Entrega

Victoria García Vega; Miguel Ángel González Caminero; Pablo Pérez Martín

2023-12-14

Digit recognition

```
library (caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library(ranger)
library(nnet)
library(mlbench)
```

```
data <- read.csv("train.csv")
test_df <- read.csv("test.csv")

data$label <- as.factor(data$label)
summary(data$label)
```

```
##    0    1    2    3    4    5    6    7    8    9
## 4132 4684 4177 4351 4072 3795 4137 4401 4063 4188
```

```
dim(train)
```

```
## NULL
```

```
#View(test_df)
```

División conjunto test y train

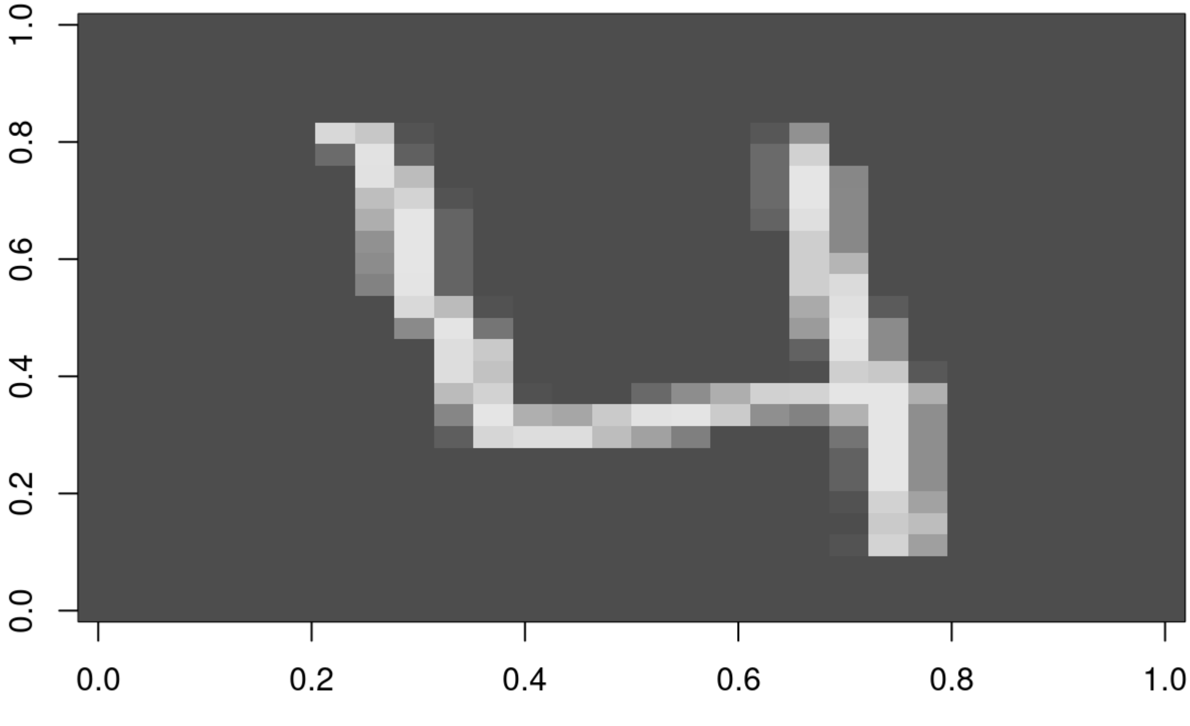
```
set.seed(33)

train_perc <- 0.75
train_index <- createDataPartition(data$label, p=train_perc, list=FALSE)

data_train <- data[train_index,]
data_test <- data[-train_index,]
```

Funciones de ayuda

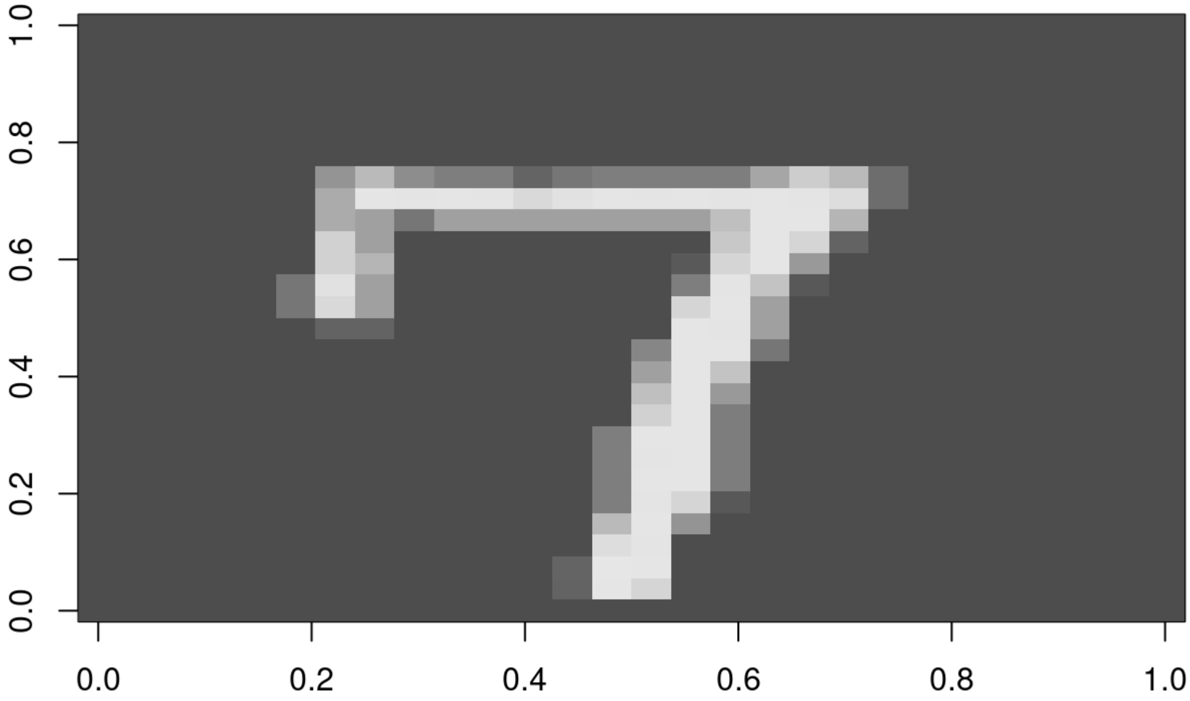
```
rotate <- function(x) t(apply(x, 2, rev))
```

Mostramos un par de imagenes de prueba

```
sample_4 <- matrix(as.numeric(data[4,-1]), nrow = 28, byrow = TRUE)
sample_7 <- matrix(as.numeric(data[7,-1]), nrow = 28, byrow = TRUE)
image(rotate(sample_4), col = grey.colors(255))
```



```
image(rotate(sample_7), col = grey.colors(255))
```



Random Forest

El primer algoritmo que vamos a utilizar es el de Random Forest, que consiste en construir múltiples árboles de decisión durante el entrenamiento y combinar sus resultados para obtener predicciones más precisas.

Para ello, vamos a utilizar la librería **ranger**, que es una versión optimizada de **randomForest**, con los siguientes parámetros:

- *num.trees=50*: El número de árboles que hemos indicado es 50, pues al incrementar este valor el tiempo de ejecución crece significativamente, mientras que el accuracity se muestra prácticamente invariable.
- *importance="impurity"*: este parámetro se utiliza para indicar si se deben calcular las importancias de las variables. Su valor por defecto es "none". Sin embargo, para nuestro modelo le hemos indicado que calcule la importancia de las variables basándose en la medida de impureza de Gini o ganancia de varianza.

```
if (!file.exists("random_forest.rds")) {

  # Entrenamos modelo
  rf <- ranger(label ~ ., data = data_train, num.trees = 50, importance = "impurity")

  # Guardamos modelo
  saveRDS(rf, file = "random_forest.rds")

} else {
  # cargamos modelo
  rf <- readRDS("random_forest.rds")
}

print(rf)
```

```
## Ranger result
##
## Call:
##  ranger(label ~ ., data = data_train, num.trees = 50, importance = "impurity")
##
## Type:                             Classification
## Number of trees:                  50
## Sample size:                      31503
## Number of independent variables:  784
## Mtry:                             28
## Target node size:                 1
## Variable importance mode:         impurity
## Splitrule:                        gini
## OOB prediction error:             5.02 %
```

```
# importance(rf)
```

```
prediction_rf<- predict(rf, data_test)

confussion_matrix_rf <- table(data_test$label, prediction_rf$predictions)
confussion_matrix_rf
```

```
##
##        0    1    2    3    4    5    6    7    8    9
##   0 1021    0    0    1    1    1    2    0    7    0
##   1    0 1150    7    5    2    2    1    3    1    0
##   2    6    2  997    3   12    0    4   12    7    1
##   3    5    1   11 1024    0   13    3    9   14    7
##   4    0    2    3    0  987    0    4    3    1   18
##   5    3    2    1   17    2  899   10    0    8    6
##   6    8    0    2    0    3    6 1010    0    5    0
##   7    0    4   11    0    7    0    0 1056    2   20
##   8    3    8    5   11    7    4    6    1  957   13
##   9    3    1    3   12   13    5    1    9    6  994
```

```
accuracy_rf <- mean(prediction_rf$predictions == data_test$label)
cat("Accuracy with random forest:", accuracy_rf) # 0.96
```

```
## Accuracy with random forest: 0.9617033
```

Boosting

A continuación, usaremos tres algoritmos distintos de *boosting*: C5.0, AdaBoost.M1 y Boosted Linear Model.

Usaremos para ello la librería *caret* junto con las librerías *adabag*, *plyr*, *bst* y *C50*.

Entrenamiento

```
control <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
set.seed(Sys.time())
metric <- "Accuracy"
```

C5.0

```
grid <- expand.grid(trials = 20,
                    model = "tree",
                    winnow = FALSE)
if (file.exists("boosting_c50.rds")) {
  model_c50 <- readRDS(file = "boosting_c50.rds")
} else {
  model_c50 <- train(label ~ .,
                     data = data_train[1:1000, ],
                     method = "C5.0",
                     metric = metric,
                     tuneGrid = grid,
                     trControl = control)
  saveRDS(model_c50, file = "boosting_c50.rds")
}
```

AdaBoost.M1

```
grid <- expand.grid(mfinal = 20,
                    maxdepth = 5,
                    coeflearn = c("Breiman"))
if (file.exists("boosting_adaboost.rds")) {
  model_adaboost <- readRDS(file = "boosting_adaboost.rds")
} else {
  model_adaboost <- train(label ~ .,
                          data = data_train[1:1000, ],
                          method = "AdaBoost.M1",
                          metric = metric,
                          tuneGrid = grid,
                          trControl = control)
  saveRDS(model_adaboost, file = "boosting_adaboost.rds")
}
```

Boosted Linear Model

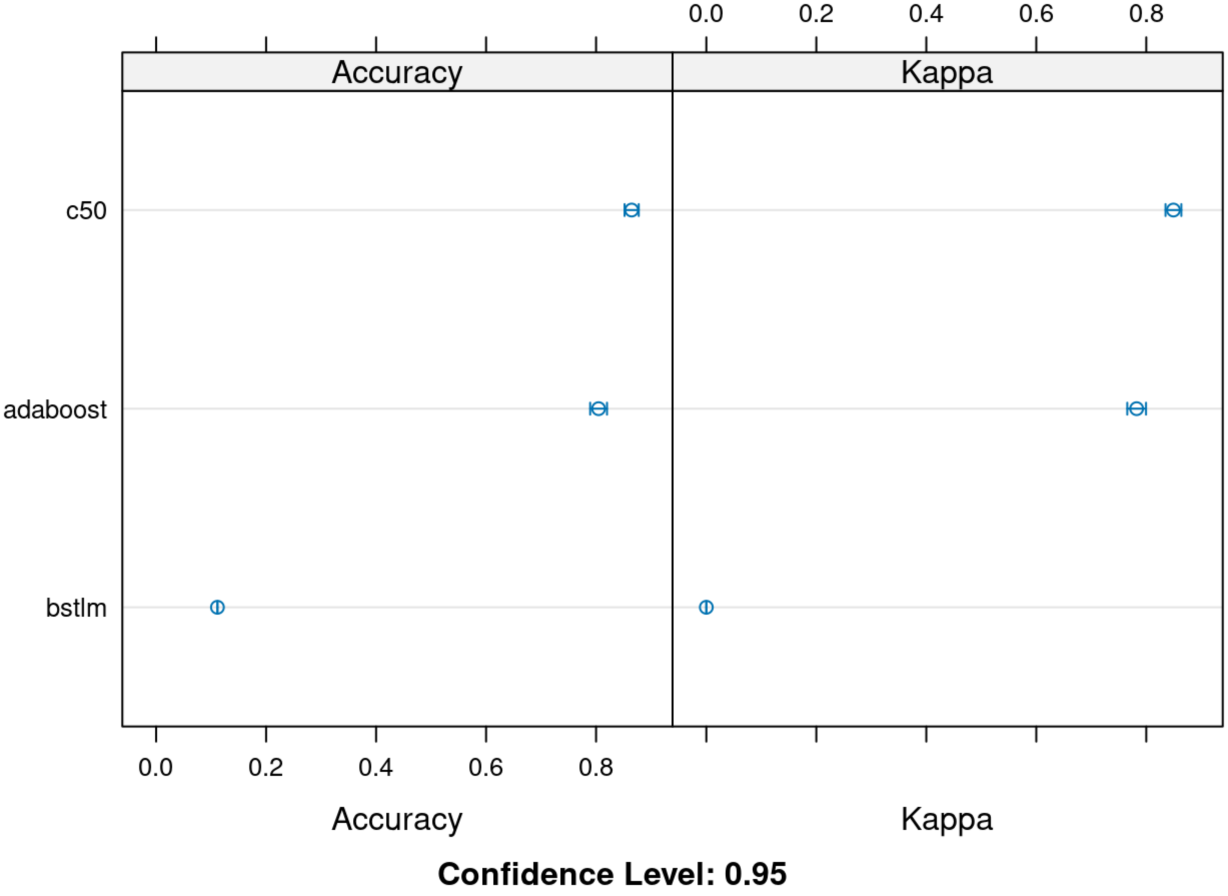
```
grid <- expand.grid(mstop = 20,
                    nu = 0.001)
if (file.exists("boosting_bstlm.rds")) {
  model_bstlm <- readRDS(file = "boosting_bstlm.rds")
} else {
  model_bstlm <- train(label ~ .,
                       data = data_train,
                       method = "BstLm",
                       metric = metric,
                       tuneGrid = grid,
                       trControl = control)
  saveRDS(model_bstlm, file = "boosting_bstlm.rds")
}
```

Resultados

```
results <- resamples(list(c50 = model_c50,
                          adaboost = model_adaboost,
                          bstlm = model_bstlm))
summary(results)
```

```
## 
## Call:
## summary.resamples(object = results)
## 
## Models: c50, adaboost, bstlm 
## Number of resamples: 30 
## 
## Accuracy 
##               Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## c50      0.7941176 0.8484848 0.8650000 0.8647134 0.8800000 0.9300000    0
## adaboost 0.7346939 0.7682576 0.8009901 0.8045802 0.8379208 0.8910891    0
## bstlm    0.1113225 0.1114020 0.1114640 0.1115132 0.1116663 0.1117815    0
## 
## Kappa 
##               Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## c50      0.7710560 0.8310963 0.8496932 0.8494053 0.8663622 0.9220316    0
## adaboost 0.7047167 0.7420515 0.7786948 0.7825081 0.8195683 0.8789233    0
## bstlm    0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000    0
```

```
dotplot(results)
```



Test

C5.0

```
pred_c50 <- predict(model_c50, data_test[1:1000, ])
conf_matrix_c50 <- table(data_test[1:1000, ]$label, pred_c50)
```

```
conf_matrix_c50
```

```
##    pred_c50
##       0   1   2   3   4   5   6   7   8   9
##   0  98   0   0   0   0   1   1   0   0   0
##   1   1 115   1   0   0   1   0   0   0   0
##   2   2   1  95   0   0   0   2   0   4   0
##   3   0   0   2  53   0   1   1   0   4   3
##   4   1   1   0   0  89   0   0   1   1   8
##   5   2   0   1   4   1  75   0   0   2   1
##   6   1   0   0   0   4   2  94   0   3   0
##   7   1   1   1   0   3   0   0 105   1   4
##   8   0   2   4   0   0   0   0   0  90   0
##   9   0   0   3   2   4   1   0   3   2  96
```

```
accuracy_c50 <- mean(pred_c50 == data_test[1:1000, ]$label)
cat("Accuracy with C5.0 boosting: ", accuracy_c50)
```

```
## Accuracy with C5.0 boosting:  0.91
```

AdaBoost.M1

```
pred_adaboost <- predict(model_adaboost, data_test[1:1000, ])
conf_matrix_adaboost <- table(data_test[1:1000, ]$label, pred_adaboost)
```

```
conf_matrix_adaboost
```

```
##    pred_adaboost
##       0   1   2   3   4   5   6   7   8   9
##   0  99   0   1   0   0   0   0   0   0   0
##   1   1 114   0   0   0   1   0   0   1   1
##   2   3   3  88   0   1   1   2   3   2   1
##   3   0   1   1  45   0   4   2   0   3   8
##   4   0   1   1   0  76   0   2   1   3  17
##   5   0   0   0   6   0  69   1   0   2   8
##   6   2   1   0   0   4   1  94   0   1   1
##   7   0   2   2   0   2   0   0 101   1   8
##   8   1   2   7   4   0   2   0   0  77   3
##   9   2   0   1   2  11   1   0   9   2  83
```

```
accuracy_adaboost <- mean(pred_adaboost == data_test[1:1000, ]$label)
cat("Accuracy with AdaBoost.M1 boosting: ", accuracy_adaboost)
```

```
## Accuracy with AdaBoost.M1 boosting:  0.846
```

Boosted Linear Model

```
pred_bstlm <- predict(model_bstlm, data_test)
conf_matrix_bstlm <- table(data_test$label, pred_bstlm)
```

```
conf_matrix_bstlm
```

```
##    pred_bstlm
##        0    1    2    3    4    5    6    7    8    9
##   0    0 1033    0    0    0    0    0    0    0    0
##   1    0 1171    0    0    0    0    0    0    0    0
##   2    0 1044    0    0    0    0    0    0    0    0
##   3    0 1087    0    0    0    0    0    0    0    0
##   4    0 1018    0    0    0    0    0    0    0    0
##   5    0  948    0    0    0    0    0    0    0    0
##   6    0 1034    0    0    0    0    0    0    0    0
##   7    0 1100    0    0    0    0    0    0    0    0
##   8    0 1015    0    0    0    0    0    0    0    0
##   9    0 1047    0    0    0    0    0    0    0    0
```

```
accuracy_bstlm <- mean(pred_bstlm == data_test$label)
cat("Accuracy with Boosted Linear Model boosting: ", accuracy_bstlm)
```

```
## Accuracy with Boosted Linear Model boosting:  0.1115557
```

Conclusión final

De las tres posibilidades de boosting, el método C5.0 es el que más precisión reporta (alrededor del 90%)

Red neuronal sencilla

```
if (!file.exists("simple_nnet_model.rds")) {

  # Entrenamos modelo
  simple_nnet <- multinom(label ~ ., data=data_train, MaxNWts=10000, decay=5e-3,
maxit=100)

  # Guardamos modelo
  saveRDS(simple_nnet, file = "simple_nnet_model.rds")

} else {
  # cargamos modelo
  simple_nnet <- readRDS("simple_nnet_model.rds")
}
```

```
prediction_simple_nnet <- predict(simple_nnet, data_test, type = "class")
prediction_simple_nnet[1:4]
```

```
## [1] 3 1 2 7
## Levels: 0 1 2 3 4 5 6 7 8 9
```

```
data_test$label[1:4]
```

```
## [1] 3 1 2 7
## Levels: 0 1 2 3 4 5 6 7 8 9
```

```
confussion_matrix <- table(data_test$label, prediction_simple_nnet)
confussion_matrix
```

```
##    prediction_simple_nnet
##        0    1    2    3    4    5    6    7    8    9
##   0  978    0    7    1    4   11   19    4    8    1
##   1    0 1120    7    8    2    2    2    2   26    2
##   2    5    7  913   26   23   17   13   16   19    5
##   3   11    9   28  915    3   34   10   16   38   23
##   4    4    4    3    1  932    0   14    6   10   44
##   5   22   10   12   36   18  751   31    3   51   14
##   6    6    2   18    0    9   11  975    5    8    0
##   7    7   11   21    4    8    2    2  997    5   43
##   8    9   20   23   19   17   25    9    5  868   20
##   9    5    4    5   10   37    3    1   35   10  937
```

```
accuracy_simple_nnet <- mean(prediction_simple_nnet == data_test$label)
cat("Accuracy with a simple neural network:", accuracy_simple_nnet)
```

```
## Accuracy with a simple neural network: 0.8941602
```

```
# Obtenemos un accuracy cercano al 90% no está mal pero vamos a intentar mejorarl
o
```