Evaluación MLII (temas 1, 2, 6 y 7): Ejercicio 2

Aprendizaje profundo

Inmaculada Perea Fernández

mayo 2017

Trabajando ahora con todas las letras, es decir, con los 20000 casos del data frame *LetterRecognition*, construir y evaluar un perceptrón multicapas basado en aprendizaje profundo para predecir la variable *lettr*

1 Librerías y servicio h2o

1.1. Carga e instalación de librerías necesarias

```
if (!require('mlbench')) install.packages('mlbench'); library('mlbench')
if (!require('h2o')) install.packages('h2o'); library('h2o')
```

1.2. Inicialización del servicio h2o

```
localH2O = h2o.init(max_mem_size = '3g',
                    ip = "127.0.0.1",    # localhost
                    port = 54321,
                    nthreads = -1)       # Configura número de hilos según recursos disponibles
```

2. Carga, inspección y preparación de los datos

2.1. Carga de los datos

```
data(LetterRecognition)
dim(LetterRecognition)
```

```
## [1] 20000    17
```

```
head(LetterRecognition)
```

```
##   lettr x.box y.box width high onpix x.bar y.bar x2bar y2bar xybar x2ybr
## 1     T     2     8     3    5     1     8    13     0     6     6    10
## 2     I     5    12     3    7     2    10     5     5     4    13     3
## 3     D     4    11     6    8     6    10     6     2     6    10     3
## 4     N     7    11     6    6     3     5     9     4     6     4     4
## 5     G     2     1     3    1     1     8     6     6     6     6     5
## 6     S     4    11     5    8     3     8     8     6     9     5     6
##   xy2br x.ege xegvy y.ege yegvx
## 1     8     0     8     0     8
## 2     9     2     8     4    10
## 3     7     3     7     3     9
## 4    10     6    10     2     8
```

```
## 5     9     1     7     5    10
## 6     6     0     8     9     7
```

```
str(LetterRecognition)
```

```
## 'data.frame':    20000 obs. of  17 variables:
##  $ lettr: Factor w/ 26 levels "A","B","C","D",..: 20 9 4 14 7 19 2 1 10 13 ...
##  $ x.box: num  2 5 4 7 2 4 4 1 2 11 ...
##  $ y.box: num  8 12 11 11 1 11 2 1 2 15 ...
##  $ width: num  3 3 6 6 3 5 5 3 4 13 ...
##  $ high : num  5 7 8 6 1 8 4 2 4 9 ...
##  $ onpix: num  1 2 6 3 1 3 4 1 2 7 ...
##  $ x.bar: num  8 10 10 5 8 8 8 8 10 13 ...
##  $ y.bar: num  13 5 6 9 6 8 7 2 6 2 ...
##  $ x2bar: num  0 5 2 4 6 6 6 2 2 6 ...
##  $ y2bar: num  6 4 6 6 6 9 6 2 6 2 ...
##  $ xybar: num  6 13 10 4 6 5 7 8 12 12 ...
##  $ x2ybr: num  10 3 3 4 5 6 6 2 4 1 ...
##  $ xy2br: num  8 9 7 10 9 6 6 8 8 9 ...
##  $ x.ege: num  0 2 3 6 1 0 2 1 1 8 ...
##  $ xegvy: num  8 8 7 10 7 8 8 6 6 1 ...
##  $ y.ege: num  0 4 3 2 5 9 7 2 1 1 ...
##  $ yegvx: num  8 10 9 8 10 7 10 7 7 8 ...
```

```
table(LetterRecognition$lettr)
```

```
##
##   A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R
## 789 766 736 805 768 775 773 734 755 747 739 761 792 783 753 803 783 758
##   S   T   U   V   W   X   Y   Z
## 748 796 813 764 752 787 786 734
```

2.2. División entrenamiento y test

Destinamos un 70% de los datos a entrenamiento y un 30% para test

```
set.seed(271)
n=nrow(LetterRecognition)
train.index=sort(sample(1:n, ceiling(0.7*n)))
train.data=LetterRecognition[train.index,]
test.data=LetterRecognition[-train.index,]
```

Conjunto de entrenamiento

```
dim(train.data)
```

```
## [1] 14000    17
```

```
summary(train.data)
```

```
##      lettr          x.box           y.box           width
##  T      :  582   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
##  D      :  571   1st Qu.: 3.000   1st Qu.: 5.000   1st Qu.: 4.000
##  P      :  566   Median : 4.000   Median : 7.000   Median : 5.000
##  M      :  565   Mean   : 4.017   Mean   : 7.032   Mean   : 5.114
```

```
##  X      :  562   3rd Qu.: 5.000   3rd Qu.: 9.000   3rd Qu.: 6.000
##  A      :  557   Max.   :15.000   Max.   :15.000   Max.   :15.000
##  (Other):10597
##       high            onpix            x.bar           y.bar
##  Min.   : 0.000   Min.   : 0.000   Min.   : 0.00   Min.   : 0.000
##  1st Qu.: 4.000   1st Qu.: 2.000   1st Qu.: 6.00   1st Qu.: 6.000
##  Median : 6.000   Median : 3.000   Median : 7.00   Median : 7.000
##  Mean   : 5.372   Mean   : 3.499   Mean   : 6.91   Mean   : 7.484
##  3rd Qu.: 7.000   3rd Qu.: 5.000   3rd Qu.: 8.00   3rd Qu.: 8.000
##  Max.   :15.000   Max.   :15.000   Max.   :15.00   Max.   :15.000
##
##      x2bar            y2bar            xybar           x2ybr
##  Min.   : 0.000   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 3.000   1st Qu.: 4.000   1st Qu.: 7.000   1st Qu.: 5.000
##  Median : 4.000   Median : 5.000   Median : 8.000   Median : 6.000
##  Mean   : 4.623   Mean   : 5.194   Mean   : 8.279   Mean   : 6.452
##  3rd Qu.: 6.000   3rd Qu.: 7.000   3rd Qu.:10.000   3rd Qu.: 8.000
##  Max.   :15.000   Max.   :15.000   Max.   :15.000   Max.   :15.000
##
##      xy2br            x.ege            xegvy           y.ege
##  Min.   : 0.000   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 7.000   1st Qu.: 1.000   1st Qu.: 8.000   1st Qu.: 2.000
##  Median : 8.000   Median : 3.000   Median : 8.000   Median : 3.000
##  Mean   : 7.914   Mean   : 3.038   Mean   : 8.339   Mean   : 3.692
##  3rd Qu.: 9.000   3rd Qu.: 4.000   3rd Qu.: 9.000   3rd Qu.: 5.000
##  Max.   :15.000   Max.   :15.000   Max.   :15.000   Max.   :15.000
##
##      yegvx
##  Min.   : 0.000
##  1st Qu.: 7.000
##  Median : 8.000
##  Mean   : 7.805
##  3rd Qu.: 9.000
##  Max.   :15.000
##
```

Conjunto de test

```
dim(test.data)
```

```
## [1] 6000   17
```

```
summary(test.data)
```

```
##      lettr          x.box            y.box            width
##  G      : 257   Min.   : 0.00   Min.   : 0.000   Min.   : 0.00
##  U      : 256   1st Qu.: 3.00   1st Qu.: 5.000   1st Qu.: 4.00
##  E      : 255   Median : 4.00   Median : 7.000   Median : 5.00
##  N      : 253   Mean   : 4.04   Mean   : 7.043   Mean   : 5.14
##  W      : 249   3rd Qu.: 5.00   3rd Qu.: 9.000   3rd Qu.: 6.00
##  S      : 245   Max.   :15.00   Max.   :15.000   Max.   :15.00
##  (Other):4485
##       high            onpix            x.bar           y.bar
##  Min.   : 0.000   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
```

```
##  1st Qu.: 4.000   1st Qu.: 2.000   1st Qu.: 6.000   1st Qu.: 6.000
##  Median : 6.000   Median : 3.000   Median : 7.000   Median : 7.000
##  Mean   : 5.374   Mean   : 3.522   Mean   : 6.869   Mean   : 7.538
##  3rd Qu.: 7.000   3rd Qu.: 5.000   3rd Qu.: 8.000   3rd Qu.: 9.000
##  Max.   :15.000   Max.   :15.000   Max.   :15.000   Max.   :15.000
##
##      x2bar            y2bar            xybar           x2ybr
##  Min.   : 0.000   Min.   : 0.000   Min.   : 0.00   Min.   : 0.00
##  1st Qu.: 3.000   1st Qu.: 3.000   1st Qu.: 7.00   1st Qu.: 5.00
##  Median : 4.000   Median : 5.000   Median : 8.00   Median : 6.00
##  Mean   : 4.643   Mean   : 5.143   Mean   : 8.29   Mean   : 6.46
##  3rd Qu.: 6.000   3rd Qu.: 7.000   3rd Qu.:10.00   3rd Qu.: 8.00
##  Max.   :15.000   Max.   :15.000   Max.   :15.00   Max.   :15.00
##
##      xy2br            x.ege            xegvy            y.ege
##  Min.   : 1.000   Min.   : 0.000   Min.   : 1.000   Min.   : 0.00
##  1st Qu.: 7.000   1st Qu.: 1.750   1st Qu.: 8.000   1st Qu.: 2.00
##  Median : 8.000   Median : 3.000   Median : 8.000   Median : 3.00
##  Mean   : 7.964   Mean   : 3.066   Mean   : 8.338   Mean   : 3.69
##  3rd Qu.: 9.000   3rd Qu.: 4.000   3rd Qu.: 9.000   3rd Qu.: 5.00
##  Max.   :15.000   Max.   :15.000   Max.   :15.000   Max.   :15.00
##
##      yegvx
##  Min.   : 0.000
##  1st Qu.: 7.000
##  Median : 8.000
##  Mean   : 7.793
##  3rd Qu.: 8.000
##  Max.   :14.000
##
```

2.3. Conversión de los datos al formato h2o

Con las siguientes instrucciones convertiremos los data frames en objetos que pueden ser procesados en el entorno paralelizado de cálculo con h2o.

```
train.hex <- as.h2o(train.data)
test.hex<- as.h2o(test.data)
```

3. Búsqueda exhaustiva de hiper-parámetros

Utilizaremos la librería *h2o* para construir un modelo basado en aprendizaje profundo pero antes realizaremos una búsqueda en el espacio de parámetros para encontrar los parámetros que ofrezcan mejor rendiemiento del modelo.

Utilizaremos la función *grid* de la librería *h2o* para explorar varias combinaciones de tamaños de la red y del parámetro de regularización L1. La función *grid* calcula además el error de clasificación de cada modelo mediante validación cruzada.

3.1 Búsqueda cartesiana

En la búsqueda cartesiana introducimos 3 posibles configuraciones de red y 2 posibles valores de parámetros de penalización L1. Son los que se muestran a continuación.

Con la función *grid* se entrenará el producto cartesiano de ambos conjuntos de parámetros, es decir, 3x2=6 modelos en total.

Definición de hiper-parámetros

```
# Tamaños de capa oculta
hidden_search = list(c(100, 100),    # 2 capas, cada una de 100 nodos
                     c(50, 50, 50),  # 3 capas, cada una de 50 nodos
                     c(200, 200))    # 2 capas, cada una de 200 nodos

# Parámetro de penalización L1
l1_search = c(1e-4, 1e-3)


hyper_params= list(hidden = hidden_search,
                   l1 = l1_search)
```

Definición del criterio de búsqueda

```
search_criteria = list(strategy = "Cartesian")
```

Búsqueda cartesiana

```
search_h2o_grid.cartesian = h2o.grid("deeplearning",
                             x = 2:17,
                             y = 1,
                             training_frame = train.hex,
                             validation_frame = test.hex,
                             distribution ="multinomial",
                             activation = 'RectifierWithDropout',
                             hyper_params = hyper_params,
                             nfolds = 5,
                             score_interval = 2,
                             epochs = 50,
                             stopping_rounds = 3,
                             stopping_tolerance = 0.05,
                             stopping_metric = "misclassification",
                             search_criteria = search_criteria)
```

Carga de resultados en fichero

Este proceso lleva varios minutos. Para evitar repetir la búsqueda en futuras ejecuciones se va a cargar en el fichero *search_h2o_grid.cartesian.RData* el resultado de la búsqueda anterior, junto con el conjunto de test y entrenamiento. De este modo es posible cargar el resultado directemente sin tener que esperar a que finalice la búsqueda.

```
save(train.data,
     test.data,
     search_h2o_grid.cartesian,
     file="search_h2o_grid.cartesian.RData")
```

Con las siguiente instrucción se puede cargar el fichero *search_h2o_grid.cartesian.RData* que contiene los datos

```
load(file="search_h2o_grid.cartesian.RData")
search_h2o_grid.cartesian
```

```
## H2O Grid Details
## ================
##
## Grid ID: Grid_DeepLearning_train.data_model_R_1496261201550_1
## Used hyper parameters:
##   -  hidden
##   -  l1
## Number of models: 6
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by increasing logloss
##         hidden     l1
## 1   [200, 200] 1.0E-4
## 2   [200, 200]  0.001
## 3   [100, 100] 1.0E-4
## 4   [100, 100]  0.001
## 5 [50, 50, 50]  0.001
## 6 [50, 50, 50] 1.0E-4
##                                                    model_ids
## 1 Grid_DeepLearning_train.data_model_R_1496261201550_1_model_2
## 2 Grid_DeepLearning_train.data_model_R_1496261201550_1_model_5
## 3 Grid_DeepLearning_train.data_model_R_1496261201550_1_model_0
## 4 Grid_DeepLearning_train.data_model_R_1496261201550_1_model_3
## 5 Grid_DeepLearning_train.data_model_R_1496261201550_1_model_4
## 6 Grid_DeepLearning_train.data_model_R_1496261201550_1_model_1
##               logloss
## 1 0.31522287924547626
## 2   0.393258236108136
## 3  0.5062267382884097
## 4  0.5579606459206349
## 5  1.3557581562374585
## 6  1.4517334176001975
```

Evaluación de los modelos obtenidos mediante búsqueda cartesiana

```
nmodelos.cartesian=length(search_h2o_grid.cartesian@model_ids)
Error.cartesian=numeric(nmodelos.cartesian)


for (i in 1:nmodelos.cartesian)
  {
   model_id= search_h2o_grid.cartesian@model_ids[[i]]
```

```
   entropia <- h2o.logloss(h2o.getModel(model_id), xval = TRUE)
   Error.cartesian[i]=entropia
   print(sprintf("Enropia VC (cartesian search): %f", entropia))
}
```

```
## [1] "Enropia VC (cartesian search): 0.315223"
## [1] "Enropia VC (cartesian search): 0.393258"
## [1] "Enropia VC (cartesian search): 0.506227"
## [1] "Enropia VC (cartesian search): 0.557961"
## [1] "Enropia VC (cartesian search): 1.355758"
## [1] "Enropia VC (cartesian search): 1.451733"
```

```
which.min(Error.cartesian)
```

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

```
search.cartesian.model=search_h2o_grid.cartesian@model_ids[[which.min(which.min(Error.cartesian))]]
```

El mejor modelo obtenido con las búsqueda es el que está compuesto de 2 capas ocultas, cada una de 200 nodos y el valor *L1* es 1.0E-4. Este modelo es el que presenta menor valor de entropía (0.3129)

3.2 Búsqueda aleatoria

Definición de hiper-parámetros

```
# Tamaños de capa oculta
hidden_search = lapply(1:100, function(x)10+sample(50,sample(4), replace=TRUE))

# Parámetro de penalización L1
l1_search = seq(1e-6, 1e-3, 1e-6)


hyper_params= list(hidden = hidden_search,
                   l1 = l1_search)
```

Definición del criterio de búsqueda

Indicamos que realice búsquedas con todas las combinaciones de los hiperparámetros que se dan como entrada en *hyper_params*. Indicamos el criterio de parada *max_model*, para que pare cuando evalúe 30 modelos.

```
search_criteria = list(strategy = "RandomDiscrete",
                       max_models = 30)
```

Búsqueda aleatoria

```
search_h2o_grid.random = h2o.grid("deeplearning",
                            x = 2:17,
                            y = 1,
                            training_frame = train.hex,
                            validation_frame = test.hex,
                            distribution ="multinomial",
                            activation = 'RectifierWithDropout',
```

```
                          hyper_params = hyper_params,
                          nfolds = 5,
                          score_interval = 2,
                          epochs = 50,
                          stopping_rounds = 3,
                          stopping_tolerance = 0.05,
                          stopping_metric = "misclassification",
                          search_criteria = search_criteria)
```

Carga de resultados en fichero

Este proceso lleva varios minutos. Para evitar repetir la búsqueda en futuras ejecuciones se va a cargar en el fichero *search_h2o_grid.random.RData* el resultado de la búsqueda anterior, junto con el conjunto de test y entrenamiento. De este modo es posible cargar el resultado directamente sin tener que esperar a que finalice la búsqueda.

```
save(train.data,
     test.data,
     search_h2o_grid.random,
     file="search_h2o_grid.random.RData")
```

Con las siguiente instrucción se puede cargar el fichero *search_h2o_grid.random.RData* que contiene los datos

```
load(file="search_h2o_grid.random.RData")
search_h2o_grid.random
```

```
## H2O Grid Details
## ================
##
## Grid ID: Grid_DeepLearning_train.data_model_R_1496261201550_2
## Used hyper parameters:
##   -  hidden
##   -  l1
## Number of models: 30
## Number of failed models: 0
##
## Hyper-Parameter Search Summary: ordered by increasing logloss
##   hidden      l1
## 1   [59] 5.64E-4
## 2   [54] 7.06E-4
## 3   [54] 6.37E-4
## 4   [51] 5.83E-4
## 5   [48]  7.4E-5
##                                                          model_ids
## 1 Grid_DeepLearning_train.data_model_R_1496261201550_2_model_11
## 2 Grid_DeepLearning_train.data_model_R_1496261201550_2_model_21
## 3  Grid_DeepLearning_train.data_model_R_1496261201550_2_model_0
## 4 Grid_DeepLearning_train.data_model_R_1496261201550_2_model_10
## 5  Grid_DeepLearning_train.data_model_R_1496261201550_2_model_8
##              logloss
## 1  0.615680570140225
## 2 0.6611511604529999
## 3 0.6657267332618827
```

```
## 4 0.6753311504558169
## 5  0.681453906501827
##
## ---
##              hidden      l1
## 25 [34, 44, 22, 39]  8.6E-4
## 26 [35, 46, 40, 31]  3.3E-4
## 27 [26, 36, 25, 18] 2.53E-4
## 28 [47, 46, 49, 17]  9.6E-5
## 29 [37, 24, 46, 22] 6.62E-4
## 30 [26, 36, 25, 18] 4.63E-4
##                                                          model_ids
## 25  Grid_DeepLearning_train.data_model_R_1496261201550_2_model_4
## 26 Grid_DeepLearning_train.data_model_R_1496261201550_2_model_18
## 27 Grid_DeepLearning_train.data_model_R_1496261201550_2_model_19
## 28  Grid_DeepLearning_train.data_model_R_1496261201550_2_model_2
## 29  Grid_DeepLearning_train.data_model_R_1496261201550_2_model_5
## 30 Grid_DeepLearning_train.data_model_R_1496261201550_2_model_23
##               logloss
## 25 2.8889344702364923
## 26 2.9681600571343356
## 27  3.185548303633614
## 28 3.4381067762241324
## 29  3.451003415060574
## 30  3.826546342113976
```

Evaluación de los modelos obtenidos mediante búsqueda aleatoria

```
nmodelos.random=length(search_h2o_grid.random@model_ids)
Error.random=numeric(nmodelos.random)

for (i in 1:nmodelos.random)
  {
   model_id= search_h2o_grid.random@model_ids[[i]]
   entropia <- h2o.logloss(h2o.getModel(model_id), xval = TRUE)
   Error.random[i]=entropia
   print(sprintf("Enropia VC (random search): %f", entropia))
}
```

```
## [1] "Enropia VC (random search): 0.615681"
## [1] "Enropia VC (random search): 0.661151"
## [1] "Enropia VC (random search): 0.665727"
## [1] "Enropia VC (random search): 0.675331"
## [1] "Enropia VC (random search): 0.681454"
## [1] "Enropia VC (random search): 0.690360"
## [1] "Enropia VC (random search): 0.695727"
## [1] "Enropia VC (random search): 0.703517"
## [1] "Enropia VC (random search): 0.872494"
## [1] "Enropia VC (random search): 0.909974"
## [1] "Enropia VC (random search): 1.122941"
## [1] "Enropia VC (random search): 1.215549"
## [1] "Enropia VC (random search): 1.222064"
## [1] "Enropia VC (random search): 1.222078"
```

```
## [1] "Enropia VC (random search): 1.228145"
## [1] "Enropia VC (random search): 1.290247"
## [1] "Enropia VC (random search): 1.521158"
## [1] "Enropia VC (random search): 1.634747"
## [1] "Enropia VC (random search): 1.790680"
## [1] "Enropia VC (random search): 1.855634"
## [1] "Enropia VC (random search): 1.869950"
## [1] "Enropia VC (random search): 1.939410"
## [1] "Enropia VC (random search): 1.943405"
## [1] "Enropia VC (random search): 2.335389"
## [1] "Enropia VC (random search): 2.888934"
## [1] "Enropia VC (random search): 2.968160"
## [1] "Enropia VC (random search): 3.185548"
## [1] "Enropia VC (random search): 3.438107"
## [1] "Enropia VC (random search): 3.451003"
## [1] "Enropia VC (random search): 3.826546"
```

```
which.min(Error.random)
```

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

```
search.random.model=search_h2o_grid.random@model_ids[[which.min(which.min(Error.random))]]
```

En esta búsqueda obtenemos que el mejor modelo encontrado de los 30 evaluados es el que se compone de una capa oculta de 48 nodos y con un L1=5.26E-4. Con este modelo se obtiene una entropía igual a 0.6742. El modelo obtenido, aunque es más simple que el encontrado con búsqueda cartesiana, pero tiene una entropia mucho mayor. Por tanto el mejor modelo es el obtenido por búsqueda cartesiana.

4. Medida del rendimiento sobre el conjunto test

Nos quedamos con el mejor modelo obtenido de las 2 estrategias de búsqueda realizadas en los apartados 3.1 y 3.2, es decir, el modelo compuesto por 2 capas ocultas cada una con 200 nodos.

```
modelo=h2o.getModel(search.cartesian.model)
modelo
```

```
## Model Details:
## ==============
##
## H2OMultinomialModel: deeplearning
## Model ID:  Grid_DeepLearning_train.data_model_R_1496261201550_1_model_2
## Status of Neuron Layers: predicting lettr, 26-class classification, multinomial distribution, CrossE
##   layer units             type dropout       l1       l2 mean_rate
## 1     1    16            Input  0.00 %
## 2     2   200 RectifierDropout 50.00 % 0.000100 0.000000  0.001375
## 3     3   200 RectifierDropout 50.00 % 0.000100 0.000000  0.001779
## 4     4    26          Softmax         0.000100 0.000000  0.022501
##   rate_rms momentum mean_weight weight_rms mean_bias bias_rms
## 1
## 2 0.000409 0.000000   -0.002002   0.275716  0.018152 0.199172
## 3 0.000730 0.000000   -0.034230   0.119770  0.673834 0.355266
## 4 0.086777 0.000000   -0.449202   0.533695 -4.679379 0.743352
##
##
## H2OMultinomialMetrics: deeplearning
```

```
## ** Reported on training data. **
## ** Metrics reported on temporary training frame with 9899 samples **
##
## Training Set Metrics:
## =====================
##
## MSE: (Extract with `h2o.mse`) 0.07647203
## RMSE: (Extract with `h2o.rmse`) 0.2765358
## Logloss: (Extract with `h2o.logloss`) 0.2376684
## Mean Per-Class Error: 0.06844171
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
## ========================================================================
## Confusion Matrix: vertical: actual; across: predicted
##     A   B   C   D   E F  G H I J K L M N O P Q R S T U V W X Y Z  Error
## A 397   1   3   0   0 0  0 0 0 0 0 3 1 0 0 1 0 1 1 0 1 1 0 0 5 0 0.0434
## B   0 377   0   3   0 1  0 0 1 0 1 0 0 0 0 0 0 8 0 0 0 1 0 1 0 0 0.0407
## C   0   0 365   0   5 0  4 1 0 0 1 0 0 0 6 0 0 2 0 0 2 0 0 0 0 0 0.0544
## D   0   9   0 377   0 1  0 1 0 0 0 0 1 9 1 0 0 3 0 0 0 0 0 0 0 0 0.0622
## E   0   3   4   0 312 3 19 0 1 0 1 8 0 0 0 0 0 2 2 0 0 0 0 0 0 0 0.1211
##            Rate
## A =    18 / 415
## B =    16 / 393
## C =    21 / 386
## D =    25 / 402
## E =    43 / 355
##
## ---
##          A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q
## V        0   8   0   2   0   1   2   0   0   0   0   0   0   1   1   0   0
## W        0   1   0   0   0   0   0   0   0   0   0   0   5   1   4   0   1
## X        0   2   0   2   2   0   0   0   2   1   7   5   0   0   1   0   0
## Y        0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   1
## Z        2   0   0   0  10   0   0   0   0   2   0   1   0   0   0   0   3
## Totals 404 460 381 413 346 375 378 303 362 366 339 403 403 404 375 387 381
##          R   S   T   U   V   W   X   Y   Z  Error          Rate
## V        1   0   0   1 361   1   0   1   0 0.0500 =    19 / 380
## W        1   0   0   4   1 344   0   1   0 0.0523 =    19 / 363
## X        1   1   0   0   0   0 365   1   0 0.0641 =    25 / 390
## Y        0   2   3   3  11   2   1 370   0 0.0609 =    24 / 394
## Z        0  25   2   0   0   0   1   0 317 0.1267 =    46 / 363
## Totals 441 374 387 371 388 363 386 389 320 0.0678 = 671 / 9.899
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
## =======================================================================
## Top-10 Hit Ratios:
##     k hit_ratio
## 1   1  0.932215
## 2   2  0.972219
## 3   3  0.986463
## 4   4  0.991817
## 5   5  0.994848
## 6   6  0.996565
## 7   7  0.997777
## 8   8  0.998283
```

```
## 9   9  0.998990
## 10 10  0.999293
##
##
## H2OMultinomialMetrics: deeplearning
## ** Reported on validation data. **
## ** Metrics reported on full validation frame **
##
## Validation Set Metrics:
## =====================
##
## Extract validation frame with `h2o.getFrame("test.data")`
## MSE: (Extract with `h2o.mse`) 0.08978145
## RMSE: (Extract with `h2o.rmse`) 0.2996355
## Logloss: (Extract with `h2o.logloss`) 0.2903884
## Mean Per-Class Error: 0.08866263
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,valid = TRUE)`)
## =========================================================================
## Confusion Matrix: vertical: actual; across: predicted
##     A   B   C   D   E F  G H I J K L M N O P Q R S T U V W X Y Z  Error
## A 229   0   1   0   0 0  0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0.0129
## B   0 201   0   2   0 1  0 0 0 0 0 0 0 0 0 1 0 3 1 0 0 3 0 0 0 0 0.0519
## C   0   0 203   0   3 0  4 2 0 0 1 0 0 0 3 0 0 2 0 0 0 0 0 0 0 0 0.0688
## D   0  11   0 206   0 0  0 4 0 0 0 0 1 2 4 0 0 4 0 0 1 0 0 1 0 0 0.1197
## E   0   3   4   0 226 0 14 0 0 0 0 6 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0.1137
##            Rate
## A =     3 / 232
## B =    11 / 212
## C =    15 / 218
## D =    28 / 234
## E =    29 / 255
##
## ---
##          A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q
## V        0   7   0   0   0   0   1   1   0   0   0   0   0   2   3   2   0
## W        0   0   0   0   0   0   1   0   0   0   0   0   3   2   2   0   2
## X        0   1   0   2   1   0   0   0   2   1   6   6   0   0   1   0   0
## Y        0   0   0   1   0   1   0   0   0   0   0   0   0   0   0   2   1
## Z        1   0   0   1   6   0   0   0   0   3   0   2   0   0   0   0   5
## Totals 235 259 220 232 250 240 262 196 207 215 227 207 211 249 252 234 232
##          R   S   T   U   V   W   X   Y   Z  Error          Rate
## V        0   0   0   0 206   3   0   1   0 0.0885 =    20 / 226
## W        0   0   0   2   1 236   0   0   0 0.0522 =    13 / 249
## X        0   0   0   0   0   0 205   0   0 0.0889 =    20 / 225
## Y        0   3   2   1   8   0   1 214   0 0.0855 =    20 / 234
## Z        1  13   0   0   0   0   0   0 177 0.1531 =    32 / 209
## Totals 265 253 203 254 227 249 220 220 181 0.0885 = 531 / 6.000
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,valid = TRUE)`
## =======================================================================
## Top-10 Hit Ratios:
##     k hit_ratio
## 1   1  0.911500
## 2   2  0.961833
```

```
## 3   3  0.979167
## 4   4  0.988000
## 5   5  0.992500
## 6   6  0.994167
## 7   7  0.995500
## 8   8  0.996167
## 9   9  0.996833
## 10 10  0.997500
##
##
## H2OMultinomialMetrics: deeplearning
## ** 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("train.data")`
## MSE: (Extract with `h2o.mse`) 0.09949689
## RMSE: (Extract with `h2o.rmse`) 0.3154313
## Logloss: (Extract with `h2o.logloss`) 0.3152229
## Mean Per-Class Error: 0.09425808
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,xval = TRUE)`
## ================================================================================
## Top-10 Hit Ratios:
##     k hit_ratio
## 1   1  0.906429
## 2   2  0.956143
## 3   3  0.975000
## 4   4  0.984143
## 5   5  0.988643
## 6   6  0.992357
## 7   7  0.994143
## 8   8  0.995357
## 9   9  0.996429
## 10 10  0.997571
##
##
## Cross-Validation Metrics Summary:
##                              mean           sd  cv_1_valid  cv_2_valid
## accuracy                 0.906451  0.004436146   0.8958923   0.9124507
## err                   0.093549035  0.004436146  0.10410765 0.087549336
## err_count                   262.0    13.152946       294.0       244.0
## logloss                0.31517643  0.009127537  0.32989293  0.30610064
## max_per_class_error    0.22523531  0.038834948         0.3  0.16666667
## mean_per_class_accuracy  0.9060517  0.004817532   0.8939092   0.9117332
## mean_per_class_error   0.093948305  0.004817532 0.106090784 0.088266775
## mse                    0.09948296  0.002644734  0.10505057  0.09709546
## r2                      0.9982366 3.9704857E-5    0.998133  0.99827844
## rmse                   0.31535393 0.0041747335  0.32411507  0.31160146
##                        cv_3_valid  cv_4_valid  cv_5_valid
## accuracy                0.9070018  0.91298145   0.9039286
## err                    0.09299821  0.08701854  0.09607143
## err_count                   259.0       244.0       269.0
```

```
## logloss                  0.30109027  0.30714983  0.33164844
## max_per_class_error      0.27586207  0.16666667  0.21698113
## mean_per_class_accuracy    0.907342  0.91295755   0.9043165
## mean_per_class_error    0.092657976 0.087042466  0.09568351
## mse                     0.096504316 0.095892586 0.102871865
## r2                        0.9982647   0.9982845   0.9982223
## rmse                     0.31065145   0.3096653  0.32073644
```

4.1 Cálculo de las predicciones sobre conjunto test

```
predic_test <- h2o.predict(modelo, newdata = test.hex)
pred <- as.data.frame(predic_test)
```

4.2 Tabla de confusión

```
tabla=table(test.data[,1],pred[,1])
tabla
```

```
##
##       A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q
##   A 229   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##   B   0 201   0   2   0   1   0   0   0   0   0   0   0   0   0   1   0
##   C   0   0 203   0   3   0   4   2   0   0   1   0   0   0   3   0   0
##   D   0  11   0 206   0   0   0   4   0   0   0   0   1   2   4   0   0
##   E   0   3   4   0 226   0  14   0   0   0   0   6   0   0   0   0   0
##   F   0   1   0   2   3 215   4   1   2   0   0   0   0   1   1   5   0
##   G   0   0   8   0   1   1 223   1   0   0   0   0   1   0   4   5   3
##   H   2   7   1   4   0   0   2 175   0   0   7   0   0   1   1   0   1
##   I   0   0   1   3   0   5   0   0 197   7   0   0   0   0   0   1   0
##   J   0   0   0   1   1   1   0   1   6 204   0   0   0   0   2   0   0
##   K   0   4   0   0   0   0   0   0   0   0 210   1   0   0   0   0   0
##   L   0   3   0   0   5   0   2   1   0   0   1 191   0   0   0   0   1
##   M   0   7   0   0   0   0   1   0   0   0   1   0 206   5   3   0   0
##   N   1   2   0   1   0   0   0   4   0   0   0   0   0 234   2   0   0
##   O   0   0   2   5   0   0   1   0   0   0   0   0   0   0 218   1   2
##   P   0   1   0   1   0  10   1   1   0   0   0   0   0   0   0 217   1
##   Q   2   0   0   0   1   0   4   0   0   0   0   0   0   0   6   0 216
##   R   0   3   0   1   0   0   3   2   0   0   1   0   0   2   0   0   0
##   S   0   4   0   2   3   6   0   0   0   0   0   1   0   0   0   0   0
##   T   0   3   0   0   0   0   1   3   0   0   0   0   0   0   0   0   0
##   U   0   1   0   0   0   0   0   0   0   0   0   0   0   0   2   0   0
##   V   0   7   0   0   0   0   1   1   0   0   0   0   0   2   3   2   0
##   W   0   0   0   0   0   0   1   0   0   0   0   0   3   2   2   0   2
##   X   0   1   0   2   1   0   0   0   2   1   6   6   0   0   1   0   0
##   Y   0   0   0   1   0   1   0   0   0   0   0   0   0   0   0   2   1
##   Z   1   0   0   1   6   0   0   0   0   3   0   2   0   0   0   0   5
##
##       R   S   T   U   V   W   X   Y   Z
##   A   0   1   0   1   0   0   0   0   0
##   B   3   1   0   0   3   0   0   0   0
##   C   2   0   0   0   0   0   0   0   0
```

```
##   D   4   0   0   1   0   0   1   0   0
##   E   1   0   0   0   0   0   1   0   0
##   F   0   2   5   0   0   0   0   0   0
##   G   1   2   0   0   5   2   0   0   0
##   H  21   0   0   1   0   0   1   0   0
##   I   0   2   0   0   0   0   3   0   1
##   J   0   0   0   0   0   0   1   0   0
##   K  16   1   0   0   0   0   3   0   0
##   L   1   1   0   0   0   0   3   0   0
##   M   0   0   0   0   1   3   0   0   0
##   N   9   0   0   0   0   0   0   0   0
##   O   2   0   0   0   0   2   0   0   0
##   P   0   0   0   0   0   0   0   5   0
##   Q   0   0   0   0   0   0   0   0   1
##   R 194   0   0   0   0   0   1   0   0
##   S   1 224   2   0   0   0   0   0   2
##   T   9   3 194   0   1   0   0   0   0
##   U   0   0   0 248   2   3   0   0   0
##   V   0   0   0   0 206   3   0   1   0
##   W   0   0   0   2   1 236   0   0   0
##   X   0   0   0   0   0   0 205   0   0
##   Y   0   3   2   1   8   0   1 214   0
##   Z   1  13   0   0   0   0   0   0 177
```

4.3 Porcentaje de acierto total

```
aciertos=100*diag(prop.table(tabla,1))
acierto=100*sum(diag(tabla))/sum(tabla)
round(acierto, 3)
```

```
## [1] 91.15
```

Obtenemos un porcentaje de acierto bastante elevado, por lo que el modelo obtenido es bastante satisfactorio para el conjunto de datos evaludado.

4.4 Porcentaje de acierto por categoría

```
cbind(tabla, Acierto_test=round(aciertos, 3))
```

```
##     A   B   C   D   E   F   G   H   I   J   K   L   M   N   O   P   Q   R
## A 229   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## B   0 201   0   2   0   1   0   0   0   0   0   0   0   0   0   1   0   3
## C   0   0 203   0   3   0   4   2   0   0   1   0   0   0   3   0   0   2
## D   0  11   0 206   0   0   0   4   0   0   0   0   1   2   4   0   0   4
## E   0   3   4   0 226   0  14   0   0   0   0   6   0   0   0   0   0   1
## F   0   1   0   2   3 215   4   1   2   0   0   0   0   1   1   5   0   0
## G   0   0   8   0   1   1 223   1   0   0   0   0   1   0   4   5   3   1
## H   2   7   1   4   0   0   2 175   0   0   7   0   0   1   1   0   1  21
## I   0   0   1   3   0   5   0   0 197   7   0   0   0   0   0   1   0   0
## J   0   0   0   1   1   1   0   1   6 204   0   0   0   0   2   0   0   0
## K   0   4   0   0   0   0   0   0   0   0 210   1   0   0   0   0   0  16
## L   0   3   0   0   5   0   2   1   0   0   1 191   0   0   0   0   1   1
```

```
## M   0   7   0   0   0   0   1   0   0   0   1   0 206   5   3   0   0   0
## N   1   2   0   1   0   0   0   4   0   0   0   0   0 234   2   0   0   9
## O   0   0   2   5   0   0   1   0   0   0   0   0   0   0 218   1   2   2
## P   0   1   0   1   0  10   1   1   0   0   0   0   0   0   0 217   1   0
## Q   2   0   0   0   1   0   4   0   0   0   0   0   0   0   6   0 216   0
## R   0   3   0   1   0   0   3   2   0   0   1   0   0   2   0   0   0 194
## S   0   4   0   2   3   6   0   0   0   0   0   1   0   0   0   0   0   1
## T   0   3   0   0   0   0   1   3   0   0   0   0   0   0   0   0   0   9
## U   0   1   0   0   0   0   0   0   0   0   0   0   0   0   2   0   0   0
## V   0   7   0   0   0   0   1   1   0   0   0   0   0   2   3   2   0   0
## W   0   0   0   0   0   0   1   0   0   0   0   0   3   2   2   0   2   0
## X   0   1   0   2   1   0   0   0   2   1   6   6   0   0   1   0   0   0
## Y   0   0   0   1   0   1   0   0   0   0   0   0   0   0   0   2   1   0
## Z   1   0   0   1   6   0   0   0   0   3   0   2   0   0   0   0   5   1
##     S   T   U   V   W   X   Y   Z Acierto_test
## A   1   0   1   0   0   0   0   0       98.707
## B   1   0   0   3   0   0   0   0       94.811
## C   0   0   0   0   0   0   0   0       93.119
## D   0   0   1   0   0   1   0   0       88.034
## E   0   0   0   0   0   1   0   0       88.627
## F   2   5   0   0   0   0   0   0       88.843
## G   2   0   0   5   2   0   0   0       86.770
## H   0   0   1   0   0   1   0   0       78.125
## I   2   0   0   0   0   3   0   1       89.545
## J   0   0   0   0   0   1   0   0       94.009
## K   1   0   0   0   0   3   0   0       89.362
## L   1   0   0   0   0   3   0   0       91.388
## M   0   0   0   1   3   0   0   0       90.749
## N   0   0   0   0   0   0   0   0       92.490
## O   0   0   0   0   2   0   0   0       93.562
## P   0   0   0   0   0   0   5   0       91.561
## Q   0   0   0   0   0   0   0   1       93.913
## R   0   0   0   0   0   1   0   0       93.720
## S 224   2   0   0   0   0   0   2       91.429
## T   3 194   0   1   0   0   0   0       90.654
## U   0   0 248   2   3   0   0   0       96.875
## V   0   0   0 206   3   0   1   0       91.150
## W   0   0   2   1 236   0   0   0       94.779
## X   0   0   0   0   0 205   0   0       91.111
## Y   3   2   1   8   0   1 214   0       91.453
## Z  13   0   0   0   0   0   0 177       84.689
```

Todas las categorías tienen una tasa de acierto alta. La letra más difícil de reconocer es la letra *"H"*.