

# Random Forests

## Contents

1	Lectura de datos	1
2	Training set <i>vs</i> Test set	1
3	Random Forest	1
3.1	mtry = 4 ( <b>BAGGING</b> )	2
3.2	mtry = 3	2
3.3	mtry = 2	2
4	Prediccion de un valor concreto	3
5	Importancia de variables	3

## 1 Lectura de datos

```
library(rpart)
load("datos/kidiq.Rdata")
```

## 2 Training set *vs* Test set

Dividimos los datos en dos partes, una para entrenar el modelo y otra para calcular el error de predicción con datos diferentes de los utilizados para entrenar:

```
set.seed(321)
n = nrow(d)
pos_train = sample(1:n,round(0.8*n)) # la mitad de los datos para entranamiento
datos_train = d[pos_train,] # training set
datos_test = d[-pos_train,] # test set
```

## 3 Random Forest

- Se remuestrea con reemplazamiento  $B$  veces (bootstrap).
- Se estima un arbol para cada muestra,  $f_b(x)$ . Pero cada vez que se divide un nodo en cada arbol, se seleccionan aleatoriamente  $m$  regresores de los  $p$  disponibles. Por defecto, R considera  $p/3$ . En el caso de que  $m = p$  se denomina **BAGGING**.
- Se calcula la predicción proporcionada por cada árbol,  $\hat{f}_b(x)$ .
- Se promedian las predicciones.

$$\hat{f}_{RF} = \frac{1}{B} \sum_{b=1}^B \hat{f}_b(x)$$

- La gran ventaja de random forest frente a bagging es que funciona muy bien con datos que tienen variables correlacionadas.

### 3.1 mtry = 4 (BAGGING)

```
library(randomForest)

## randomForest 4.7-1.2

## Type rfNews() to see new features/changes/bug fixes.

# numero total de regresores: 4
rf1 = randomForest(kid_score ~ ., data = datos_train, mtry = 4, ntree = 500)
```

Error del modelo:

```
yp_train_rf1 <- predict(rf1, newdata = datos_train)
y_train = datos_train$kid_score
# error cuadratico medio en los datos de training
( MSE_train_rf1 = mean((y_train - yp_train_rf1)^2) )
```

```
## [1] 87.36114
```

Error de predicción:

```
# prediccion del consumo con los datos test
yp_test_rf1 = predict(rf1, newdata = datos_test)

# error del test set
y_test = datos_test$kid_score
(MSE_test_rf1 = mean((y_test - yp_test_rf1)^2))
```

```
## [1] 380.8566
```

### 3.2 mtry = 3

```
rf2 = randomForest(kid_score ~ ., data = datos_train, mtry = 3, ntree = 500)
```

Error del modelo:

```
yp_train_rf2 <- predict(rf2, newdata = datos_train)
# error cuadratico medio en los datos de training
( MSE_train_rf2 = mean((y_train - yp_train_rf2)^2) )
```

```
## [1] 92.32
```

Error de predicción:

```
# prediccion del consumo con los datos test
yp_test_rf2 = predict(rf2, newdata = datos_test)
# error del test set
(MSE_test_rf2 = mean((y_test - yp_test_rf2)^2))
```

```
## [1] 380.8993
```

### 3.3 mtry = 2

```
rf3 = randomForest(kid_score ~ ., data = datos_train, mtry = 2, ntree = 500)
```

Error del modelo:

```
yp_train_rf3 <- predict(rf3, newdata = datos_train)
# error cuadrático medio en los datos de training
( MSE_train_rf3 = mean((y_train - yp_train_rf3)^2) )
```

```
## [1] 116.7505
```

Error de predicción:

```
# predicción del consumo con los datos test
yp_test_rf3 = predict(rf3, newdata = datos_test)

# error del test set
(MSE_test_rf3 = mean((y_test - yp_test_rf3)^2))
```

```
## [1] 376.1012
```

## 4 Predicción de un valor concreto

```
# el paquete randomForest requiere que definamos las variables cualitativas como factores
xp = data.frame(mom_iq = 95, mom_age = 30,
               mom_hs = factor("si", levels = c("no", "si")),
               mom_work = factor("notrabaja", levels = c("notrabaja", "trabaja23", "trabaja1p", "traba
# la predicción se calcula como el promedio de las predicciones de cada uno de los 500 árboles
predict(rf1, newdata = xp)
```

```
##           1
## 75.74037
```

## 5 Importancia de variables

```
rf4 = randomForest(kid_score ~ ., data = datos_train, mtry = 4, ntree = 500, importance = T)

importance(rf4)
```

```
##           %IncMSE IncNodePurity
## mom_hs      4.662314      3630.08
## mom_iq     44.337078     82797.27
## mom_work   11.665066     13623.48
## mom_age     6.490812     26319.00
```

- %IncMSE: incremento del MSE de las predicciones calculadas en los datos *out of samples* cuando los valores de una variable dada se permutan aleatoriamente (sería como quitarla del modelo). Cuando más grande es IncMSE, más importante es la variable.
- IncNodePurity: suma del descenso acumulado del RSS al partir por dicha variable.

```
varImpPlot(rf4)
```

rf4

