Machine Learning and Forecasting Models

Part 3: Tree Based Models and Neural Networks

Gabriel Vasconcelos

Random Forest

- Pacotes utilizados:

```
library(randomForest)
library(ggplot2)
library(reshape2)
library(tree)
library(gridExtra)
```

- Banco de Dados

```
data(Housing,package="Ecdat")
```

Random Forest

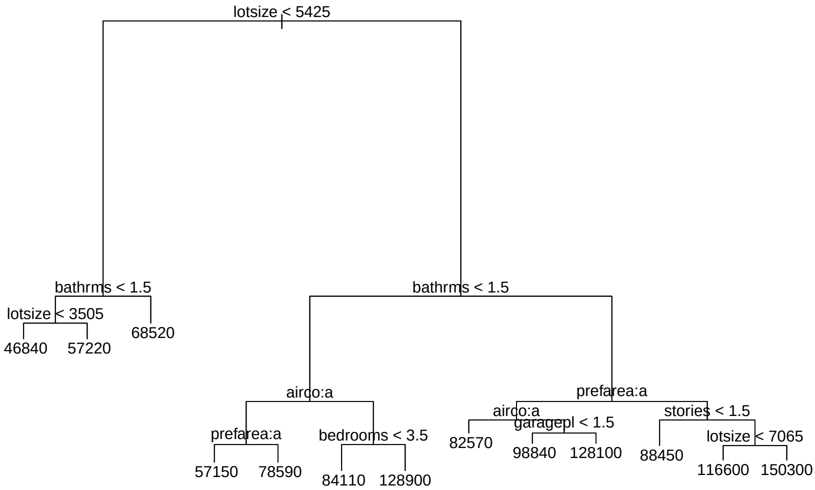
- The example dataset has 546 observations of sold houses,
- We want to forecast the selling price,
- Characteristics: Bedrooms, Bathrooms, garages, etc.

Random Forest

- First we will look at individual trees.
- Undertand the instability problem

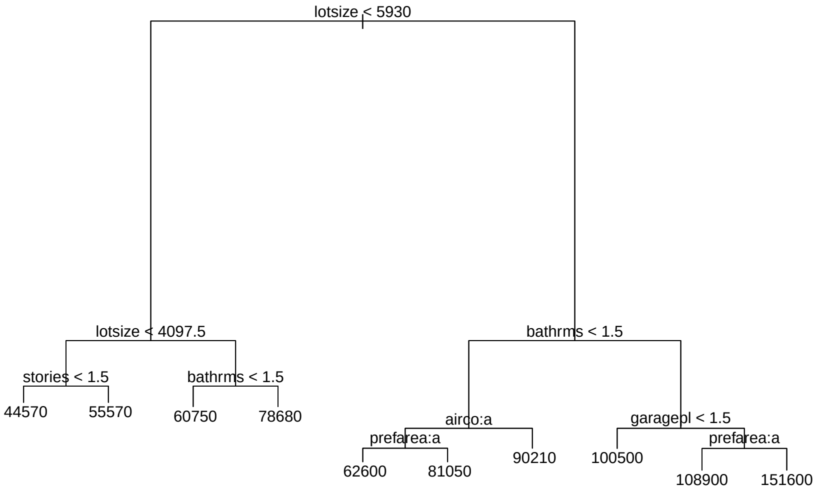
Random Forest

```r
# = First Tree
set.seed(1) # = Seed for Replication
tree1 = tree(price ~.,
             data = Housing[sample(1:nrow(Housing),replace=TRUE),])
plot(tree1);  text(tree1)
```



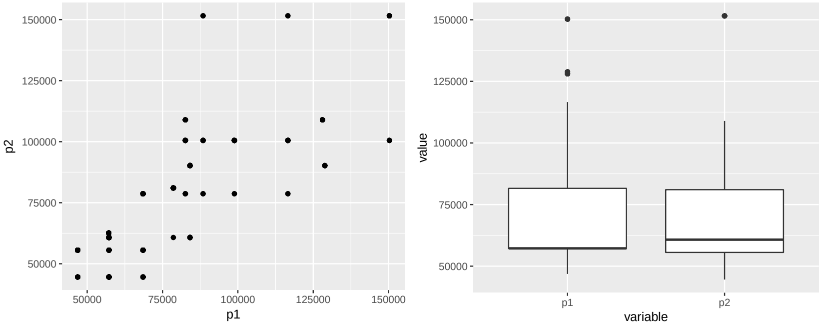
Random Forest

```r
set.seed(2)
tree2 = tree(price ~.,
             data = Housing[sample(1:nrow(Housing),replace=TRUE),])
plot(tree2);  text(tree2)
```



Random Forest

```
pred = data.frame(p1 = predict(tree1, Housing) ,p2 = predict(tree2, Housing))
g1 = ggplot(data = pred) + geom_point(aes(p1, p2))
g2 = ggplot(data = melt(pred)) + geom_boxplot(aes(variable, value))
grid.arrange(g1, g2, ncol = 2)
```



Random Forest

```
library(randomForest)
# = Regression Tree
insamp=sample(1:nrow(Housing),500)
outsamp=setdiff(1:nrow(Housing),insamp)
tree_fs = tree(price ~.,
               data = Housing[insamp,])

# = Random Forest
set.seed(1) # = Seed for replication
rf = randomForest(price ~.,
                  data = Housing[insamp,], nodesize = 10, importance = TRUE)
```

Random Forest

```
# = Calculate MSE
mse_tree = sqrt(mean((predict(tree_fs, Housing[outsamp, ]) -
                        Housing$price[outsamp])^2))
mse_rf = sqrt(mean((predict(rf, Housing[outsamp, ]) -
                      Housing$price[outsamp])^2))

mse_rf/mse_tree
```

```
## [1] 0.7986199
```

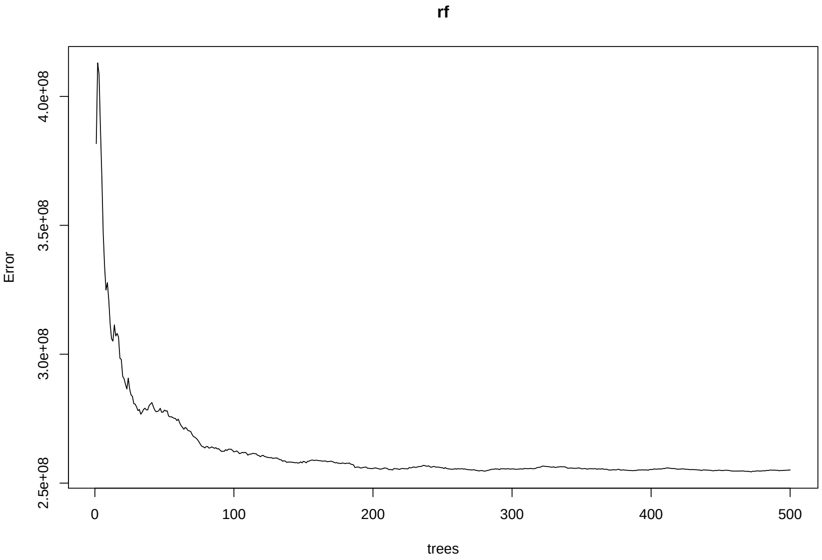
Random Forest

```r
importance(rf)
```

```
##             %IncMSE IncNodePurity
## lotsize  38.987281   91833371227
## bedrooms 18.999051   21343481145
## bathrms  35.276966   49194651344
## stories  23.667206   23954116988
## driveway 17.437953    7089694747
## recroom  11.851130    7108625124
## fullbase 14.564720    8772633419
## gashw     3.939987    5436730391
## airco    22.258920   29073690740
## garagepl 14.813148   24827758850
## prefarea 18.183930   14785925617
```

Random Forest

```r
plot(rf)
```



Random Forest

```
tr=getTree(rf,1)
tr[1:15,]
```

```
##    left daughter right daughter split var split point status prediction
## 1              2              3        11         1.0     -3   68368.86
## 2              4              5         9         1.0     -3   63315.81
## 3              6              7         9         1.0     -3   86494.95
## 4              8              9         5         1.0     -3   56660.07
## 5             10             11         1      5974.0     -3   78528.92
## 6             12             13         3         1.5     -3   74863.39
## 7             14             15         4         1.5     -3   98784.91
## 8             16             17         2         3.5     -3   45776.18
## 9             18             19        10         0.5     -3   59418.66
## 10            20             21        10         0.5     -3   67730.88
## 11            22             23         4         3.5     -3   92926.31
## 12            24             25         1      5450.0     -3   69273.17
## 13            26             27         4         1.5     -3   90143.33
## 14            28             29         7         1.0     -3   79872.73
## 15            30             31         2         3.5     -3  112206.45
```

Random Forest

- Parâmetros:

```
rf1 = randomForest(price ~.,
                   data = Housing[insamp,],
                   nodesize = 25, ntree=1000, mtry=7)
mse_rf1 = sqrt(mean((predict(rf1, Housing[outsamp, ]) -
                        Housing$price[outsamp])^2))
mse_rf/mse_rf1
```

```
## [1] 0.9387345
```

Boosting

- The xgboost package is the best Boosting package in R.
- It does not accept string variables.

```
library(xgboost)
Housing$driveway=ifelse(Housing$driveway=="yes",1,0)
Housing$recroom=ifelse(Housing$recroom=="yes",1,0)
Housing$fullbase=ifelse(Housing$fullbase=="yes",1,0)
Housing$gashw=ifelse(Housing$gashw=="yes",1,0)
Housing$airco=ifelse(Housing$prefarea=="yes",1,0)
Housing$prefarea=ifelse(Housing$prefarea=="yes",1,0)
```

Boosting

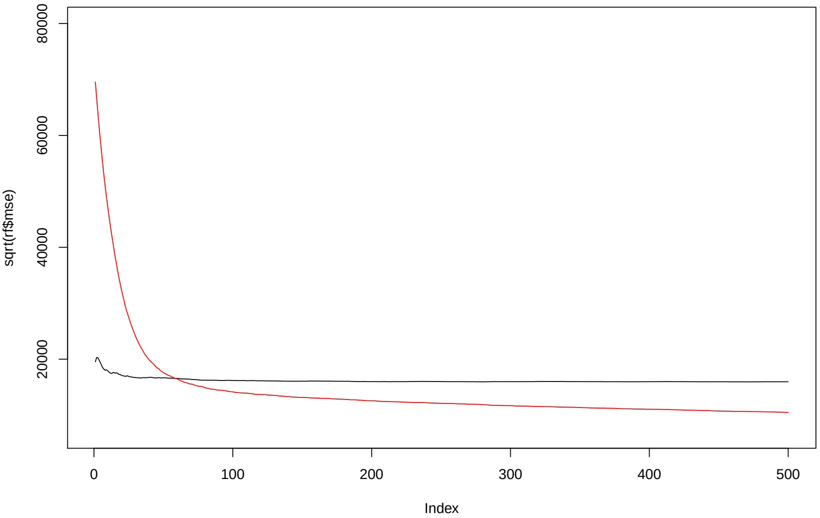
- Boosting has more parameters to tune than Random Forests,
- It is also more sentive to the parameters,
- The Boosting trees are smaller, which makes the algorithm faster.

```
y=Housing$price
x=as.matrix(Housing[,-1])
xgb=xgboost(x[insamp,],label = y[insamp],nrounds = 500,
            params=list(eta=0.05,colsample_bytree=1/3,
                        subsample=0.5,max_depth=4,
                        min_child_weigth=5))
```

```
## [1]  train-rmse:69552.265625
## [2]  train-rmse:66458.945312
## [3]  train-rmse:63567.464844
## [4]  train-rmse:60744.230469
## [5]  train-rmse:58109.117188
## [6]  train-rmse:55612.960938
## [7]  train-rmse:53253.890625
## [8]  train-rmse:51042.558594
## [9]  train-rmse:48997.394531
## [10] train-rmse:47070.867188
## [11] train-rmse:45213.398438
## [12] train-rmse:43488.046875
## [13] train-rmse:41873.070312
## [14] train-rmse:40309.351562
## [15] train-rmse:38669.523438
```

Boosting

```
plot(sqrt(rf$mse),type="l",ylim=c(7000,80000))
lines(xgb$evaluation_log[,2],col=2)
```



Boosting

```
mse_xgb = sqrt(mean((predict(xgb, x[outsamp, ]) - y[outsamp])^2))
mse_xgb
```

```
## [1] 18238.56
```

```
mse_rf
```

```
## [1] 15905.59
```

Boosting - examplo 2

- This second example is also on a Housing dataset. But in this case the dataset is bigger (5000 observations).
- Houses sold in Melbourne in 2016 and 2017.

Boosting - examplo 2

```
load("housing2.rda")
data$Distance=as.numeric(as.character(data$Distance))
x=as.matrix(data[,-2])
y=data[,2]
set.seed(1)
insamp=sample(1:nrow(data),3000)
outsamp=setdiff(1:nrow(data),insamp)
```

Boosting - exemplo 2

```
rf=randomForest(x[insamp,],y[insamp])

xgb=xgboost(data = x[insamp,],label = y[insamp],nrounds = 1000,
            params=list(eta=0.05,colsample_bytree=2/3,
                        max_depth=4,min_child_weigth=5))
```

```
## [1]  train-rmse:1389774.000000
## [2]  train-rmse:1330780.250000
## [3]  train-rmse:1274018.250000
## [4]  train-rmse:1219418.000000
## [5]  train-rmse:1168791.875000
## [6]  train-rmse:1120952.875000
## [7]  train-rmse:1074249.500000
## [8]  train-rmse:1030829.312500
## [9]  train-rmse:990472.250000
## [10] train-rmse:953062.937500
## [11] train-rmse:915938.437500
## [12] train-rmse:881595.812500
## [13] train-rmse:849436.812500
## [14] train-rmse:818364.062500
## [15] train-rmse:789748.562500
## [16] train-rmse:761661.687500
## [17] train-rmse:735612.562500
## [18] train-rmse:710991.187500
## [19] train-rmse:686907.437500
```

Boosting - exemplo 2

```
mse_xgb = sqrt(mean((predict(xgb, x[outsamp, ]) - y[outsamp])^2))
mse_rf = sqrt(mean((predict(rf, x[outsamp, ]) - y[outsamp])^2))
mse_rf
```

```
## [1] 322670.7
```

```
mse_xgb
```

```
## [1] 304297.3
```

Artificial Neural Networks: the h2o package

- Dados de preços de casas explicados por número de quartos, latitude, longitude, número de banheiros, etc. Total de 8 variáveis explicativas.
- 5814 observações. 3000 de treino e 2814 de teste

```
library(h2o)
load("housing2.rda")
data$Distance=as.numeric(as.character(data$Distance))

set.seed(1)
training = sample(1:nrow(data),3000)
test = setdiff(1:nrow(data),training)
```

Artificial Neural Networks: the h2o package

- O pacote h2o precisa ser iniciado em uma seção separada exclusiva

```
h2o.init(nthreads = -1)
```

Artificial Neural Networks: the h2o package

- Arguments:
  - y: Name of the response variable,
  - training_frame: data.frame with all variables,
  - activation: Name of the activation function,
  - epochs: How many times the algorithm goes through the data,
  - train_samples_per_iteration: Number of observations evaluated per iteration,
  - seed: seed for replication.

```
model = h2o.deeplearning(y = 'Price',
                         training_frame = as.h2o(data[training,]),
                         activation = 'Rectifier',
                         hidden = c(10,10,10,10),
                         epochs = 100,
                         train_samples_per_iteration = -2,
                         seed = 1)
```

Artificial Neural Networks: the h2o package

```
# = Calcular previsão
y_pred = h2o.predict(model, newdata = as.h2o(data[test,]))
y_pred = as.vector(y_pred)

## Fechar seção.
h2o.shutdown()
```

XGBoost as benchmark

```r
library(xgboost)
dataxgb=as.matrix(data)
set.seed(1)
xgb = xgboost(dataxgb[training,-2],label = dataxgb[training,2],
              nrounds = 1000,
              params=
               list(eta=0.05,nthread=1,colsample_bytree=2/3,
                    subsample=1,max_depth=4,
                    min_child_weigth=length(training)/200))

y_pred_xgb = predict(xgb,dataxgb[test,-2])
```

Results

```r
ann_e = sqrt(mean((y_pred-data$Price[test])^2))
xgb_e = sqrt(mean((y_pred_xgb-data$Price[test])^2))
c(ann=ann_e,xgb=xgb_e)
```

```
##      ann      xgb
## 342720.7 304202.4
```