04 18:28:14  0.159 sec        50 ,63E-2         19          1.904
   deviance_test deviance_xval deviance_se
27            NA         1.948       0.029
28            NA         1.948       0.030
29            NA         1.948       0.030
30            NA         1.948       0.031
31            NA         1.948       0.031
32            NA         1.948       0.032

Variable Importances: (Extract with `h2o.varimp`)
```

```
================================================

            variable relative_importance scaled_importance percentage
1           children          0.73578240        1.00000000 0.25779102
2     wife_education          0.72398173        0.98396174 0.25365650
3                age          0.71669679        0.97406080 0.25110412
4    standard_living          0.26016881        0.35359477 0.09115356
5 husband_occupation          0.10153838        0.13800056 0.03557531
6         exposition          0.09404980        0.12782285 0.03295158
7           religion          0.08944386        0.12156292 0.03133783
8     work_situation          0.06890158        0.09364396 0.02414057
9  husband_education          0.06361833        0.08646351 0.02228952
```

Si llamamos directamente al modelo se muestra toda la información disponible, como el tipo de modelo, coeficientes de regresión obtenidos, métricas... Para poder acceder directamente a la información de interés, H2O posee una serie de funciones que extraen información concreta del modelo.

```
> library(dplyr)
> # Coeficientes de regresión de cada uno de los predictores.
> as.data.frame(modelo_glm@model$coefficients_table) %>% head()
```

names	coefs_class_0	coefs_class_1	coefs_class_2	std_coefs_class_0	std_coefs_class_1	std_coe
Intercept	1.2524943	-2.6925869	0.7966393	-0.8870978	-1.6002609	
age	0.0221923	0.0000000	-0.0653080	0.1817727	0.0000000	
wife_education	-0.2775127	0.4417350	0.0000000	-0.2793393	0.4446425	
husband_education	-0.0796286	0.0000000	0.0000000	-0.0636183	0.0000000	
children	-0.3113427	0.0017038	0.0000000	-0.7317779	0.0040045	
religion	0.0267928	-0.2298272	0.0000000	0.0093385	-0.0801053	

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

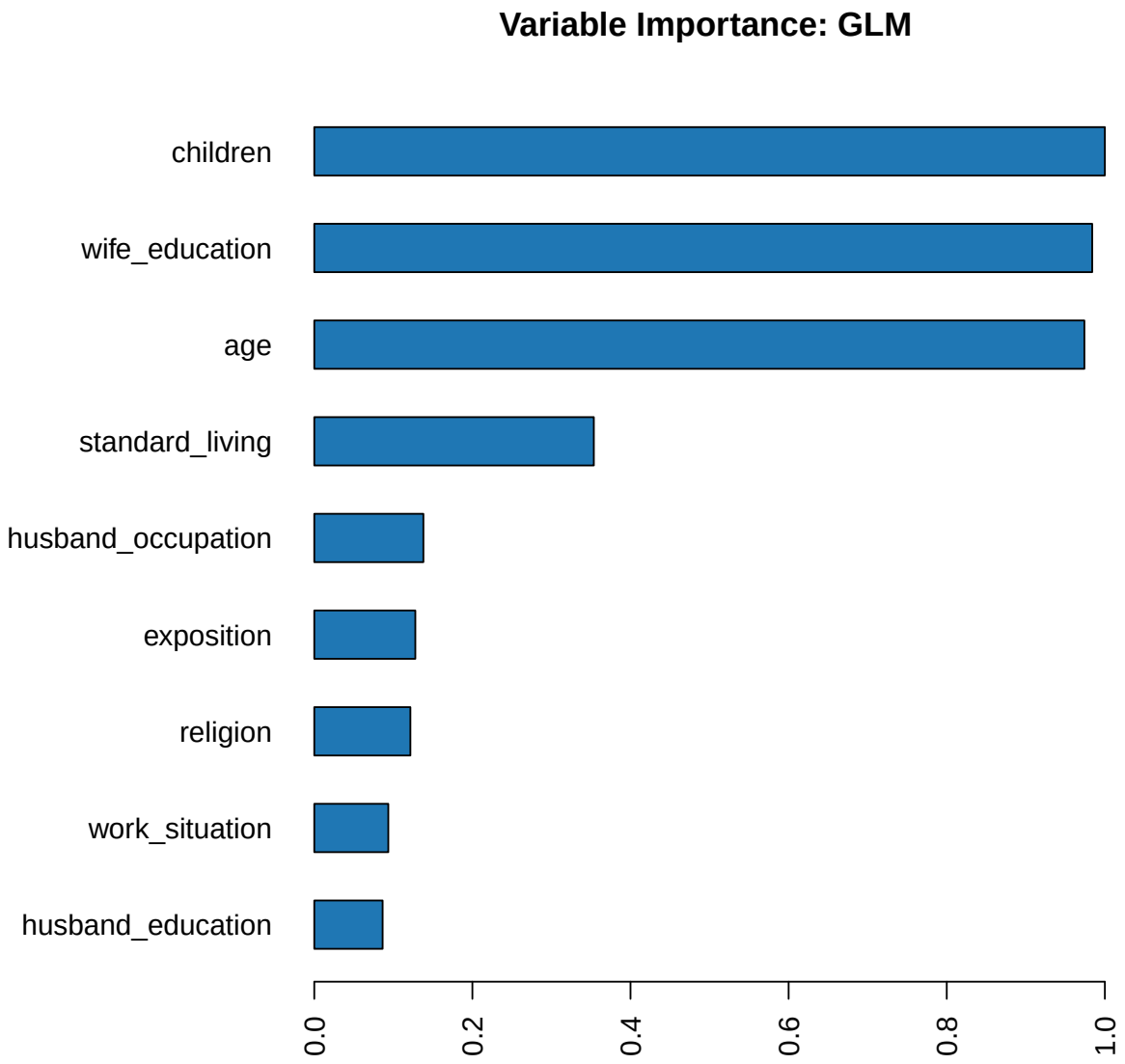
```
[1] "Intercept"         "age"               "wife_education"
[4] "husband_education" "children"          "religion"
[7] "standard_living"   "exposition"
```

La importancia de los predictores puede estudiarse a partir de las siguientes funcioes que incorpora el paquete *H2o*:

```
> # Equivalente:
> h2o.varimp(modelo_glm)
```

variable	relative_importance	scaled_importance	percentage
children	0.7357824	1.0000000	0.2577910
wife_education	0.7239817	0.9839617	0.2536565
age	0.7166968	0.9740608	0.2511041
standard_living	0.2601688	0.3535948	0.0911536
husband_occupation	0.1015384	0.1380006	0.0355753
exposition	0.0940498	0.1278228	0.0329516
religion	0.0894439	0.1215629	0.0313378
work_situation	0.0689016	0.0936440	0.0241406
husband_education	0.0636183	0.0864635	0.0222895

```
> h2o.varimp_plot(modelo_glm)
```



Podemos obtener toda una serie de métricas a partir de los datos de entrenamiento con las siguientes funciones:

```
> h2o.performance(model = modelo_glm, train = TRUE)

H2OMultinomialMetrics: glm
** Reported on training data. **

Training Set Metrics:
=====================

Extract training frame with `h2o.getFrame("RTMP_sid_8aa0_14")`
MSE: (Extract with `h2o.mse`) 0.3631632
RMSE: (Extract with `h2o.rmse`) 0.6026302
```

```
Logloss: (Extract with `h2o.logloss`) 0.9527777
Mean Per-Class Error: 0.5046235
Null Deviance: (Extract with `h2o.nulldeviance`) 2225.797
Residual Deviance: (Extract with `h2o.residual_deviance`) 1981.778
R^2: (Extract with `h2o.r2`) 0.5268848
AIC: (Extract with `h2o.aic`) NaN
Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
=========================================================================
Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
         1   2   3  Error          Rate
1      276  27 129 0.3611 =   156 / 432
2       72  82  84 0.6555 =   156 / 238
3      133  51 186 0.4973 =   184 / 370
Totals 481 160 399 0.4769 = 496 / 1.040

Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
=======================================================================
Top-3 Hit Ratios:
  k hit_ratio
1 1  0.523077
2 2  0.819231
3 3  1.000000
```

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 (el modelo creado *modelo_glm*) y un nuevo set de datos (*datos_test_h2o*).

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

  |                                                                                                 |
> predicciones

  predict        p1         p2        p3
1       3 0.2129959 0.29458523 0.4924188
2       1 0.6718292 0.11396786 0.2142030
3       1 0.8352068 0.04291943 0.1218738
4       1 0.7331536 0.08493500 0.1819114
5       1 0.7207251 0.04118371 0.2380912
6       1 0.5912863 0.15733441 0.2513793

[432 rows x 4 columns]
```

El resultado devuelto por esta función es una tabla con 4 columnas en este caso, una con la clase predicha y otras tres con la probabilidad de pertenecer a cada una de las clases. Si el nuevo set de datos incluye la variable respuesta, se pueden calcular métricas que cuantifican el grado de acierto. Por otro lado, calculamos de forma manual la precisión del modelo, que como vemos no es muy elevada (0.5138889), quiza debido al pequeño conjunto de datos.

```
> h2o.performance(model = modelo_glm, newdata = datos_test_h2o)

H2OMultinomialMetrics: glm

Test Set Metrics:
=====================

MSE: (Extract with `h2o.mse`) 0.3575179
```

```
RMSE: (Extract with `h2o.rmse`) 0.597928
Logloss: (Extract with `h2o.logloss`) 0.9432952
Mean Per-Class Error: 0.5206894
Null Deviance: (Extract with `h2o.nulldeviance`) 916.0252
Residual Deviance: (Extract with `h2o.residual_deviance`) 815.0071
R^2: (Extract with `h2o.r2`) 0.5319733
AIC: (Extract with `h2o.aic`) NaN
Confusion Matrix: Extract with `h2o.confusionMatrix(<model>, <data>)`)
=========================================================================
Confusion Matrix: Row labels: Actual class; Column labels: Predicted class
         1  2   3  Error        Rate
1      129 15  52 0.3418 =  67 / 196
2       25 35  35 0.6316 =   60 / 95
3       61 22  58 0.5887 =  83 / 141
Totals 215 72 145 0.4861 = 210 / 432

Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>, <data>)`
=======================================================================
Top-3 Hit Ratios:
  k hit_ratio
1 1  0.513889
2 2  0.831019
3 3  1.000000
```

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

```
[1] 0.5138889
```

Para intentar mejorar el modelo, intentamos una búsqueda del valor *alpha*, ya que antes se ha empleado un valor fijo. Por ello repetimos el modelo, comparando en este caso distintos valores de *alpha*. Para estimar la capacidad predictiva de cada modelo se emplea validación cruzada con 10 particiones.

```
> # Valores de alpha que se van a comparar.
> param_alpha <- list(alpha = c(0, 0.1, 0.5, 0.95, 1))
>
> grid_glm <- h2o.grid(
+     # Algoritmo y parámetros
+     algorithm      = "glm",
+     family = "multinomial",
+     link   = "family_default",
+     # 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    = param_alpha,
+     # 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 por precision.
> resultados_grid <- h2o.getGrid(
+                      grid_id = "grid_glm",
+                      sort_by = "accuracy",
+                      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 accuracy
   alpha        model_ids           accuracy
1  [0.1] grid_glm_model_2 0.5153846153846153
2 [0.95] grid_glm_model_4 0.5144230769230769
3  [1.0] grid_glm_model_5 0.5144230769230769
4  [0.0] grid_glm_model_1             0.5125
5  [0.5] grid_glm_model_3  0.510576923076923
```

En este caso, los resultados de los 5 modelos son prácticamente idénticos, con una precisión muy baja. Podríamos intentar de nueo otro modelo cambiando algunos de los argumentos, por ejemplo el tipo de familia que implementa el model, "poisson", "gamma" o "tweedie" podrian ser algunas de las opciones.

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

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

Deep Learning (Neural Networks)

Como sabemos, 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 nuestro caso, H2O incorpora redes neuronales de tipo Multi-layer - feedforward - neural networks, que se caracterizan por tener una o múltiples capas intermedias, con una estructura full conected, lo que significa que cada neurona está conectada con todas las neuronas de la capa siguiente.

Los modelos de Deep Learning ofrecidos por **H2O** tienen un número muy elevado de parámetros configurables.

Para la gran mayoría de casos, los valores por defecto dan buenos resultados, sin embargo, es conveniente conocer, al menos, los más influyentes, que incluyen funciones de arquitectura, pre-procesado, aprendizaje y regularización.

Veamos su funcionamiento con el conjunto de datos de *Covtype*. Dividimos los datos en tres grupos: 60% para entrenamiento, 20% para validación (ajuste de hiperparámetros) y 20% para pruebas finales. Pero primero escogemos solo las primeros columnas con las descripcion de las zonas y la variable clase (columna 55).

```
> # Dimensiones del set de datos
> h2o.dim(datos_h2o1)
```

```
[1] 581011     55
```

```
> # Nombre de las columnas
> h2o.colnames(datos_h2o1)
```

```
 [1] "X2596" "X51"   "X3"    "X258"  "X0"    "X510"  "X221"  "X232"  "X148"
[10] "X6279" "X1"    "X0.1"  "X0.2"  "X0.3"  "X0.4"  "X0.5"  "X0.6"  "X0.7"
[19] "X0.8"  "X0.9"  "X0.10" "X0.11" "X0.12" "X0.13" "X0.14" "X0.15" "X0.16"
[28] "X0.17" "X0.18" "X0.19" "X0.20" "X0.21" "X0.22" "X0.23" "X0.24" "X0.25"
[37] "X0.26" "X0.27" "X0.28" "X0.29" "X0.30" "X0.31" "X1.1"  "X0.32" "X0.33"
[46] "X0.34" "X0.35" "X0.36" "X0.37" "X0.38" "X0.39" "X0.40" "X0.41" "X0.42"
[55] "X5"
```

```
> seleccion <- c("X2596", "X51", "X3", "X258", "X0", "X510", "X221", "X232",
+                "X148", "X6279", "X1", "X0.1",  "X0.2" , "X0.3", "X5")
> datos_h2o1 <- datos_h2o1[seleccion]
> str(datos_h2o1)
```

```
Class 'H2OFrame' <environment: 0x000000001bf51fc8>
 - attr(*, "op")= chr "cols"
 - attr(*, "eval")= logi TRUE
 - attr(*, "id")= chr "RTMP_sid_8aa0_18"
 - attr(*, "nrow")= int 581011
 - attr(*, "ncol")= int 15
 - attr(*, "types")=List of 15
  ..$ : chr "int"
  ..$ : chr "int"
  ..$ : chr "int"
  ..$ : chr "int"
  ..$ : chr "int"
  ..$ : chr "int"
  ..$ : chr "int"
  ..$ : chr "int"
  ..$ : chr "int"
  ..$ : chr "int"
  ..$ : chr "int"
  ..$ : chr "int"
  ..$ : chr "int"
  ..$ : chr "int"
  ..$ : chr "int"
 - attr(*, "data")='data.frame':    10 obs. of  15 variables:
  ..$ X2596: num  2590 2804 2785 2595 2579 ...
  ..$ X51  : num  56 139 155 45 132 45 49 45 59 201
  ..$ X3   : num  2 9 18 2 6 7 4 9 10 4
  ..$ X258 : num  212 268 242 153 300 270 234 240 247 180
  ..$ X0   : num  -6 65 118 -1 -15 5 7 56 11 51
```

```
  ..$ X510 : num  390 3180 3090 391 67 633 573 666 636 735
  ..$ X221 : num  220 234 238 220 230 222 222 223 228 218
  ..$ X232 : num  235 238 238 234 237 225 230 221 219 243
  ..$ X148 : num  151 135 122 150 140 138 144 133 124 161
  ..$ X6279: num  6225 6121 6211 6172 6031 ...
  ..$ X1   : num  1 1 1 1 1 1 1 1 1 1
  ..$ X0.1 : num  0 0 0 0 0 0 0 0 0 0
  ..$ X0.2 : num  0 0 0 0 0 0 0 0 0 0
  ..$ X0.3 : num  0 0 0 0 0 0 0 0 0 0
  ..$ X5   : num  5 2 2 5 2 5 5 5 5 5
```

Renombramos las columnas, para entender mejor las variables:

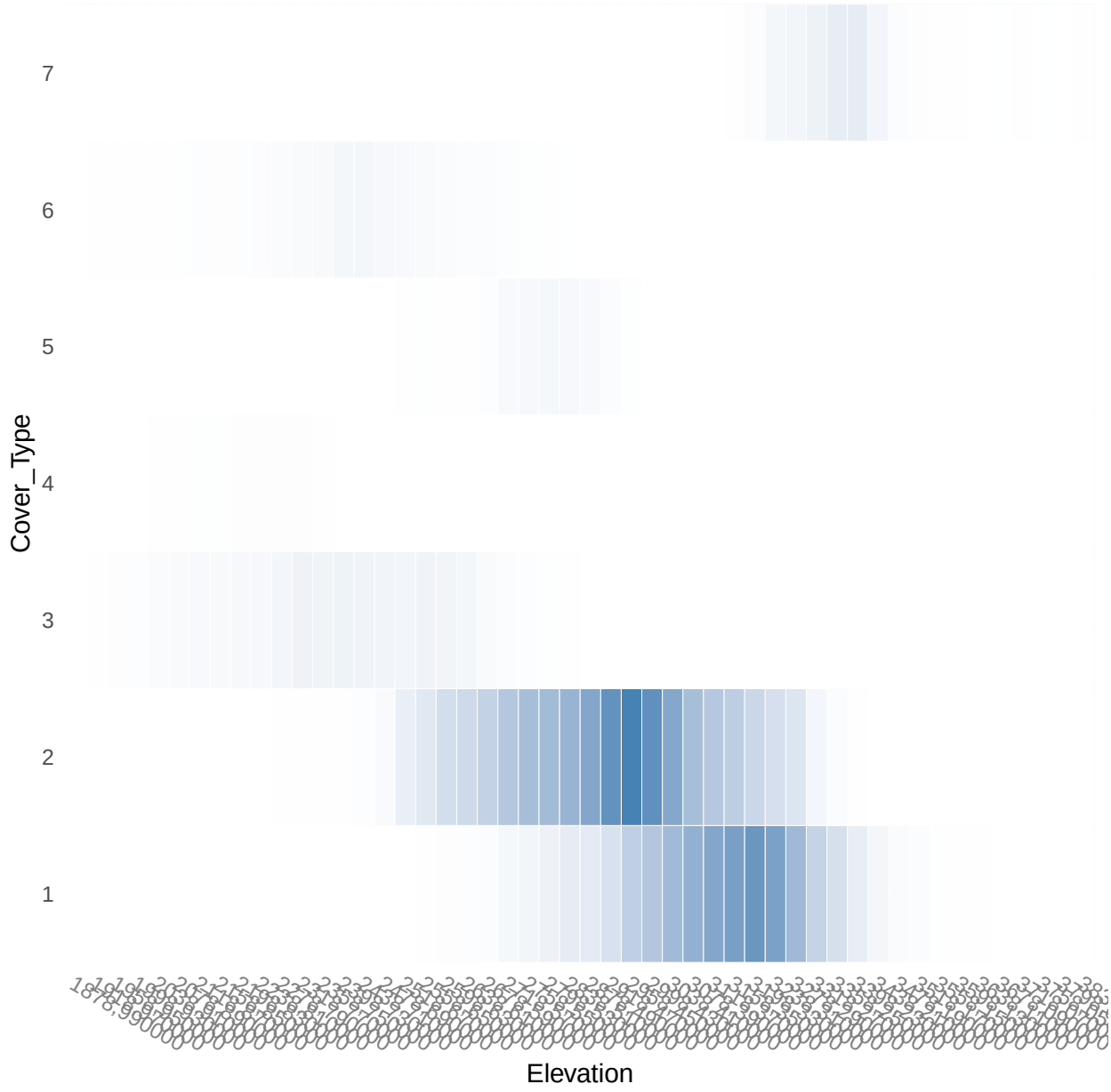
```
> colnames(datos_h2o1) <- c("Elevation", "Aspect", "Slope",
+                           "Horizontal_Distance_To_Hydrology",
+                            "Vertical_Distance_To_Hydrology",
+                           "Horizontal_Distance_To_Roadways",
+                           "Hillshade_9am","Hillshade_Noon",
+                           "Hillshade_3pm",
+                           "Horizontal_Distance_To_Fire_Points",
+                           "Wilderness_Area1","Wilderness_Area2",
+                           "Wilderness_Area3", "Wilderness_Area4",
+                           "Cover_Type")
```

Creamos los grupos con los que vamos a trabajar:

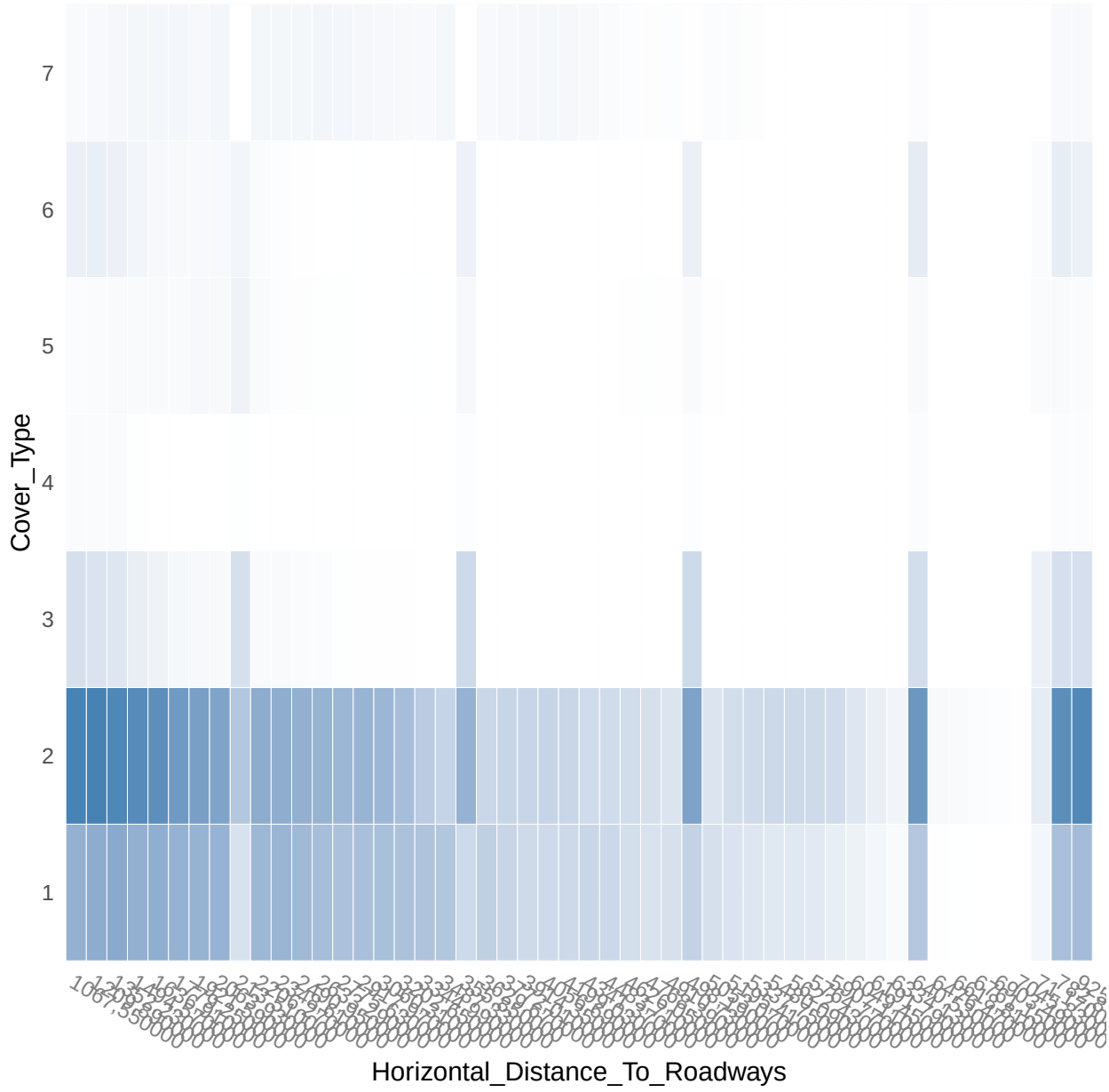
```
> splits <- h2o.splitFrame(datos_h2o1, c(0.6,0.2), seed=1234)
> train  <- h2o.assign(splits[[1]], "train.hex") # 60%
> valid  <- h2o.assign(splits[[2]], "valid.hex") # 20%
> test   <- h2o.assign(splits[[3]], "test.hex")  # 20%
```

POdemos ahora realizar diagramas de dispersión mediante binning (para columnas categóricas y numéricas) y familiarizarnos con el conjunto de datos.

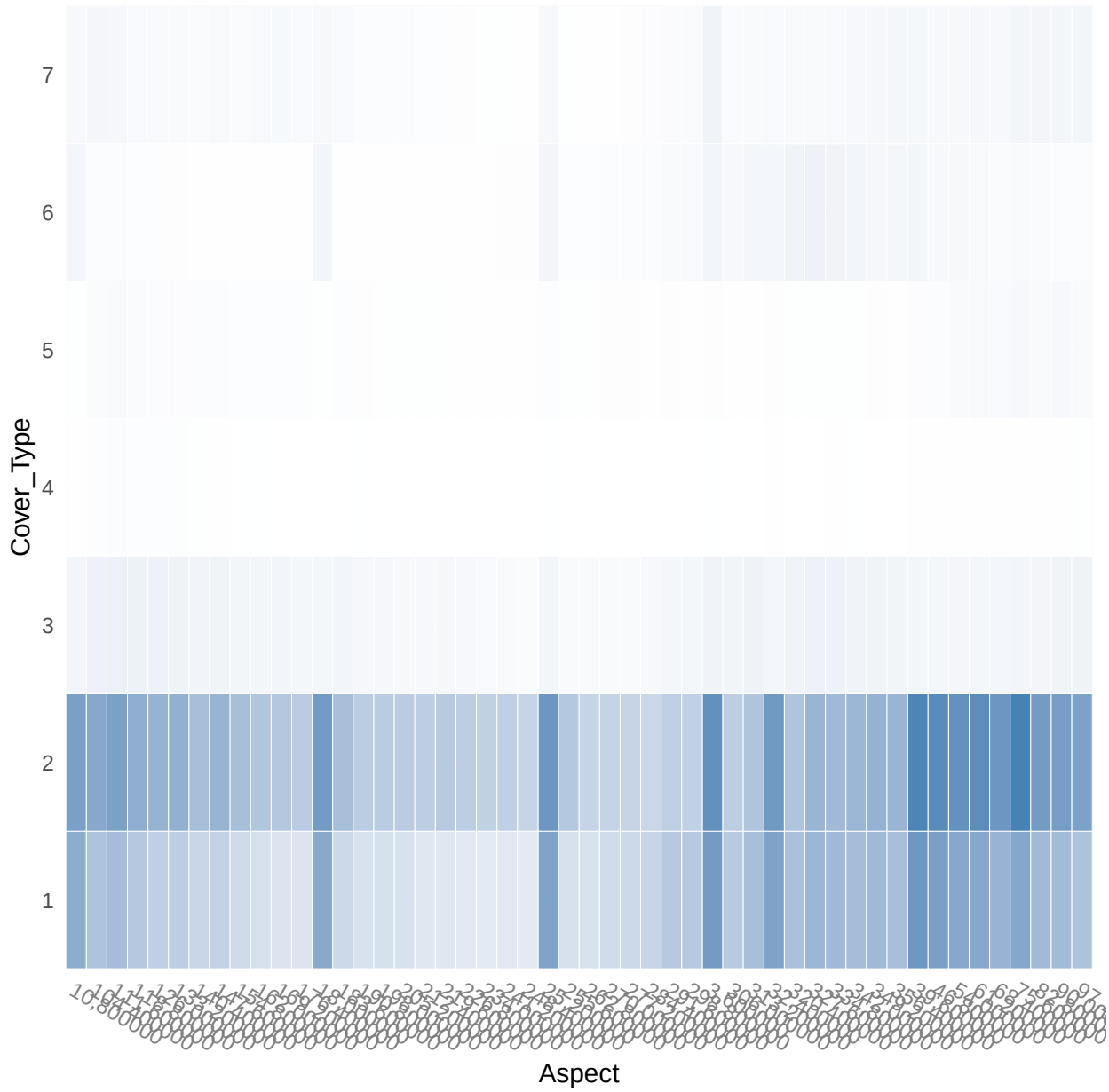
```
> #dev.new(noRStudioGD=FALSE) #direct plotting output to a new window
> par(mfrow=c(1,1)) # reset canvas
> plot(h2o.tabulate(datos_h2o1,"Elevation","Cover_Type"))
```



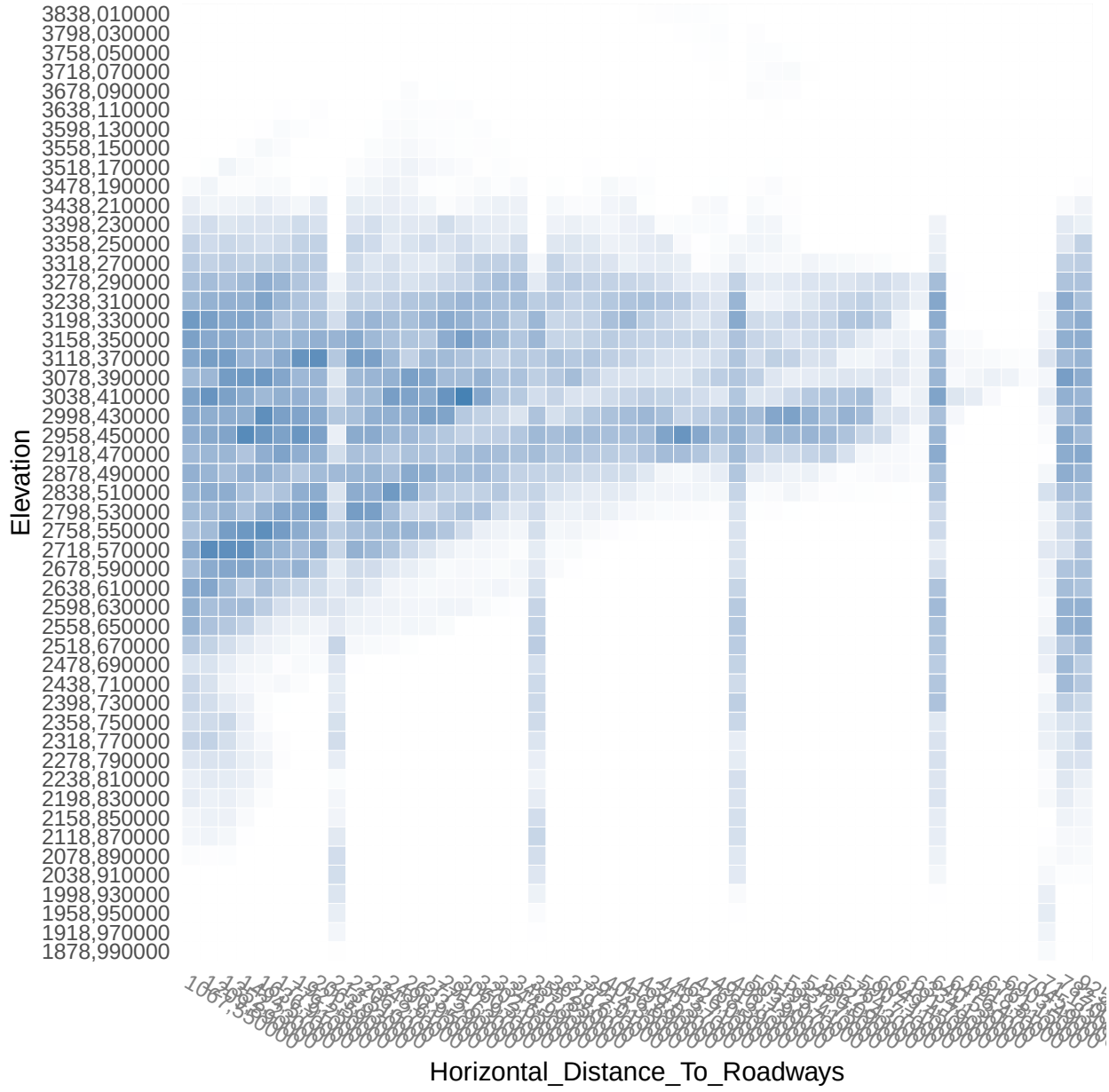

```
> plot(h2o.tabulate(datos_h2o1,"Horizontal_Distance_To_Roadways","Cover_Type"))
```




```
> plot(h2o.tabulate(datos_h2o1,"Aspect","Cover_Type"))
```




```
> plot(h2o.tabulate(datos_h2o1,"Horizontal_Distance_To_Roadways","Elevation" ))
```



Creacion del modelo

Ejecutemos nuestro primer modelo de Deep Learning en el conjunto de datos de tipo *covtype*. Queremos predecir la columna *Cover_Type*, una característica categórica con 7 niveles, y el modelo de Aprendizaje Profundo tendrá la tarea de realizar la clasificación (multi-clase). Utiliza los otros 12 predictores del conjunto de datos, de los cuales 10 son numéricos y 2 son categóricos con un total de 44 niveles.

```
> response <- "Cover_Type"
> predictors <- setdiff(names(datos_h2o1), response)
> predictors
```

```
 [1] "Elevation"                         "Aspect"
 [3] "Slope"                             "Horizontal_Distance_To_Hydrology"
 [5] "Vertical_Distance_To_Hydrology"    "Horizontal_Distance_To_Roadways"
 [7] "Hillshade_9am"                     "Hillshade_Noon"
```

```
 [9] "Hillshade_3pm"                      "Horizontal_Distance_To_Fire_Points"
[11] "Wilderness_Area1"                   "Wilderness_Area2"
[13] "Wilderness_Area3"                   "Wilderness_Area4"
```

Para que sea más rapido, solo empleamos un *epoch* (una unica pasada sobre los datos de entrenamiento).

```
> modelo1 <- h2o.deeplearning(
+   model_id="dl_model_first",
+   training_frame=train,
+   validation_frame=valid,## validation dataset: uutilizado para anotar y detenerse
+   x=predictors,
+   y=response,
+   #activation="Rectifier",## por defecto
+   #hidden=c(200,200), ## por defecto: 2 capas ocultas con 200 neuronas cada una
+   epochs=1,
+   variable_importances=T ## no posible por defecto
+ )
```

```
  |                                                                              |
```

```
> summary(modelo1)
```

```
Model Details:
==============

H2ORegressionModel: deeplearning
Model Key:  dl_model_first
Status of Neuron Layers: predicting Cover_Type, regression, gaussian distribution, Quadratic loss, 43.4
  layer units      type dropout       l1       l2 mean_rate rate_rms momentum
1     1    14     Input  0.00 %       NA       NA        NA       NA       NA
2     2   200 Rectifier  0.00 % 0.000000 0.000000  0.002380 0.002105 0.000000
3     3   200 Rectifier  0.00 % 0.000000 0.000000  0.050681 0.069830 0.000000
4     4     1    Linear      NA 0.000000 0.000000  0.000485 0.000335 0.000000
  mean_weight weight_rms mean_bias bias_rms
1          NA         NA        NA       NA
2    0.009371   0.183125  0.394768 0.211183
3   -0.037541   0.108719  0.912294 0.101889
4    0.038707   0.103152  0.253973 0.000000

H2ORegressionMetrics: deeplearning
** Reported on training data. **
** Metrics reported on temporary training frame with 9943 samples **

MSE:  1.052228
RMSE:  1.025782
MAE:  0.6756147
RMSLE:  0.2768449
Mean Residual Deviance :  1.052228


H2ORegressionMetrics: deeplearning
** Reported on validation data. **
** Metrics reported on full validation frame **

MSE:  1.045236
RMSE:  1.022368
```

```
MAE:  0.6794943
RMSLE:  0.2777639
Mean Residual Deviance :  1.045236
```

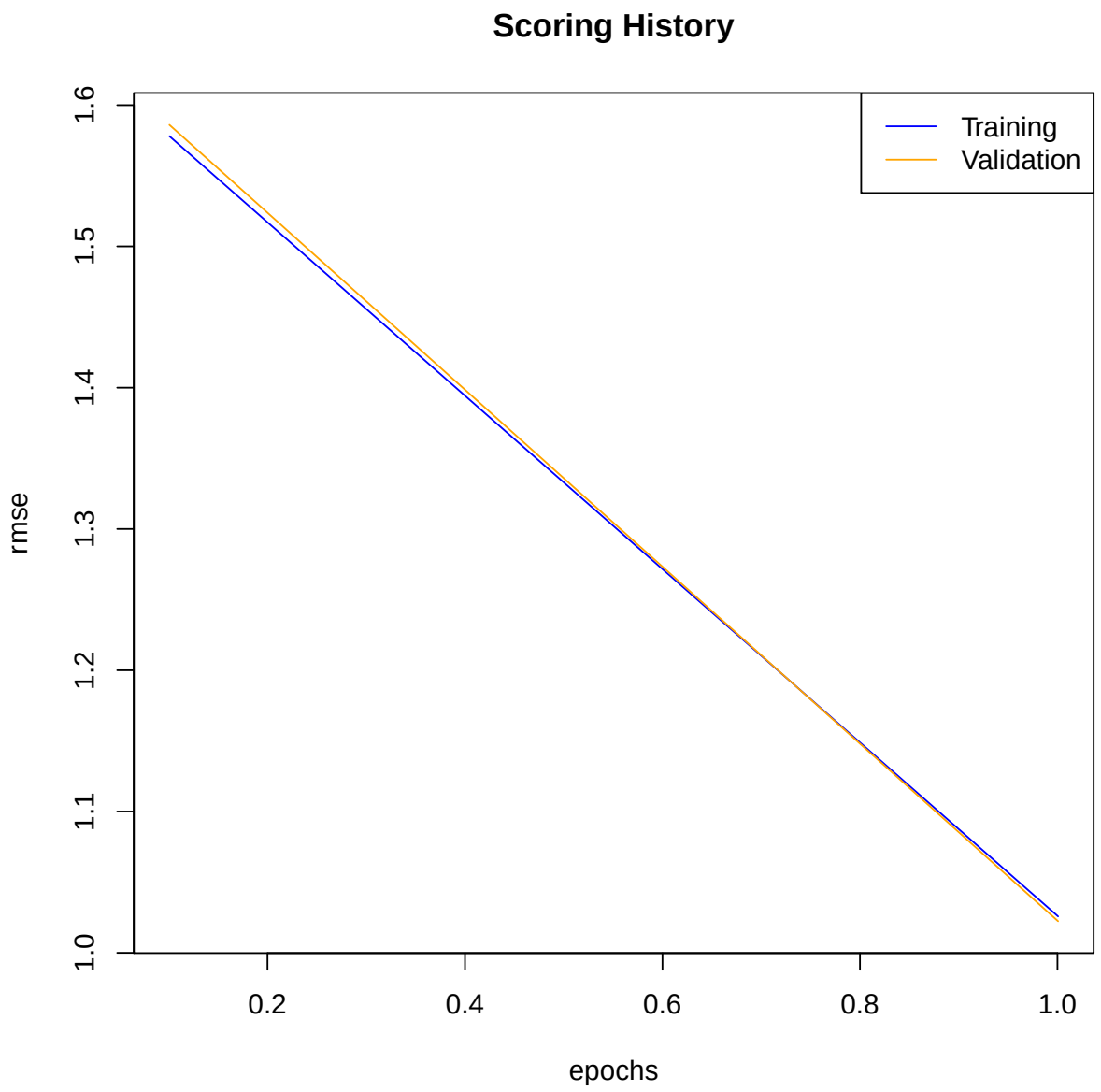
```
Scoring History:
            timestamp   duration training_speed  epochs iterations
1 2020-05-04 18:28:23  0.000 sec             NA 0.00000          0
2 2020-05-04 18:28:25  2.560 sec  22063 obs/sec 0.10064          1
3 2020-05-04 18:28:32  9.935 sec  43217 obs/sec 1.00065         10
        samples training_rmse training_deviance training_mae training_r2
1      0.000000            NA                NA           NA          NA
2  35125.000000       1.57802           2.49015      1.21538    -0.27698
3 349240.000000       1.02578           1.05223      0.67561     0.46040
  validation_rmse validation_deviance validation_mae validation_r2
1              NA                   NA             NA            NA
2         1.58605              2.51556        1.21869      -0.28614
3         1.02237              1.04524        0.67949       0.46560
```

```
Variable Importances: (Extract with `h2o.varimp`)
=================================================
```

```
Variable Importances:
                            variable relative_importance scaled_importance
1                          Elevation            1.000000          1.000000
2     Horizontal_Distance_To_Roadways            0.899024          0.899024
3  Horizontal_Distance_To_Fire_Points            0.830127          0.830127
4                    Wilderness_Area1            0.709394          0.709394
5                    Wilderness_Area3            0.552268          0.552268
6                    Wilderness_Area2            0.537204          0.537204
7                    Wilderness_Area4            0.467852          0.467852
8                       Hillshade_9am            0.458056          0.458056
9                      Hillshade_Noon            0.445861          0.445861
10   Horizontal_Distance_To_Hydrology            0.431724          0.431724
11                             Slope            0.388883          0.388883
12                      Hillshade_3pm            0.371236          0.371236
13     Vertical_Distance_To_Hydrology            0.363807          0.363807
14                             Aspect            0.348883          0.348883
   percentage
1    0.128134
2    0.115196
3    0.106368
4    0.090898
5    0.070764
6    0.068834
7    0.059948
8    0.058693
9    0.057130
10   0.055319
11   0.049829
12   0.047568
13   0.046616
```

```
14   0.044704
```

```
> plot(modelo1)
```



La importancia de cada una de las variables en los modelos de redes neuronales son difíciles de calcular, y existen muchas dificultades. *H2O* Deep Learning ha implementado el método de Gedeon y devuelve importancias de variables relativas en orden descendente de importancia.

```
> head(as.data.frame(h2o.varimp(modelo1)))
```

variable	relative_importance	scaled_importance	percentage
Elevation	1.0000000	1.0000000	0.1281342
Horizontal_Distance_To_Roadways	0.8990236	0.8990236	0.1151957
Horizontal_Distance_To_Fire_Points	0.8301275	0.8301275	0.1063677
Wilderness_Area1	0.7093942	0.7093942	0.0908976

variable	relative_importance	scaled_importance	percentage
Wilderness_Area3	0.5522678	0.5522678	0.0707644
Wilderness_Area2	0.5372036	0.5372036	0.0688341

El siguiente paso es ejecutar otra red más pequeña y dejamos que se detenga automáticamente una vez que converge la tasa de clasificación errónea (específicamente, si el promedio móvil de longitud 2 no mejora al menos un 1% en 2 eventos de puntuación consecutivos). También se muestra el conjunto de validación en 10,000 filas para una puntuación más rápida.

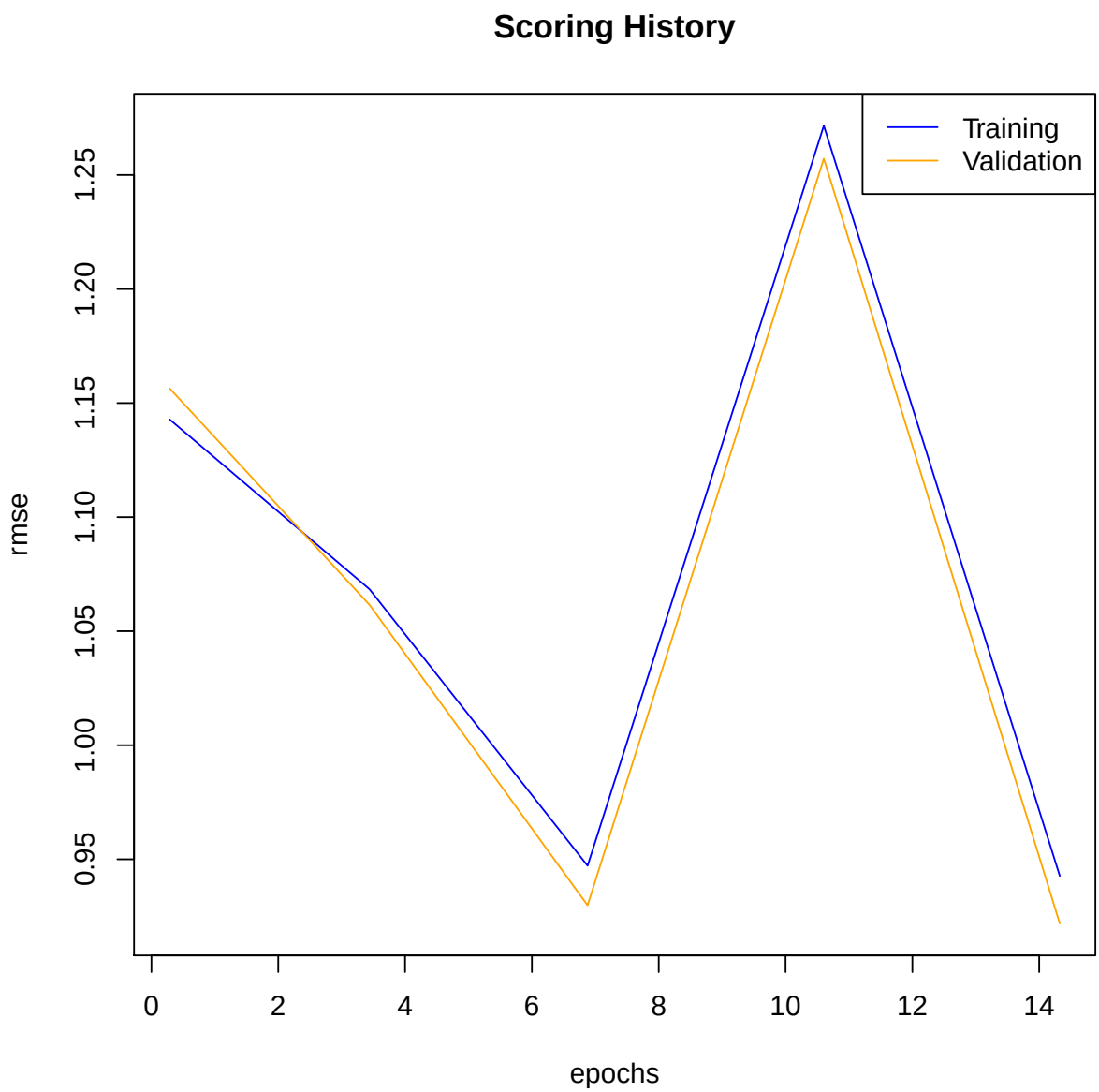
```
> modelo2 <- h2o.deeplearning(
+   model_id="dl_model_faster",
+   training_frame=train,
+   validation_frame=valid,
+   x=predictors,
+   y=response,
+   hidden=c(32,32,32),  ## una pequeña red, corre más rápido
+   epochs=100000,       ## aunque se espera que converja antes...
+   score_validation_samples=10000, ## conjunto de datos de validación (más rápido)
+   stopping_rounds=2,
+   stopping_metric="MSE", ## podria ser "RMSE","logloss","r2"
+   stopping_tolerance=0.01
+ )

  |                                                                      |
```

```
> summary(modelo2)

Model Details:
==============

H2ORegressionModel: deeplearning
Model Key:  dl_model_faster
Status of Neuron Layers: predicting Cover_Type, regression, gaussian distribution, Quadratic loss, 2.625
  layer units      type dropout       l1       l2 mean_rate rate_rms momentum
1     1    14     Input  0.00 %       NA       NA        NA       NA       NA
2     2    32 Rectifier  0.00 % 0.000000 0.000000  0.000821 0.000829 0.000000
3     3    32 Rectifier  0.00 % 0.000000 0.000000  0.000997 0.001749 0.000000
4     4    32 Rectifier  0.00 % 0.000000 0.000000  0.002578 0.008065 0.000000
5     5     1    Linear      NA 0.000000 0.000000  0.000280 0.000149 0.000000
  mean_weight weight_rms mean_bias bias_rms
1          NA         NA        NA       NA
2    0.011220   0.512085  0.706872 0.792938
3   -0.070862   0.393566  0.876974 0.198566
4   -0.122099   0.431070  0.766918 0.743798
5   -0.075959   0.556018  1.217720 0.000000

H2ORegressionMetrics: deeplearning
** Reported on training data. **
** Metrics reported on temporary training frame with 10034 samples **

MSE:  0.8887089
RMSE:  0.9427136
MAE:  0.5925201
RMSLE:  0.2615857
```

```
Mean Residual Deviance :  0.8887089


H2ORegressionMetrics: deeplearning
** Reported on validation data. **
** Metrics reported on temporary validation frame with 9949 samples **

MSE:  0.8498096
RMSE:  0.9218512
MAE:  0.5897616
RMSLE:  0.2589432
Mean Residual Deviance :  0.8498096




Scoring History:
            timestamp   duration training_speed   epochs iterations
1 2020-05-04 18:28:34  0.000 sec             NA  0.00000          0
2 2020-05-04 18:28:34  0.778 sec 143713 obs/sec  0.28742          1
3 2020-05-04 18:28:39  5.976 sec 204347 obs/sec  3.43922         12
4 2020-05-04 18:28:45 11.062 sec 219469 obs/sec  6.87812         24
5 2020-05-04 18:28:50 16.399 sec 227742 obs/sec 10.60361         37
6 2020-05-04 18:28:55 21.657 sec 232554 obs/sec 14.32453         50
         samples training_rmse training_deviance training_mae training_r2
1       0.000000            NA                NA           NA          NA
2  100312.000000       1.14285           1.30610      0.73461     0.32877
3 1200336.000000       1.06840           1.14149      0.70235     0.41337
4 2400561.000000       0.94717           0.89714      0.61103     0.53894
5 3700809.000000       1.27159           1.61694      0.97780     0.16902
6 4999463.000000       0.94271           0.88871      0.59252     0.54327
  validation_rmse validation_deviance validation_mae validation_r2
1              NA                  NA             NA            NA
2         1.15642             1.33731        0.73978       0.33623
3         1.06154             1.12686        0.69674       0.44068
4         0.92983             0.86459        0.60319       0.57086
5         1.25715             1.58043        0.97753       0.21555
6         0.92185             0.84981        0.58976       0.57820

Variable Importances: (Extract with `h2o.varimp`)
=================================================

Variable Importances:
                            variable relative_importance scaled_importance
1                    Wilderness_Area3            1.000000          1.000000
2                           Elevation            0.969589          0.969589
3     Horizontal_Distance_To_Roadways            0.882409          0.882409
4  Horizontal_Distance_To_Fire_Points            0.797878          0.797878
5                    Wilderness_Area2            0.784861          0.784861
6                    Wilderness_Area1            0.704381          0.704381
7                    Wilderness_Area4            0.677878          0.677878
8                      Hillshade_Noon            0.391654          0.391654
9    Horizontal_Distance_To_Hydrology            0.352817          0.352817
10                     Hillshade_9am            0.309966          0.309966
```

```
11                      Hillshade_3pm            0.282547          0.282547
12                              Slope            0.237789          0.237789
13     Vertical_Distance_To_Hydrology            0.223592          0.223592
14                             Aspect            0.176445          0.176445
   percentage
1    0.128340
2    0.124437
3    0.113248
4    0.102400
5    0.100729
6    0.090400
7    0.086999
8    0.050265
9    0.045280
10   0.039781
11   0.036262
12   0.030518
13   0.028696
14   0.022645
```

```
> plot(modelo2)
```



Tuning

Con algunos ajustes (tuning), es posible obtener una tasa de error del conjunto de prueba inferior al 10% en aproximadamente un minuto. Las tasas de error por debajo del 5% son posibles con modelos más grandes. Tenga en cuenta que los métodos deep tree pueden ser más efectivos para este conjunto de datos que Deep Learning, ya que dividen directamente el espacio en sectores, lo que parece ser mas apropiado en este caso.

```
> modelo3 <- h2o.deeplearning(
+   model_id="dl_model_tuned",
+   training_frame=train,
+   validation_frame=valid,
+   x=predictors,
+   y=response,
+   overwrite_with_best_model=F, ## modelo final después de 10 epoch, incluso si no es el mejor.
```

```
+   hidden=c(128,128,128),  ## más capas ocultas -> interacciones más complejas
+   epochs=10,              ## para que sea lo suficientemente corto
+   score_validation_samples=1000,
+   score_duty_cycle=0.025, ## no anota más del 2.5% del tiempo
+   # adaptive_rate= FALSE, ## tasa de aprendizaje ajustada manualmente
+   rate=0.01,
+   rate_annealing=2e-6,
+   momentum_start=0.2, ## impulso sintonizado manualmente
+   momentum_stable=0.4,
+   momentum_ramp=1e7,
+   l1=1e-5,           ## agrega cierta regularización L1 / L2
+   l2=1e-5,
+   max_w2=10          ## ayuda a la estabilidad del rectificador
+ )
```

```
  |                                                                              |
```

```
> summary(modelo3)
```

```
Model Details:
==============

H2ORegressionModel: deeplearning
Model Key:  dl_model_tuned
Status of Neuron Layers: predicting Cover_Type, regression, gaussian distribution, Quadratic loss, 35.0
  layer units      type dropout       l1       l2 mean_rate rate_rms momentum
1     1    14     Input  0.00 %       NA       NA        NA       NA       NA
2     2   128 Rectifier  0.00 % 0.000010 0.000010  0.001292 0.000950 0.000000
3     3   128 Rectifier  0.00 % 0.000010 0.000010  0.002247 0.002791 0.000000
4     4   128 Rectifier  0.00 % 0.000010 0.000010  0.010596 0.027082 0.000000
5     5     1    Linear      NA 0.000010 0.000010  0.000287 0.000182 0.000000
  mean_weight weight_rms mean_bias bias_rms
1          NA         NA        NA       NA
2    0.023963   0.331185  0.496726 0.363709
3   -0.046475   0.185954  0.871697 0.130920
4   -0.057037   0.193662  0.748431 0.379154
5   -0.022931   0.269781  2.004985 0.000000

H2ORegressionMetrics: deeplearning
** Reported on training data. **
** Metrics reported on temporary training frame with 10002 samples **

MSE:  0.8356351
RMSE:  0.9141308
MAE:  0.6101008
RMSLE:  0.2734858
Mean Residual Deviance :  0.8356351


H2ORegressionMetrics: deeplearning
** Reported on validation data. **
** Metrics reported on temporary validation frame with 996 samples **

MSE:  0.7099983
```

```
RMSE:  0.842614
MAE:  0.5759556
RMSLE:  0.2526945
Mean Residual Deviance :  0.7099983
```

Scoring History:

	timestamp	duration	training_speed	epochs	iterations
1	2020-05-04 18:28:56	0.000 sec	NA	0.00000	0
2	2020-05-04 18:28:59	3.582 sec	29049 obs/sec	0.28541	1
3	2020-05-04 18:29:06	10.216 sec	39986 obs/sec	1.14548	4
4	2020-05-04 18:29:12	16.071 sec	44294 obs/sec	2.00423	7
5	2020-05-04 18:29:17	21.605 sec	46992 obs/sec	2.86425	10
6	2020-05-04 18:29:23	27.011 sec	48850 obs/sec	3.72452	13
7	2020-05-04 18:29:28	32.321 sec	50228 obs/sec	4.58288	16
8	2020-05-04 18:29:34	37.658 sec	51157 obs/sec	5.44098	19
9	2020-05-04 18:29:39	42.923 sec	51944 obs/sec	6.29922	22
10	2020-05-04 18:29:44	48.205 sec	52548 obs/sec	7.15819	25
11	2020-05-04 18:29:49	53.425 sec	53108 obs/sec	8.01904	28
12	2020-05-04 18:29:55	58.641 sec	53557 obs/sec	8.87798	31
13	2020-05-04 18:30:00	1 min 3.940 sec	53871 obs/sec	9.73847	34
14	2020-05-04 18:30:02	1 min 5.731 sec	53977 obs/sec	10.02476	35

	samples	training_rmse	training_deviance	training_mae	training_r2
1	0.000000	NA	NA	NA	NA
2	99612.000000	1.18002	1.39244	0.77616	0.30914
3	399787.000000	0.97505	0.95073	0.62310	0.52829
4	699506.000000	1.36439	1.86156	0.94192	0.07638
5	999663.000000	0.95704	0.91592	0.58968	0.54556
6	1299908.000000	0.89532	0.80161	0.58880	0.60228
7	1599488.000000	0.88382	0.78114	0.54437	0.61243
8	1898979.000000	0.89695	0.80451	0.56051	0.60084
9	2198517.000000	0.98511	0.97044	0.66267	0.51852
10	2498307.000000	0.97752	0.95555	0.64355	0.52590
11	2798757.000000	0.95795	0.91766	0.63961	0.54470
12	3098540.000000	0.81299	0.66095	0.49803	0.67207
13	3398864.000000	0.79232	0.62777	0.50377	0.68853
14	3498783.000000	0.91413	0.83564	0.61010	0.58540

	validation_rmse	validation_deviance	validation_mae	validation_r2
1	NA	NA	NA	NA
2	1.19613	1.43073	0.77229	0.27235
3	0.96037	0.92232	0.59816	0.53092
4	1.29986	1.68964	0.92981	0.14068
5	0.85455	0.73025	0.53609	0.62861
6	0.88114	0.77641	0.56149	0.60513
7	0.88666	0.78616	0.54143	0.60017
8	0.81185	0.65910	0.51854	0.66479
9	0.89660	0.80389	0.62753	0.59116
10	0.91244	0.83254	0.60811	0.57658
11	0.90381	0.81688	0.61981	0.58455
12	0.78116	0.61021	0.47946	0.68966
13	0.74827	0.55990	0.46983	0.71524
14	0.84261	0.71000	0.57596	0.63891

```
Variable Importances: (Extract with `h2o.varimp`)
=================================================

Variable Importances:
                            variable relative_importance scaled_importance
1  Horizontal_Distance_To_Fire_Points            1.000000          1.000000
2                           Elevation            0.929982          0.929982
3     Horizontal_Distance_To_Roadways            0.879362          0.879362
4                     Wilderness_Area1            0.816690          0.816690
5                     Wilderness_Area3            0.603950          0.603950
6                     Wilderness_Area2            0.544202          0.544202
7                     Wilderness_Area4            0.505349          0.505349
8    Horizontal_Distance_To_Hydrology             0.502873          0.502873
9                       Hillshade_Noon            0.431744          0.431744
10                       Hillshade_9am            0.426906          0.426906
11     Vertical_Distance_To_Hydrology             0.368905          0.368905
12                              Slope             0.363683          0.363683
13                       Hillshade_3pm            0.333631          0.333631
14                              Aspect            0.285971          0.285971
   percentage
1    0.125106
2    0.116346
3    0.110013
4    0.102173
5    0.075558
6    0.068083
7    0.063222
8    0.062912
9    0.054014
10   0.053408
11   0.046152
12   0.045499
13   0.041739
14   0.035777
```

Comparemos el error de entrenamiento con los errores de validación y prueba

```
> h2o.performance(modelo3, train=T)  ## sampled training data (del modelo)

H2ORegressionMetrics: deeplearning
** Reported on training data. **
** Metrics reported on temporary training frame with 10002 samples **

MSE:  0.8356351
RMSE:  0.9141308
MAE:  0.6101008
RMSLE:  0.2734858
Mean Residual Deviance :  0.8356351

> h2o.performance(modelo3, valid=T)  ## sampled validation data (del modelo)

H2ORegressionMetrics: deeplearning
** Reported on validation data. **
** Metrics reported on temporary validation frame with 996 samples **
```

```
MSE:  0.7099983
RMSE:  0.842614
MAE:  0.5759556
RMSLE:  0.2526945
Mean Residual Deviance :  0.7099983
```

```
> h2o.performance(modelo3, newdata=train) ## completo training data
```

```
H2ORegressionMetrics: deeplearning

MSE:  0.8232329
RMSE:  0.9073218
MAE:  0.6111452
RMSLE:  NaN
Mean Residual Deviance :  0.8232329
```

```
> h2o.performance(modelo3, newdata=valid) ## completo validation data
```

```
H2ORegressionMetrics: deeplearning

MSE:  0.8287249
RMSE:  0.9103433
MAE:  0.6133667
RMSLE:  NaN
Mean Residual Deviance :  0.8287249
```

```
> h2o.performance(modelo3, newdata=test)  ## completo test data
```

```
H2ORegressionMetrics: deeplearning

MSE:  0.8261798
RMSE:  0.9089444
MAE:  0.6114966
RMSLE:  NaN
Mean Residual Deviance :  0.8261798
```

Para confirmar que la matriz de confusión en el conjunto de validación (aquí, el conjunto de prueba) era correcta, hacemos una predicción en el conjunto de prueba y comparamos las matrices de confusión explícitamente:

```
> pred <- h2o.predict(modelo3, test)
```

```
  |                                                                              |
```

```
> pred
```

```
   predict
1 2.208862
2 2.302197
3 3.163374
4 3.026346
5 3.098466
6 3.178703

[115979 rows x 1 column]
```

```
> test$Accuracy <- pred$predict == test$Cover_Type
> 1-mean(test$Accuracy)
```

```
[1] 1
```

Importancia de los predictores

Dado que hay muchos parámetros que pueden afectar la precisión del modelo, el ajuste de hiperparámetros es especialmente importante para Deep Learning. Para la velocidad, solo entrenaremos en las primeras 10,000 filas del conjunto de datos de entrenamiento:

```
> sampled_train=train[1:10000,]
>
> # El método de búsqueda de hiperparámetro más simple es una exploración
> # de fuerza bruta del producto cartesiano completo de todas las
> # combinaciones especificadas por una búsqueda de cuadrícula:
>
> hyper_params <- list(
+   hidden=list(c(32,32,32),c(64,64)),
+   input_dropout_ratio=c(0,0.05),
+   rate=c(0.01,0.02),
+   rate_annealing=c(1e-8,1e-7,1e-6)
+ )
> hyper_params
```

```
$hidden
$hidden[[1]]
[1] 32 32 32

$hidden[[2]]
[1] 64 64


$input_dropout_ratio
[1] 0.00 0.05

$rate
[1] 0.01 0.02

$rate_annealing
[1] 1e-08 1e-07 1e-06
```

```
> grid <- h2o.grid(
+   algorithm="deeplearning",
+   grid_id="dl_grid",
+   training_frame=sampled_train,
+   validation_frame=valid,
+   x=predictors,
+   y=response,
+   epochs=10,
+   stopping_metric="MSE",
+   stopping_tolerance=1e-2, ## para cuando MSE no mejora >=1% para dos eventos
+   stopping_rounds=2,
+   score_validation_samples=100000, ## conjunto de validación
+   score_duty_cycle=0.025, ## no anote más del 2.5% del tiempo
+   adaptive_rate=F,        ## tasa de aprendizaje ajustada manualmente
+   momentum_start=0.5,     ## impulso sintonizado manualmente
+   momentum_stable=0.9,
```

```
+   momentum_ramp=1e7,
+   l1=1e-5,
+   l2=1e-5,
+   activation=c("Rectifier"),
+   max_w2=10,
+   hyper_params=hyper_params
+ )
```

```
  |                                                                              |
```

```
> grid
```

```
H2O Grid Details
================

Grid ID: dl_grid
Used hyper parameters:
  -  hidden
  -  input_dropout_ratio
  -  rate
  -  rate_annealing
Number of models: 23
Number of failed models: 1

Hyper-Parameter Search Summary: ordered by increasing residual_deviance
        hidden input_dropout_ratio rate rate_annealing        model_ids
1     [64, 64]                0.05 0.01         1.0E-7 dl_grid_model_12
2 [32, 32, 32]                 0.0 0.01         1.0E-7  dl_grid_model_9
3 [32, 32, 32]                 0.0 0.02         1.0E-8  dl_grid_model_5
4     [64, 64]                 0.0 0.02         1.0E-7 dl_grid_model_14
5 [32, 32, 32]                0.05 0.01         1.0E-6 dl_grid_model_19
   residual_deviance
1 1.9140617137551958
2 1.9468879027547097
3  1.987515506141201
4 1.9921774440955742
5 2.0292986297757185

---
         hidden input_dropout_ratio rate rate_annealing        model_ids
18     [64, 64]                0.05 0.02         1.0E-7 dl_grid_model_16
19     [64, 64]                 0.0 0.02         1.0E-6 dl_grid_model_22
20 [32, 32, 32]                 0.0 0.01         1.0E-6 dl_grid_model_17
21     [64, 64]                 0.0 0.02         1.0E-8  dl_grid_model_6
22 [32, 32, 32]                 0.0 0.02         1.0E-6 dl_grid_model_21
23 [32, 32, 32]                 0.0 0.02         1.0E-7 dl_grid_model_13
    residual_deviance
18  2.881340666741913
19   2.89249247928604
20 2.9437584937737102
21  3.564528036517894
22   4.86431076499405
23  5.390075952825526
Failed models
-------------
```

```
  hidden input_dropout_ratio rate rate_annealing status_failed
[64, 64]                0.05 0.02         1.0E-6          FAIL

 "DistributedException from localhost/127.0.0.1:54321: '\n\nTrying to predict with an unstable model.\n
```

Se muestran los modelos ordenados de mayor a menor AUC:

```
> resultados_grid <- h2o.getGrid(
+                       grid_id = "dl_grid",
+                       sort_by = "r2",
+                       decreasing = TRUE
+                     )
>
> data.frame(resultados_grid@summary_table) %>% select(-model_ids
+                                                      )
```

hidden	input_dropout_ratio	rate	rate_annealing	r2
[64, 64]	0.05	0.01	1.0E-7	0.02795495083117261
[32, 32, 32]	0.0	0.01	1.0E-7	0.00906561166327069
[64, 64]	0.0	0.02	1.0E-7	-0.020578501353580414
[32, 32, 32]	0.0	0.02	1.0E-8	-0.021555411701700944
[32, 32, 32]	0.05	0.01	1.0E-6	-0.03862687590041691
[64, 64]	0.0	0.01	1.0E-6	-0.07849444071469791
[64, 64]	0.0	0.01	1.0E-8	-0.12805239129693002
[32, 32, 32]	0.05	0.01	1.0E-7	-0.1544393496726706
[64, 64]	0.05	0.01	1.0E-8	-0.1560366751290625
[32, 32, 32]	0.05	0.02	1.0E-7	-0.16550408371247838
[64, 64]	0.0	0.01	1.0E-7	-0.17647405708352015
[32, 32, 32]	0.0	0.01	1.0E-8	-0.29767818261795753
[64, 64]	0.05	0.02	1.0E-8	-0.30448468246144555
[32, 32, 32]	0.05	0.01	1.0E-8	-0.3331097993271366
[32, 32, 32]	0.05	0.02	1.0E-8	-0.3584382532482129
[64, 64]	0.05	0.01	1.0E-6	-0.4344065489204363
[32, 32, 32]	0.05	0.02	1.0E-6	-0.46292044659419207
[64, 64]	0.05	0.02	1.0E-7	-0.4694605179831788
[64, 64]	0.0	0.02	1.0E-6	-0.47901140688605026
[32, 32, 32]	0.0	0.01	1.0E-6	-0.5065634808985269
[64, 64]	0.0	0.02	1.0E-8	-0.8229132888371764
[32, 32, 32]	0.0	0.02	1.0E-6	-1.4815712265250256
[32, 32, 32]	0.0	0.02	1.0E-7	-1.7510001052115558

El modelo que consigue mayor r2 de validación es el que tiene una arquitectura de tres capas con 32 neuronas. Veamos qué modelo tuvo el error de validación más bajo:

```
> grid <- h2o.getGrid("dl_grid",sort_by="r2",decreasing=FALSE)
> grid

H2O Grid Details
================

Grid ID: dl_grid
Used hyper parameters:
  -  hidden
  -  input_dropout_ratio
```

```
  -  rate
  -  rate_annealing
Number of models: 23
Number of failed models: 1

Hyper-Parameter Search Summary: ordered by increasing r2
        hidden input_dropout_ratio rate rate_annealing        model_ids
1 [32, 32, 32]                 0.0 0.02         1.0E-7 dl_grid_model_13
2 [32, 32, 32]                 0.0 0.02         1.0E-6 dl_grid_model_21
3     [64, 64]                 0.0 0.02         1.0E-8  dl_grid_model_6
4 [32, 32, 32]                 0.0 0.01         1.0E-6 dl_grid_model_17
5     [64, 64]                 0.0 0.02         1.0E-6 dl_grid_model_22
                    r2
1  -1.7510001052115558
2  -1.4815712265250256
3  -0.8229132888371764
4  -0.5065634808985269
5 -0.47901140688605026

---
         hidden input_dropout_ratio rate rate_annealing        model_ids
18     [64, 64]                 0.0 0.01         1.0E-6 dl_grid_model_18
19 [32, 32, 32]                0.05 0.01         1.0E-6 dl_grid_model_19
20 [32, 32, 32]                 0.0 0.02         1.0E-8  dl_grid_model_5
21     [64, 64]                 0.0 0.02         1.0E-7 dl_grid_model_14
22 [32, 32, 32]                 0.0 0.01         1.0E-7  dl_grid_model_9
23     [64, 64]                0.05 0.01         1.0E-7 dl_grid_model_12
                     r2
18  -0.07849444071469791
19  -0.03862687590041691
20 -0.021555411701700944
21 -0.020578501353580414
22   0.00906561166327069
23   0.02795495083117261
Failed models
-------------
   hidden input_dropout_ratio rate rate_annealing status_failed
 [64, 64]                0.05 0.02         1.0E-6          FAIL

 "DistributedException from localhost/127.0.0.1:54321: '\n\nTrying to predict with an unstable model.\n
> ## Para ver qué otros criterios "sort_by" están permitidos
> #grid <- h2o.getGrid("dl_grid",sort_by="wrong_thing",decreasing=FALSE)
>
> ## ordenar por r2
> h2o.getGrid("dl_grid",sort_by="r2",decreasing=FALSE)

H2O Grid Details
================

Grid ID: dl_grid
Used hyper parameters:
  -  hidden
  -  input_dropout_ratio
  -  rate
```

```
  -  rate_annealing
Number of models: 23
Number of failed models: 1

Hyper-Parameter Search Summary: ordered by increasing r2
        hidden input_dropout_ratio rate rate_annealing        model_ids
1 [32, 32, 32]                 0.0 0.02         1.0E-7 dl_grid_model_13
2 [32, 32, 32]                 0.0 0.02         1.0E-6 dl_grid_model_21
3     [64, 64]                 0.0 0.02         1.0E-8  dl_grid_model_6
4 [32, 32, 32]                 0.0 0.01         1.0E-6 dl_grid_model_17
5     [64, 64]                 0.0 0.02         1.0E-6 dl_grid_model_22
                    r2
1  -1.7510001052115558
2  -1.4815712265250256
3  -0.8229132888371764
4  -0.5065634808985269
5 -0.47901140688605026

---
         hidden input_dropout_ratio rate rate_annealing        model_ids
18     [64, 64]                 0.0 0.01         1.0E-6 dl_grid_model_18
19 [32, 32, 32]                0.05 0.01         1.0E-6 dl_grid_model_19
20 [32, 32, 32]                 0.0 0.02         1.0E-8  dl_grid_model_5
21     [64, 64]                 0.0 0.02         1.0E-7 dl_grid_model_14
22 [32, 32, 32]                 0.0 0.01         1.0E-7  dl_grid_model_9
23     [64, 64]                0.05 0.01         1.0E-7 dl_grid_model_12
                     r2
18  -0.07849444071469791
19  -0.03862687590041691
20 -0.021555411701700944
21 -0.020578501353580414
22   0.00906561166327069
23   0.02795495083117261
Failed models
-------------
   hidden input_dropout_ratio rate rate_annealing status_failed
 [64, 64]                0.05 0.02         1.0E-6          FAIL

 "DistributedException from localhost/127.0.0.1:54321: '\n\nTrying to predict with an unstable model.\n
```

```
> ## Find the best model and its full set of parameters
> grid@summary_table[1,]
```

hidden	input_dropout_ratio	rate	rate_annealing	model_ids	r2
[32, 32, 32]	0.0	0.02	1.0E-7	dl_grid_model_13	-1.7510001052115558

```
> best_model <- h2o.getModel(grid@model_ids[[1]])
> best_model

Model Details:
==============

H2ORegressionModel: deeplearning
```

```
Model ID:  dl_grid_model_13
Status of Neuron Layers: predicting Cover_Type, regression, gaussian distribution, Quadratic loss, 2.625
  layer units      type dropout       l1       l2 mean_rate rate_rms momentum
1     1    14     Input  0.00 %       NA       NA        NA       NA       NA
2     2    32 Rectifier  0.00 % 0.000010 0.000010  0.019802 0.000000 0.504000
3     3    32 Rectifier  0.00 % 0.000010 0.000010  0.019802 0.000000 0.504000
4     4    32 Rectifier  0.00 % 0.000010 0.000010  0.019802 0.000000 0.504000
5     5     1    Linear      NA 0.000010 0.000010  0.019802 0.000000 0.504000
  mean_weight weight_rms  mean_bias  bias_rms
1          NA         NA         NA        NA
2   -0.102501   0.788281 -13.823506 25.450706
3   -0.062271   0.457593   2.403866  3.538407
4   -0.150044   0.274997   0.623739  0.908314
5    0.120219   0.531865   0.050745  0.000000


H2ORegressionMetrics: deeplearning
** Reported on training data. **
** Metrics reported on full training frame **

MSE:  4.034905
RMSE:  2.008707
MAE:  1.757277
RMSLE:  0.4728226
Mean Residual Deviance :  4.034905


H2ORegressionMetrics: deeplearning
** Reported on validation data. **
** Metrics reported on temporary validation frame with 99863 samples **

MSE:  5.390076
RMSE:  2.321654
MAE:  2.234597
RMSLE:  0.6533468
Mean Residual Deviance :  5.390076
> print(best_model@allparameters)

$model_id
[1] "dl_grid_model_13"

$training_frame
[1] "RTMP_sid_8aa0_25"

$validation_frame
[1] "valid.hex"

$nfolds
[1] 0

$keep_cross_validation_models
[1] TRUE

$keep_cross_validation_predictions
```

```
[1] FALSE

$keep_cross_validation_fold_assignment
[1] FALSE

$fold_assignment
[1] "AUTO"

$ignore_const_cols
[1] TRUE

$score_each_iteration
[1] FALSE

$balance_classes
[1] FALSE

$max_after_balance_size
[1] 5

$max_confusion_matrix_size
[1] 20

$max_hit_ratio_k
[1] 0

$overwrite_with_best_model
[1] TRUE

$use_all_factor_levels
[1] TRUE

$standardize
[1] TRUE

$activation
[1] "Rectifier"

$hidden
[1] 32 32 32

$epochs
[1] 10

$train_samples_per_iteration
[1] -2

$target_ratio_comm_to_comp
[1] 0.05

$seed
[1] "-30818424322120721"

$adaptive_rate
```

```
[1] FALSE

$rho
[1] 0.99

$epsilon
[1] 1e-08

$rate
[1] 0.02

$rate_annealing
[1] 1e-07

$rate_decay
[1] 1

$momentum_start
[1] 0.5

$momentum_ramp
[1] 1e+07

$momentum_stable
[1] 0.9

$nesterov_accelerated_gradient
[1] TRUE

$input_dropout_ratio
[1] 0

$l1
[1] 1e-05

$l2
[1] 1e-05

$max_w2
[1] 10

$initial_weight_distribution
[1] "UniformAdaptive"

$initial_weight_scale
[1] 1

$loss
[1] "Automatic"

$distribution
[1] "AUTO"

$quantile_alpha
```

```
[1] 0.5

$tweedie_power
[1] 1.5

$huber_alpha
[1] 0.9

$score_interval
[1] 5

$score_training_samples
[1] 10000

$score_validation_samples
[1] 100000

$score_duty_cycle
[1] 0.025

$classification_stop
[1] 0

$regression_stop
[1] 1e-06

$stopping_rounds
[1] 2

$stopping_metric
[1] "MSE"

$stopping_tolerance
[1] 0.01

$max_runtime_secs
[1] 1.797693e+308

$score_validation_sampling
[1] "Uniform"

$diagnostics
[1] TRUE

$fast_mode
[1] TRUE

$force_load_balance
[1] TRUE

$variable_importances
[1] TRUE

$replicate_training_data
```

```
[1] TRUE

$single_node_mode
[1] FALSE

$shuffle_training_data
[1] FALSE

$missing_values_handling
[1] "MeanImputation"

$quiet_mode
[1] FALSE

$autoencoder
[1] FALSE

$sparse
[1] FALSE

$col_major
[1] FALSE

$average_activation
[1] 0

$sparsity_beta
[1] 0

$max_categorical_features
[1] 2147483647

$reproducible
[1] FALSE

$export_weights_and_biases
[1] FALSE

$mini_batch_size
[1] 1

$categorical_encoding
[1] "AUTO"

$elastic_averaging
[1] FALSE

$elastic_averaging_moving_rate
[1] 0.9

$elastic_averaging_regularization
[1] 0.001

$x
```

```
 [1] "Elevation"                         "Aspect"
 [3] "Slope"                             "Horizontal_Distance_To_Hydrology"
 [5] "Vertical_Distance_To_Hydrology"    "Horizontal_Distance_To_Roadways"
 [7] "Hillshade_9am"                     "Hillshade_Noon"
 [9] "Hillshade_3pm"                     "Horizontal_Distance_To_Fire_Points"
[11] "Wilderness_Area1"                  "Wilderness_Area2"
[13] "Wilderness_Area3"                  "Wilderness_Area4"

$y
[1] "Cover_Type"
```

```
> print(h2o.performance(best_model, valid=T))
```

```
H2ORegressionMetrics: deeplearning
** Reported on validation data. **
** Metrics reported on temporary validation frame with 99863 samples **

MSE:  5.390076
RMSE:  2.321654
MAE:  2.234597
RMSLE:  0.6533468
Mean Residual Deviance :  5.390076
```

```
> print(h2o.logloss(best_model, valid=T))
```

```
NULL
```

Una vez que estamos satisfechos con los resultados, podemos guardar el modelo en el disco (en el clúster). En este ejemplo, almacenamos el modelo en un directorio llamado mybest_deeplearning_covtype_model, que se creará para nosotros desde entonces force=TRUE.

```
> path <- h2o.saveModel(best_model,
+           path="./mybest_deeplearning_covtype_model", force=TRUE)
```

Se puede cargar más tarde con el siguiente comando:

```
> print(path)
> m_loaded <- h2o.loadModel(path)
> summary(m_loaded)
```

Este modelo es completamente funcional y puede inspeccionarse, reiniciarse o usarse para calificar un conjunto de datos, etc. Tenga en cuenta que la compatibilidad binaria entre las versiones H2O no está garantizada actualmente.

Resources

Mas informacion sobre machine learning con **H2O** y **R**

H2O

- Documentation for H2O and Sparkling Water: http://docs.h2o.ai/
- Glossary of terms: https://github.com/h2oai/h2o-3/blob/master/h2o-docs/src/product/tutorials/glossary.md
- Open forum for questions about H2O (Google account required): https://groups.google.com/forum/#!forum/h2ostream
- Track or file bug reports for H2O: https://jira.h2o.ai
- GitHub repository for H2O: https://github.com/h2oai

R

- About R: https://www.r-project.org/about.html
- Download R: https://cran.r-project.org/mirrors.html
- Latest R API H2O documentation: http://h2o-release.s3.amazonaws.com/h2o/latest_stable_Rdoc.html
- Tutorial H2O and R: http://docs.h2o.ai/h2o-tutorials/latest-stable/resources.html
- Machine Learning con H2O y R: https://rpubs.com/Joaquin_AR/406480

References

Aiello, Spencer, Eric Eckstrand, Anqi Fu, Mark Landry, and Patrick Aboyoun. 2016. "Machine Learning with R and H2o." *H2O Booklet*.

Candel, Arno, Viraj Parmar, Erin LeDell, and Anisha Arora. 2016. "Deep Learning with H2o." *H2O. Ai Inc.*