Ensemble Methods

Adaptive and Gradient Boosting

Santander Meteorology Group

Objetivo:

En la presente práctica trataremos de profundizar en la utilización de los métodos de basados ensembles y compararlos con su contrapartida basada en árboles. Para ello, en primer lugar instalaremos los paquetes asociados a los métodos de ensembles

```
install.packages("randomForest")
install.packages("adabag")
install.packages("gbm")
```

y cargamos las librerías correspondientes:

```
library(tree) ## arboles
library(rpart) ## Tree-based model
library(randomForest) ## bagging: random forests
library(adabag) ## boosting: adaptive boosting
library(gbm) ## boosting: Gradient boosting
library(caret)
library(MASS)
```

A lo largo de la práctica usaremos varios datasets para ejemplificar el uso de las diferentes funciones.

Clasificación:

Ejemplo: Iris dataset

Cargamos los datos y definimos los conjuntos de `train` y de `test`:

```r
## train/test partition
set.seed(23)
n <- nrow(iris)
indtrain <- sample(1:n, round(0.75*n))  # indices for train
indtest <- setdiff(1:n, indtrain)  # indices for test
```

y predecimos y evaluamos sobre ambos conjuntos utilizando un árbol de decisión:

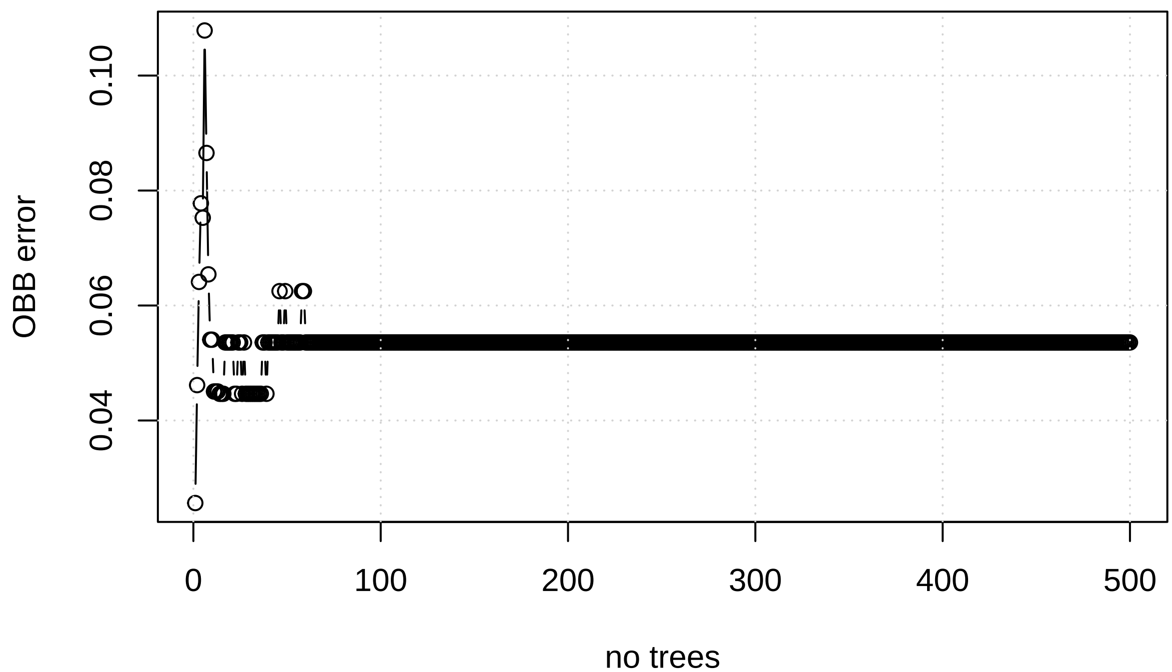
```r
## Single Tree:
t <- tree(Species ~., iris, subset = indtrain,
          control = tree.control(length(indtrain), mincut = 1, minsize = 2, mindev = 0))
## Prediction for test
pred.t.test <- predict(t, iris[indtest, ], type = "class")
## Prediction for train
pred.t.train <- predict(t, iris[indtrain, ], type = "class")
## Accuracy
print(c(sum(diag(table(pred.t.test, iris$Species[indtest]))) / length(indtest),
        sum(diag(table(pred.t.train, iris$Species[indtrain]))) / length(indtrain)))
```

```
## [1] 0.9210526 1.0000000
```

A continuación, siguiendo con lo visto en la sesión anterior, realizamos la predicción considerando los `random forest` utilizando el valor por defector para el número de variables seleccionadas para cada árbol:

```r
set.seed(23)
## Bagging: Random Forests
rf <- randomForest(Species ~., iris , subset = indtrain, ntree = 500, mtry = 2)


# OOB error
plot(rf$err.rate[, 1], type = "b", xlab = "no trees",
ylab = "OBB error")
grid()
```



A la vista de los resultados consideramos el número de árboles óptimo sobre 100 dado que es la zona de estabilización del parámetros de validación:

```
## Bagging: Random Forests
rf <- randomForest(Species ~., iris , subset = indtrain, ntree = 100, mtry = 2)


## Prediction for test
pred.rf.test <- predict(rf, iris[indtest, ])
## Prediction for train
pred.rf.train <- predict(rf, iris[indtrain, ])
## Accuracy
print(c(sum(diag(table(pred.rf.test, iris$Species[indtest]))) / length(indtest),
        sum(diag(table(pred.rf.train, iris$Species[indtrain]))) / length(indtrain)))
```

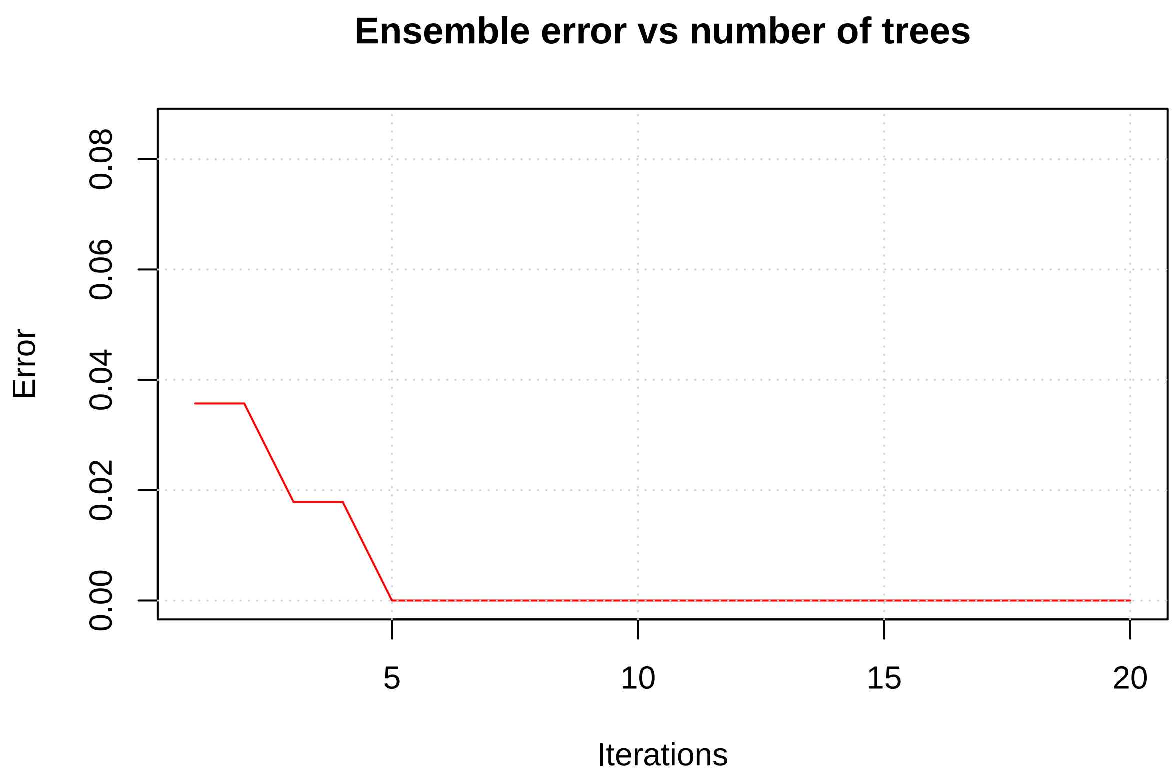
```
## [1] 0.9473684 1.0000000
```

Consideremos ahora el método estándar de boosting, el Adaptive Boosting (adaboost). Para ello, revisemos inicialmente los parámetros de la función boosting:

```r
## Boosting: Adaptive Boosting (AdaBoost)
? boosting
```

Dado el tamaño del dataset, consideremos un número limitado de árboles, pero suficientemente grande, para ver el número óptimo de árboles a utilizar:

```r
# AdaBoost with 20 trees (mfinal)
ab = boosting(Species ~., iris[indtrain, ], mfinal = 20, boos = FALSE)
# train errors as a function of number of trees
plot(errorevol(ab, iris[indtrain, ]))
grid()
```



Como vemos, a partir de 5 árboles ya el error es nulo, de modo que el número máximo de árboles debe ser a lo sumo 5.

```r
## Boosting: Adaptive Boosting (AdaBoost)
## 20 trees (mfinal)
ab <- boosting(Species ~., iris[indtrain, ], mfinal = 5)
## Prediction for test
pred.ab.test <- predict(ab, iris[indtest, ])
## Prediction for train
pred.ab.train <- predict(ab, iris[indtrain, ])
## Accuracy
c(sum(diag(table(pred.ab.test$class, iris$Species[indtest]))) / length(indtest),
  sum(diag(table(pred.ab.train$class, iris$Species[indtrain]))) / length(indtrain))
```

```
## [1] 0.9210526 0.9910714
```

Al considerar árboles, y como dice la ayuda, podemos definir los parámetros de control de los árboles construidos con la función `rpart`:

```r
## Boosting: Adaptive Boosting (AdaBoost)
ab <- boosting(Species ~., iris[indtrain, ], mfinal = 5,
               control=rpart.control(minsplit = 2, minbucket = 1, cp = 0.01))
## Prediction for test
pred.ab.test <- predict(ab, iris[indtest, ])
## Prediction for train
pred.ab.train <- predict(ab, iris[indtrain, ])
## Accuracy
c(sum(diag(table(pred.ab.test$class, iris$Species[indtest]))) / length(indtest),
  sum(diag(table(pred.ab.train$class, iris$Species[indtrain]))) / length(indtrain))
```

```
## [1] 0.9473684 1.0000000
```

Finalmente, consideremos el Gradient Boosting para lo cual revisemos los parámetros de la función `gbm`:

```
? gbm
```

Los argumentos base vistos en la sesión teórica se corresponden con:

- shrinkage
- n.trees
- interaction.depth

Si bien hay otros argumentos que permiten el control de las caracteristicas de los arboles, la validacion cruzada o la aleatorizacion del conjunto de entrenamiento.

```
## Boosting: Gradient Boosting
gb <- gbm(Species~., data=iris[indtrain, ], n.trees=1000, interaction.depth=20,
          shrinkage = 0.01)
```

```
## Distribution not specified, assuming multinomial ...
```

¿Cuántos árboles seleccionarías?

```
## Prediction for test
pred.gb.test <- predict(object = gb, newdata = iris[indtest, ], n.trees = 1000,
                        type = "response")
## Prediction for train
pred.gb.train <- predict(object = gb, newdata = iris[indtrain, ], n.trees = 1000,
                         type = "response")
## Accuracy
c(sum(diag(table(attributes(pred.gb.test)$dimnames[[2]][apply(pred.gb.test, FUN = which.max, MA
                 iris$Species[indtest]))) / length(indtest),
  sum(diag(table(attributes(pred.gb.test)$dimnames[[2]][apply(pred.gb.train, FUN = which.max, MA
                 iris$Species[indtrain]))) / length(indtrain))
```

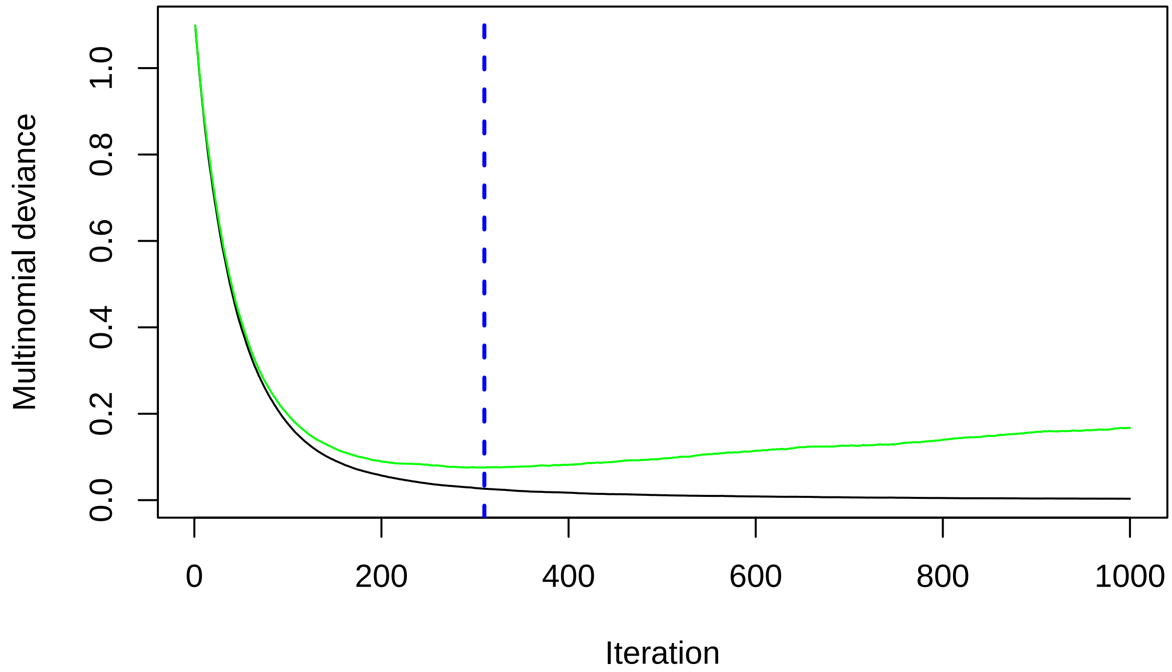
```
## [1] 0.9210526 1.0000000
```

Como dijimos, pueden incluirse parámetros que controlen la validación cruzada, obteniendo el valor óptimo:

```r
gb.cv <- gbm(Species~., data=iris[indtrain, ], n.trees=1000,
             interaction.depth=20, shrinkage = 0.01, cv.folds = 4)
```

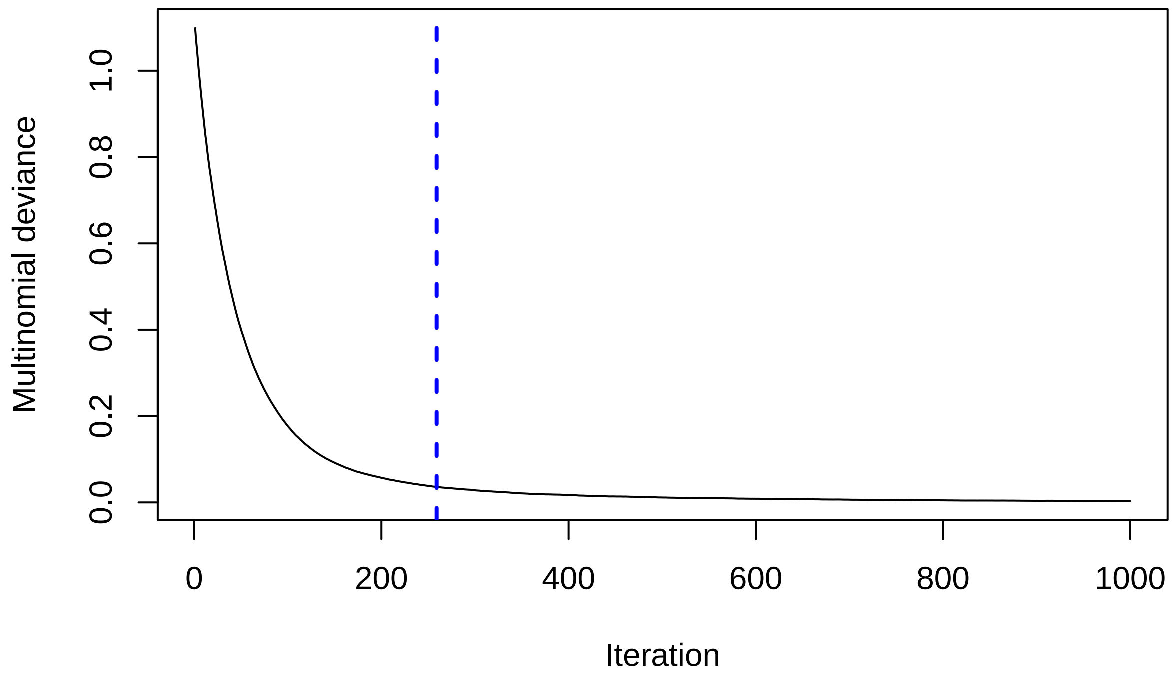
```
## Distribution not specified, assuming multinomial ...
```

```r
ntree_opt_cv <- gbm.perf(gb.cv, method = "cv")
```




```r
ntree_opt_oob <- gbm.perf(gb.cv, method = "OOB")
```

```
## OOB generally underestimates the optimal number of iterations although predictive performance
```




```r
print(ntree_opt_cv)
```

```
## [1] 310
```

```r
print(ntree_opt_oob)
```

```
## [1] 259
## attr(,"smoother")
## Call:
## loess(formula = object$oobag.improve ~ x, enp.target = min(max(4,
##     length(x)/10), 50))
##
## Number of Observations: 1000
## Equivalent Number of Parameters: 40
## Residual Standard Error: 0.0005396
```

Que podemos usar en el ajuste:

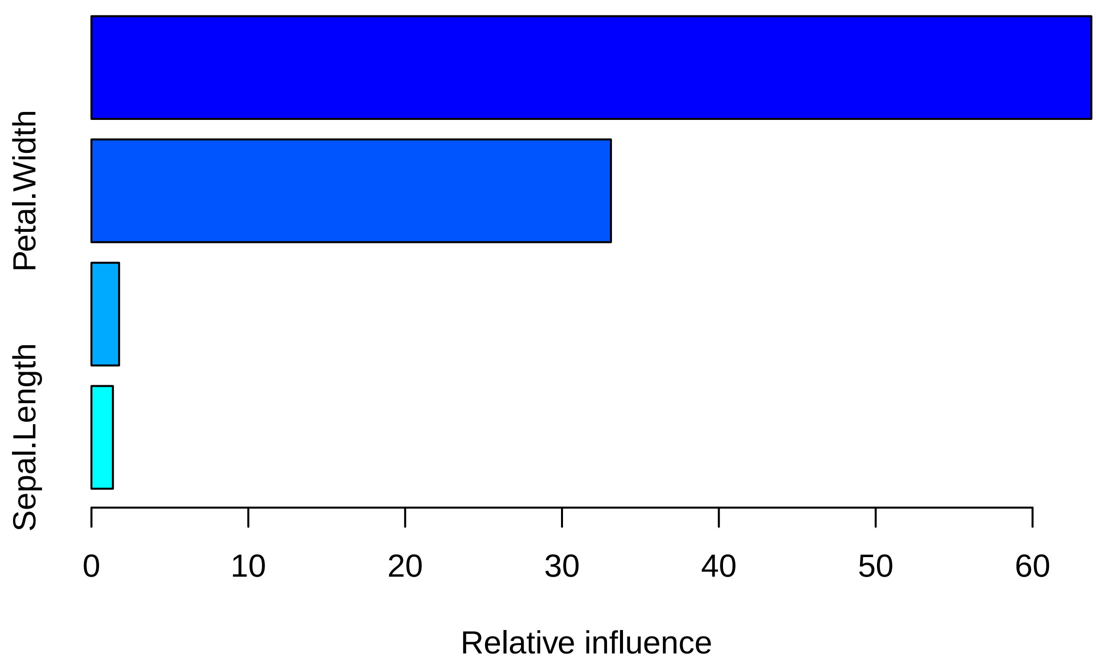
```
gb <- gbm(Species~., data=iris[indtrain, ], n.trees=ntree_opt_cv,
          interaction.depth=20, shrinkage = 0.01)
```

```
## Distribution not specified, assuming multinomial ...
```

```
print(gb)
```

```
## gbm(formula = Species ~ ., data = iris[indtrain, ], n.trees = ntree_opt_cv,
##     interaction.depth = 20, shrinkage = 0.01)
## A gradient boosted model with multinomial loss function.
## 310 iterations were performed.
## There were 4 predictors of which 4 had non-zero influence.
```

```
summary(gb)
```




```
##                       var   rel.inf
## Petal.Length Petal.Length 63.748011
```

```
## Petal.Width   Petal.Width 33.121518
## Sepal.Width   Sepal.Width  1.762364
## Sepal.Length Sepal.Length  1.368107
```

```r
## Prediction for test
pred.gb.test <- predict(object = gb, newdata = iris[indtest, ],
                        n.trees = ntree_opt_cv, type = "response")
## Prediction for train
pred.gb.train <- predict(object = gb, newdata = iris[indtrain, ],
                         n.trees = ntree_opt_cv, type = "response")
## Accuracy
c(sum(diag(table(attributes(pred.gb.test)$dimnames[[2]][apply(pred.gb.test, FUN = which.max, MA
                 iris$Species[indtest]))) / length(indtest),
  sum(diag(table(attributes(pred.gb.test)$dimnames[[2]][apply(pred.gb.train, FUN = which.max, MA
                 iris$Species[indtrain]))) / length(indtrain))
```

```
## [1] 0.9210526 1.0000000
```

Nuevas implementaciones optimizan el algoritmo realizando el ajuste de forma más eficiente, las cuales probaremos con el dataset `Meteo`:

```r
install.packages("xgboost") ## Extreme Gradient Boosting
```

Ejemplo: Meteo dataset

```r
library(MASS)
library(tree)
library(randomForest)
library(adabag)
library(gbm)
library(caret)
library(xgboost)
```

Cargamos los datos

```r
load("~/meteo.RData")
```

y definimos los conjuntos de `train` y de `test`:

```r
## Keeping the first 10 years (3650 days) for this example
n <- 3650
y <- y[1:n]
x <- x[1:n, ]
## train/test partition
set.seed(23)
indtrain <- sample(1:n, round(0.75*n))  # indices for train
indtest <- setdiff(1:n, indtrain)  # indices for test
```

Sigamos con el caso de clasificación:

```r
## binary occurrence (1/0)
occ <- y
occ[which(y < 1)] <- 0
occ[which(y >= 1)] <- 1
## dataframe for occurrence
df.occ <- data.frame(y.occ = as.factor(occ), predictors = x)
```

y probemos las diferentes técnicas vistas a lo largo del curso.

Decision Trees:

```r
## Single Tree:
t <- tree(y.occ ~., df.occ, subset = indtrain,
          control = tree.control(length(indtrain), mincut = 1, minsize = 2, mindev = 0))
## Prediction for test
pred.t.test <- predict(t, df.occ[indtest, ], type = "class")
## Prediction for train
pred.t.train <- predict(t, df.occ[indtrain, ], type = "class")
## Accuracy
```

```r
print(c(sum(diag(table(pred.t.test, df.occ$y.occ[indtest]))) / length(indtest),
        sum(diag(table(pred.t.train, df.occ$y.occ[indtrain]))) / length(indtrain)))
```

```
## [1] 0.8267544 1.0000000
```

Para discutir más en profundidad la validación obtengamos las matrices de confusión para el test

```r
confusionMatrix(pred.t.test, df.occ$y.occ[indtest])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 649  73
##          1  85 105
##
##                Accuracy : 0.8268
##                  95% CI : (0.8006, 0.8508)
##     No Information Rate : 0.8048
##     P-Value [Acc > NIR] : 0.05007
##
##                   Kappa : 0.4623
##
##  Mcnemar's Test P-Value : 0.38151
##
##             Sensitivity : 0.8842
##             Specificity : 0.5899
##          Pos Pred Value : 0.8989
##          Neg Pred Value : 0.5526
##              Prevalence : 0.8048
##          Detection Rate : 0.7116
##    Detection Prevalence : 0.7917
```

```
##       Balanced Accuracy : 0.7370
##
##        'Positive' Class : 0
##
```

y el train:

```r
confusionMatrix(pred.t.train, df.occ$y.occ[indtrain])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2171    0
##          1    0  567
##
##                Accuracy : 1
##                  95% CI : (0.9987, 1)
##     No Information Rate : 0.7929
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.7929
##          Detection Rate : 0.7929
##    Detection Prevalence : 0.7929
```

```
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

Bagging: Random Forests

```
## Single Tree:
rf <- randomForest(y.occ ~., df.occ, subset = indtrain, ntree = 500)
## Prediction for test
pred.rf.test <- predict(rf, df.occ[indtest, ])
## Prediction for train
pred.rf.train <- predict(rf, df.occ[indtrain, ])
## Accuracy
print(c(sum(diag(table(pred.rf.test, df.occ$y.occ[indtest]))) / length(indtest),
        sum(diag(table(pred.rf.train, df.occ$y.occ[indtrain]))) / length(indtrain)))
```

```
## [1] 0.8804825 1.0000000
```

Para discutir más en profundidad la validación obtengamos las matrices de confusión para el test

```
confusionMatrix(pred.rf.test, df.occ$y.occ[indtest])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 696  71
##          1  38 107
##
##                Accuracy : 0.8805
##                  95% CI : (0.8576, 0.9008)
##     No Information Rate : 0.8048
```

```
##     P-Value [Acc > NIR] : 6.949e-10
##
##                   Kappa : 0.5908
##
##  Mcnemar's Test P-Value : 0.002176
##
##             Sensitivity : 0.9482
##             Specificity : 0.6011
##          Pos Pred Value : 0.9074
##          Neg Pred Value : 0.7379
##              Prevalence : 0.8048
##          Detection Rate : 0.7632
##    Detection Prevalence : 0.8410
##       Balanced Accuracy : 0.7747
##
##        'Positive' Class : 0
##
```

y el train:

```
confusionMatrix(pred.rf.train, df.occ$y.occ[indtrain])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2171    0
##          1    0  567
##
##                Accuracy : 1
##                  95% CI : (0.9987, 1)
##     No Information Rate : 0.7929
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.7929
##          Detection Rate : 0.7929
##    Detection Prevalence : 0.7929
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

Boosting: Adaptive Boosting (AdaBoost)

```r
## 20 trees (mfinal)
ab <- boosting(y.occ ~., df.occ[indtrain, ], mfinal = 20)
## Prediction for test
pred.ab.test <- predict(ab, df.occ[indtest, ])
## Prediction for train
pred.ab.train <- predict(ab, df.occ[indtrain, ])
## Accuracy
print(c(sum(diag(table(pred.ab.test$class, df.occ$y.occ[indtest]))) / length(indtest),
        sum(diag(table(pred.ab.train$class, df.occ$y.occ[indtrain]))) / length(indtrain)))
```

```
## [1] 0.8684211 1.0000000
```

Para discutir más en profundidad la validación obtengamos las matrices de confusión para el test

```
confusionMatrix(as.factor(pred.ab.test$class), df.occ$y.occ[indtest])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 681  67
##          1  53 111
##
##                Accuracy : 0.8684
##                  95% CI : (0.8447, 0.8897)
##     No Information Rate : 0.8048
##     P-Value [Acc > NIR] : 2.486e-07
##
##                   Kappa : 0.5683
##
##  Mcnemar's Test P-Value : 0.2353
##
##             Sensitivity : 0.9278
##             Specificity : 0.6236
##          Pos Pred Value : 0.9104
##          Neg Pred Value : 0.6768
##              Prevalence : 0.8048
##          Detection Rate : 0.7467
##    Detection Prevalence : 0.8202
##       Balanced Accuracy : 0.7757
##
##        'Positive' Class : 0
##
```

y el train:

```r
confusionMatrix(as.factor(pred.ab.train$class), df.occ$y.occ[indtrain])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2171    0
##          1    0  567
##
##                Accuracy : 1
##                  95% CI : (0.9987, 1)
##     No Information Rate : 0.7929
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.7929
##          Detection Rate : 0.7929
##    Detection Prevalence : 0.7929
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

Boosting: eXtreme Gradient Boosting

```
df.gb.occ <- df.occ
df.gb.occ$y.occ <- as.character(df.gb.occ$y.occ)

gb <- xgboost(data = x[indtrain,], label = df.gb.occ$y.occ[indtrain], max.depth = 6, eta = 1, nr
              nthread = 2, objective = "binary:logistic")
```

```
## [14:22:39] WARNING: amalgamation/../src/learner.cc:1115: Starting in XGBoost 1.3.0, the defau
## [1]  train-logloss:0.268630
```

```
print(gb)
```

```
## ##### xgb.Booster
## raw: 8.2 Kb
## call:
##   xgb.train(params = params, data = dtrain, nrounds = nrounds,
##     watchlist = watchlist, verbose = verbose, print_every_n = print_every_n,
##     early_stopping_rounds = early_stopping_rounds, maximize = maximize,
##     save_period = save_period, save_name = save_name, xgb_model = xgb_model,
##     callbacks = callbacks, max.depth = 6, eta = 1, nthread = 2,
##     objective = "binary:logistic")
## params (as set within xgb.train):
##   max_depth = "6", eta = "1", nthread = "2", objective = "binary:logistic", validate_paramete
## xgb.attributes:
##   niter
## callbacks:
##   cb.print.evaluation(period = print_every_n)
##   cb.evaluation.log()
## niter: 1
## nfeatures : 320
## evaluation_log:
```

```
##  iter train_logloss
##     1       0.26863
```

```r
summary(gb)
```

```
##                Length Class              Mode
## handle            1   xgb.Booster.handle externalptr
## raw            8309   -none-             raw
## niter             1   -none-             numeric
## evaluation_log    2   data.table         list
## call             17   -none-             call
## params            5   -none-             list
## callbacks         2   -none-             list
## nfeatures         1   -none-             numeric
```

```r
## Prediction for test
pred.gb.test <- predict(gb, newdata = x[indtest, ], type = "response")
## Prediction for train
pred.gb.train <- predict(gb, newdata = x[indtrain, ], type = "response")
## Accuracy
pred.gb.test.bin <- as.factor(ifelse(pred.gb.test>mean(as.numeric(df.gb.occ$y.occ[indtrain])),1
pred.gb.train.bin <- as.factor(ifelse(pred.gb.train>mean(as.numeric(df.gb.occ$y.occ[indtrain]))


print(c(sum(diag(table(pred.gb.test.bin, df.gb.occ$y.occ[indtest]))) / length(indtest),
        sum(diag(table(pred.gb.train.bin, df.gb.occ$y.occ[indtrain]))) / length(indtrain)))
```

```
## [1] 0.8223684 0.8703433
```

Para discutir más en profundidad la validación obtengamos las matrices de confusión para el test

```r
confusionMatrix(pred.gb.test.bin, df.occ$y.occ[indtest])
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction   0   1
##          0 618  46
##          1 116 132
##
##                Accuracy : 0.8224
##                  95% CI : (0.796, 0.8466)
##     No Information Rate : 0.8048
##     P-Value [Acc > NIR] : 0.0966
##
##                   Kappa : 0.5079
##
##  Mcnemar's Test P-Value : 5.922e-08
##
##             Sensitivity : 0.8420
##             Specificity : 0.7416
##          Pos Pred Value : 0.9307
##          Neg Pred Value : 0.5323
##              Prevalence : 0.8048
##          Detection Rate : 0.6776
##    Detection Prevalence : 0.7281
##       Balanced Accuracy : 0.7918
##
##        'Positive' Class : 0
##
```

y el train:

```
confusionMatrix(pred.gb.train.bin, df.occ$y.occ[indtrain])
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction    0    1
##          0 1884   68
##          1  287  499
##
##                Accuracy : 0.8703
##                  95% CI : (0.8572, 0.8827)
##     No Information Rate : 0.7929
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6545
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8678
##             Specificity : 0.8801
##          Pos Pred Value : 0.9652
##          Neg Pred Value : 0.6349
##              Prevalence : 0.7929
##          Detection Rate : 0.6881
##    Detection Prevalence : 0.7129
##       Balanced Accuracy : 0.8739
##
##        'Positive' Class : 0
##
```

Predicción:

A modo de ejemplo hemos utilizado el dataset `Boston` para problemas de predicción.

```r
library(MASS)

n <- nrow(Boston)
# train/test partition
indtrain <- sample(1:n, round(0.75*n))  # indices for train
indtest <- setdiff(1:n, indtrain)  # indices for test

# RF
rf <- randomForest(medv ~., Boston , subset = indtrain)
# RF configuration?

# OOB error?
plot(rf$mse, type = "l", xlab = "no. trees", ylab = "OOB error")
grid()
```

Extender el análisis hecho para el problema de clasificación a este dataset, comparando las diferentes aproximaciones.

Session Info:

```
## R version 4.1.2 (2021-11-01)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.3 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=es_ES.UTF-8        LC_COLLATE=en_US.UTF-8
```

```
##  [5] LC_MONETARY=es_ES.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=es_ES.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=es_ES.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] xgboost_1.5.0.2     MASS_7.3-54         gbm_2.1.8
##  [4] adabag_4.2          doParallel_1.0.16   iterators_1.0.13
##  [7] foreach_1.5.1       caret_6.0-90        lattice_0.20-45
## [10] ggplot2_3.3.5       randomForest_4.6-14 rpart_4.1-15
## [13] tree_1.0-41
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.6           lubridate_1.8.0      listenv_0.8.0
##  [4] class_7.3-19         digest_0.6.27        ipred_0.9-12
##  [7] utf8_1.2.2           parallelly_1.29.0    R6_2.5.0
## [10] plyr_1.8.6           stats4_4.1.2         e1071_1.7-9
## [13] evaluate_0.14        highr_0.8            pillar_1.6.4
## [16] rlang_0.4.12         data.table_1.14.2    Matrix_1.4-0
## [19] rmarkdown_2.11       splines_4.1.2        gower_0.2.2
## [22] stringr_1.4.0        munsell_0.5.0        proxy_0.4-26
## [25] compiler_4.1.2       xfun_0.29            pkgconfig_2.0.3
## [28] globals_0.14.0       htmltools_0.5.1.1    nnet_7.3-16
## [31] tidyselect_1.1.1     tibble_3.1.6         prodlim_2019.11.13
## [34] codetools_0.2-18     fansi_0.5.0          future_1.23.0
## [37] crayon_1.4.2         dplyr_1.0.7          withr_2.4.3
```

```
## [40] recipes_0.1.17       ModelMetrics_1.2.2.2 grid_4.1.2
## [43] jsonlite_1.7.2       nlme_3.1-152         gtable_0.3.0
## [46] lifecycle_1.0.1      magrittr_2.0.1       pROC_1.18.0
## [49] scales_1.1.1         future.apply_1.8.1   stringi_1.5.3
## [52] reshape2_1.4.4       timeDate_3043.102    ellipsis_0.3.2
## [55] generics_0.1.1       vctrs_0.3.8          lava_1.6.10
## [58] tools_4.1.2          glue_1.4.2           purrr_0.3.4
## [61] survival_3.2-13      yaml_2.2.1           colorspace_2.0-2
## [64] knitr_1.31
```