The beauty of trees

Statistical Methods for Data Science

Federico Pigozzi

Letícia Negrão Pinto

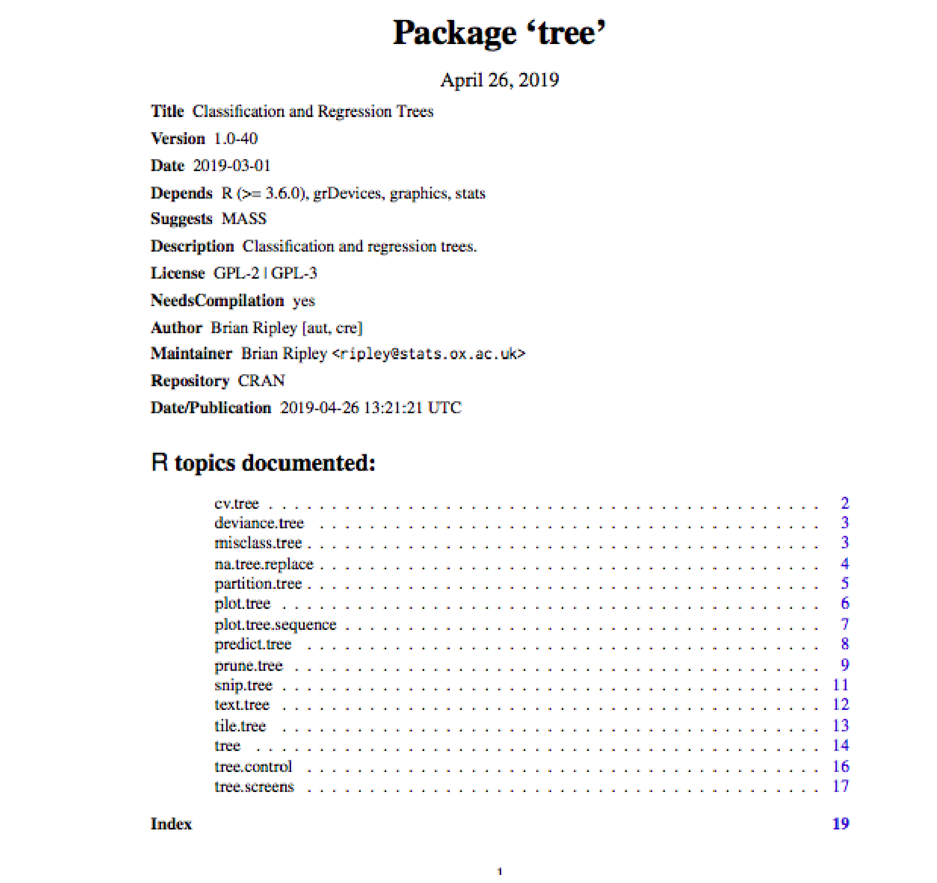
Eduardo Gonnelli

DSSC - 2019

tree R package

Classification trees

Regression trees



Package 'tree'

April 26, 2019

Title Classification and Regression Trees

Version 1.0-40

Date 2019-03-01

Depends R (>= 3.6.0), grDevices, graphics, stats

Suggests MASS

Description Classification and regression trees.

License GPL-2 | GPL-3

NeedsCompilation yes

Author Brian Ripley [aut, cre]

Maintainer Brian Ripley `<ripley@stats.ox.ac.uk>`

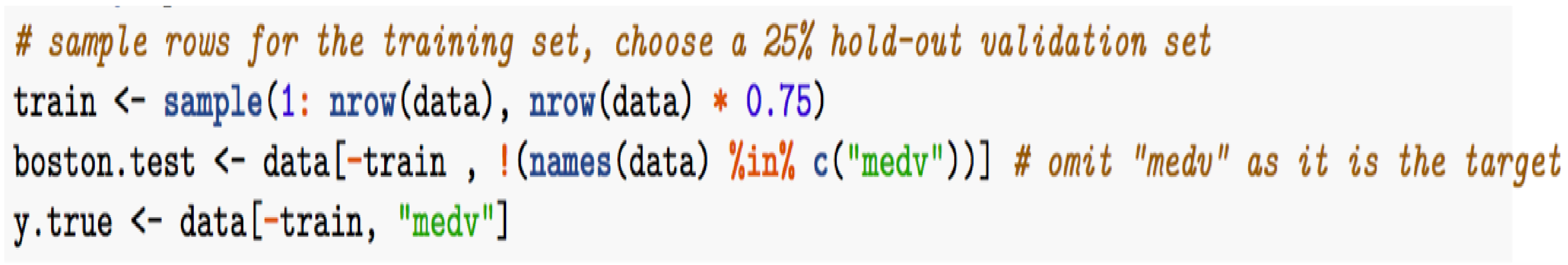
Repository CRAN

Date/Publication 2019-04-26 13:21:21 UTC

R topics documented:

1

Regression trees - Boston data set



```r
# sample rows for the training set, choose a 25% hold-out validation set
train <- sample(1: nrow(data), nrow(data) * 0.75)
boston.test <- data[-train , !(names(data) %in% c("medv"))] # omit "medv" as it is the target
y.true <- data[-train, "medv"]
```

Building a Tree

- Recursive binary splitting
- Impurity measures: deviance (default), Gini

```
tree.boston <- tree(medv~., data, subset=train) # all variables, subset train
```

```
pred.tree <- predict(tree.boston, newdata=boston.test, type="vector")
RMSE(pred.tree, y.true)
```

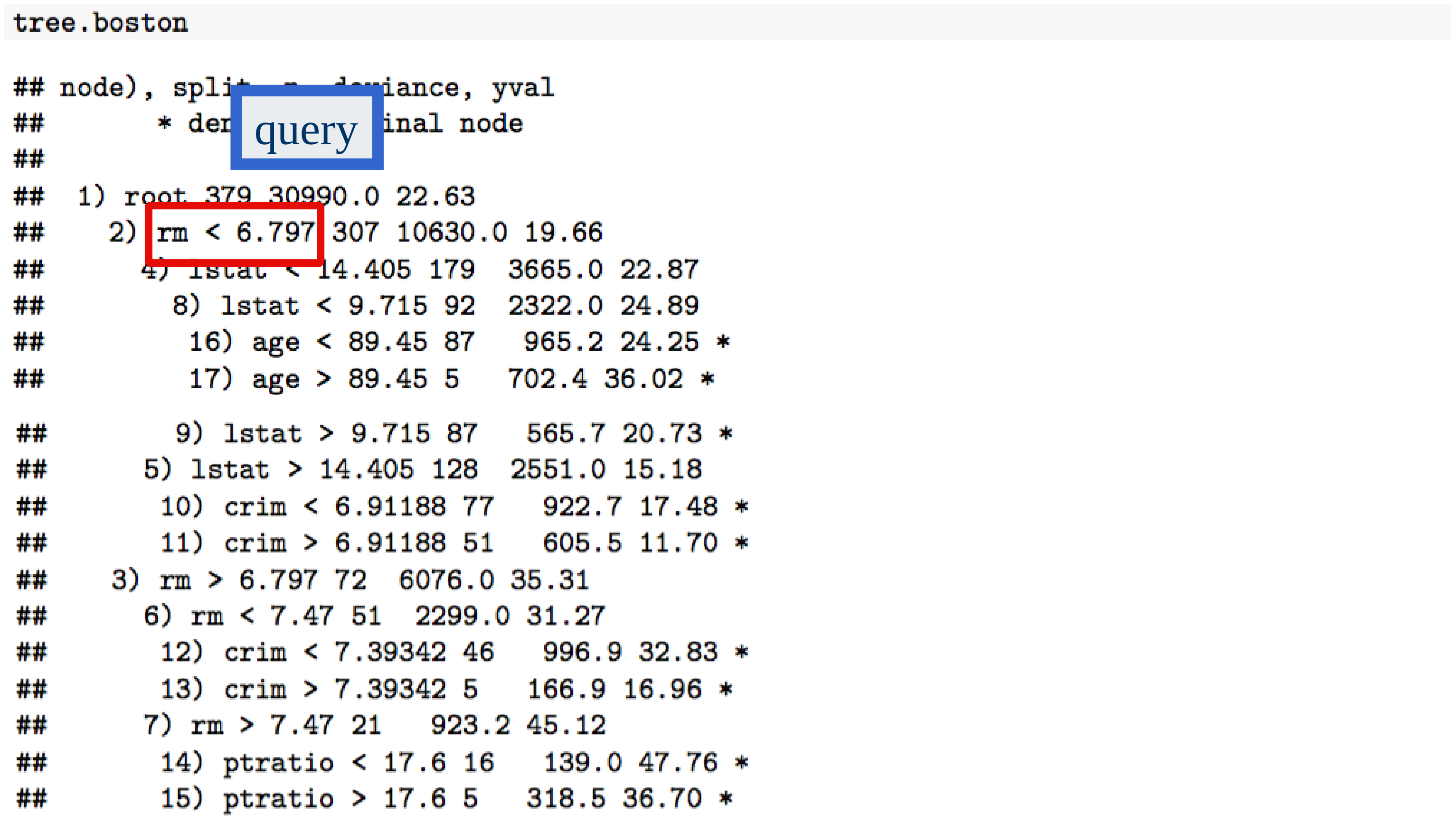
```
## [1] 4.262707
```

Visualize Tree Information - 1

```
summary(tree.boston)
```

```
##
## Regression tree:
## tree(formula = medv ~ ., data = data, subset = train)
## Variables actually used in tree construction:
## [1] "rm"      "lstat"   "age"     "crim"    "ptratio"
## Number of terminal nodes:  9
## Residual mean deviance:  14.55 = 5383 / 370
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -14.8000  -2.0910  -0.1253   0.0000   2.0730  17.1700
```

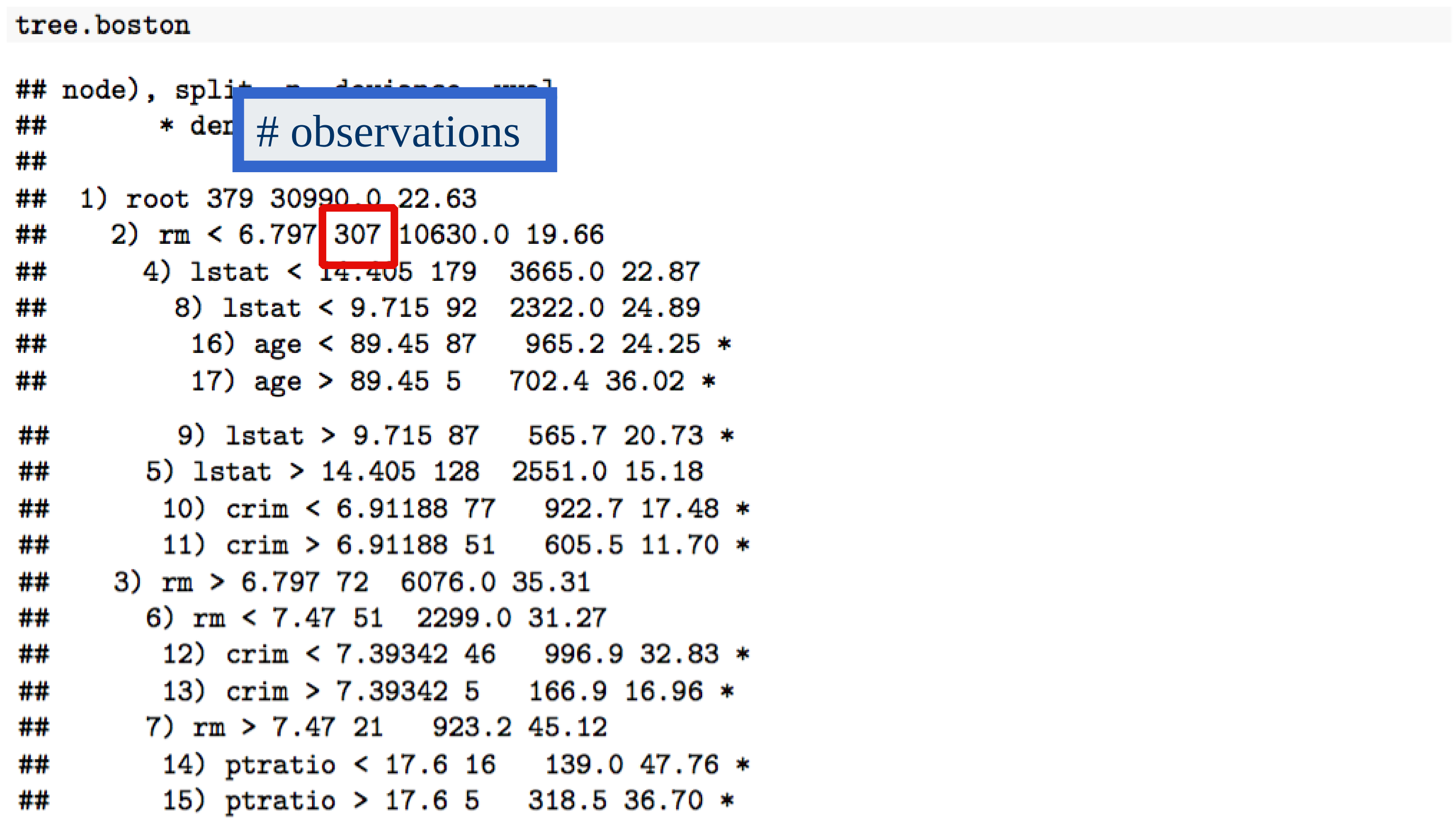
Visualizing Tree Information - 2



```
tree.boston
```

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 379 30990.0 22.63
##    2) rm < 6.797 307 10630.0 19.66
##      4) lstat < 14.405 179  3665.0 22.87
##        8) lstat < 9.715 92  2322.0 24.89
##         16) age < 89.45 87   965.2 24.25 *
##         17) age > 89.45 5   702.4 36.02 *
##        9) lstat > 9.715 87   565.7 20.73 *
##      5) lstat > 14.405 128  2551.0 15.18
##       10) crim < 6.91188 77   922.7 17.48 *
##       11) crim > 6.91188 51   605.5 11.70 *
##    3) rm > 6.797 72  6076.0 35.31
##      6) rm < 7.47 51  2299.0 31.27
##       12) crim < 7.39342 46   996.9 32.83 *
##       13) crim > 7.39342 5   166.9 16.96 *
##      7) rm > 7.47 21   923.2 45.12
##       14) ptratio < 17.6 16   139.0 47.76 *
##       15) ptratio > 17.6 5   318.5 36.70 *
```

Visualizing Tree Information - 2

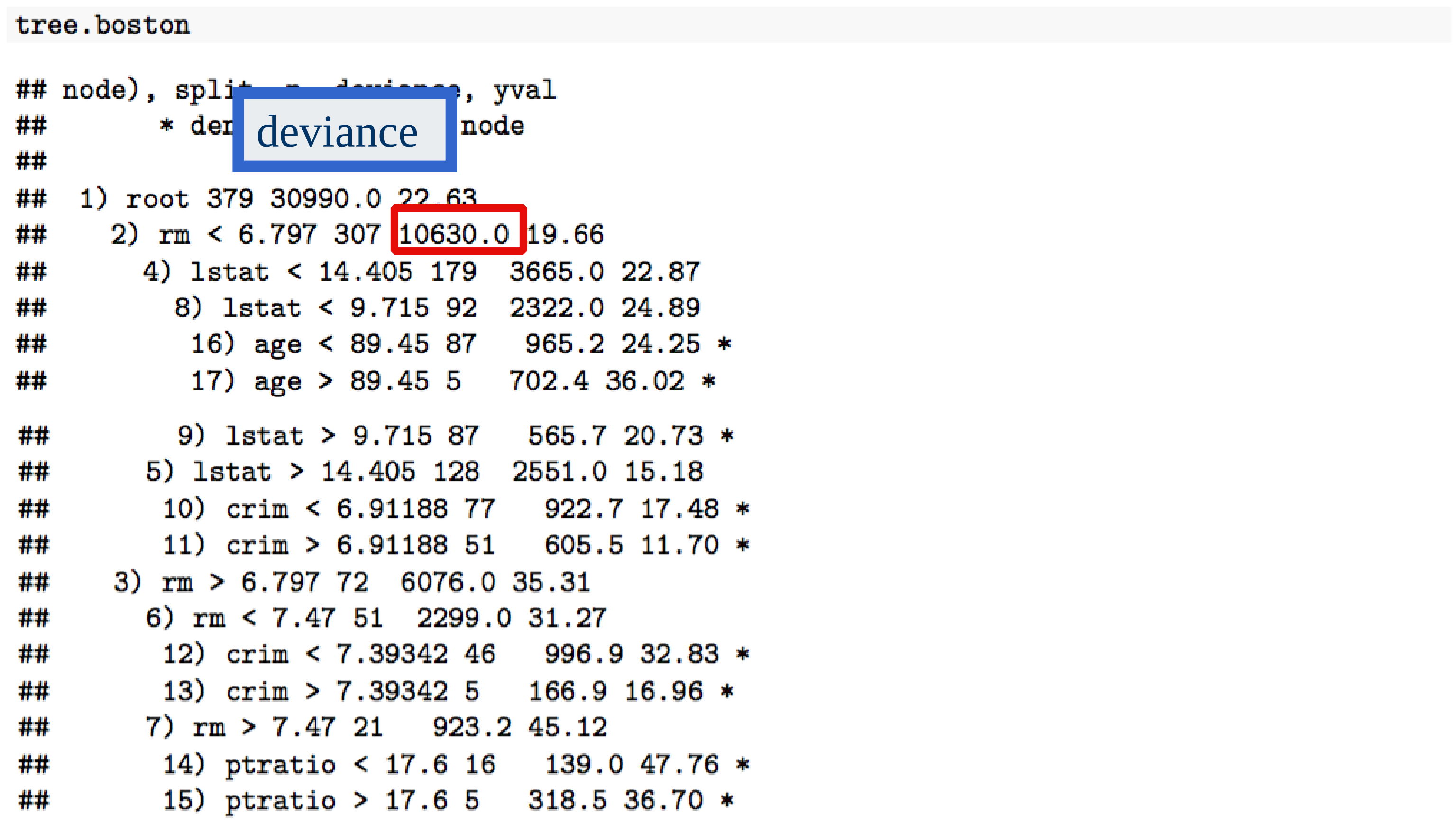


```
tree.boston
```

```
## node), spli
##        * der
##
##  1) root 379 30990.0 22.63
##    2) rm < 6.797 307 10630.0 19.66
##      4) lstat < 14.405 179  3665.0 22.87
##        8) lstat < 9.715 92  2322.0 24.89
##         16) age < 89.45 87   965.2 24.25 *
##         17) age > 89.45 5   702.4 36.02 *
##        9) lstat > 9.715 87   565.7 20.73 *
##      5) lstat > 14.405 128  2551.0 15.18
##       10) crim < 6.91188 77   922.7 17.48 *
##       11) crim > 6.91188 51   605.5 11.70 *
##    3) rm > 6.797 72  6076.0 35.31
##      6) rm < 7.47 51  2299.0 31.27
##       12) crim < 7.39342 46   996.9 32.83 *
##       13) crim > 7.39342 5   166.9 16.96 *
##      7) rm > 7.47 21   923.2 45.12
##       14) ptratio < 17.6 16   139.0 47.76 *
##       15) ptratio > 17.6 5   318.5 36.70 *
```

# observations

Visualizing Tree Information - 2



```
tree.boston
```

```
## node), split, n, deviance, yval
##        * den deviance node
##
##  1) root 379 30990.0 22.63
##    2) rm < 6.797 307 10630.0 19.66
##      4) lstat < 14.405 179  3665.0 22.87
##        8) lstat < 9.715 92  2322.0 24.89
##         16) age < 89.45 87   965.2 24.25 *
##         17) age > 89.45 5   702.4 36.02 *
##        9) lstat > 9.715 87   565.7 20.73 *
##      5) lstat > 14.405 128  2551.0 15.18
##       10) crim < 6.91188 77   922.7 17.48 *
##       11) crim > 6.91188 51   605.5 11.70 *
##    3) rm > 6.797 72  6076.0 35.31
##      6) rm < 7.47 51  2299.0 31.27
##       12) crim < 7.39342 46   996.9 32.83 *
##       13) crim > 7.39342 5   166.9 16.96 *
##      7) rm > 7.47 21   923.2 45.12
##       14) ptratio < 17.6 16   139.0 47.76 *
##       15) ptratio > 17.6 5   318.5 36.70 *
```

Visualize Tree Information - 2

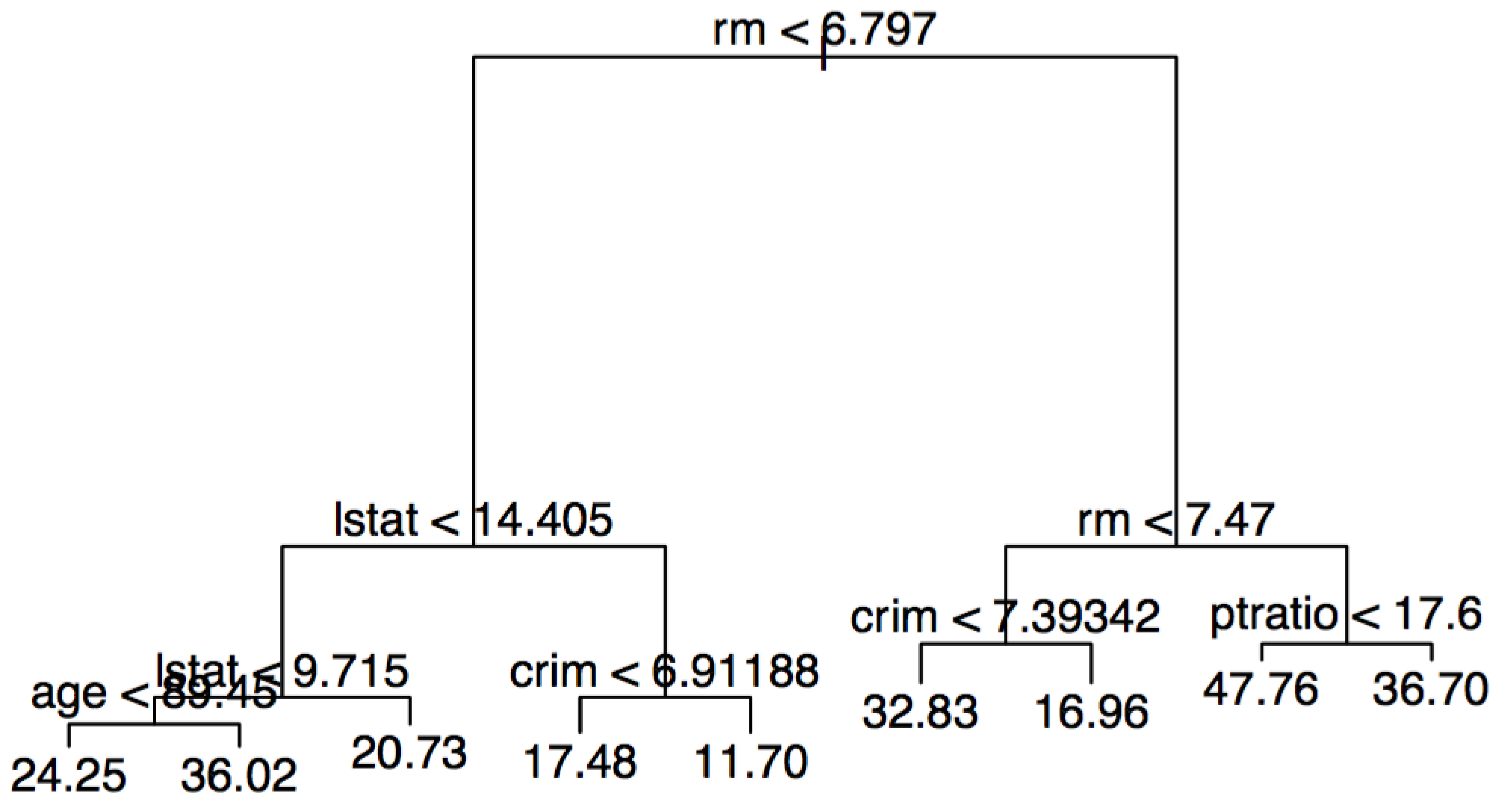
```
tree.boston
```

mean value (class probs. for classification)

```
## node), spli
##        * der
##
##  1) root 379 30990.0 22.63
##    2) rm < 6.797 307 10630.0 19.66
##      4) lstat < 14.405 179  3665.0 22.87
##        8) lstat < 9.715 92  2322.0 24.89
##         16) age < 89.45 87   965.2 24.25 *
##         17) age > 89.45 5   702.4 36.02 *
##        9) lstat > 9.715 87   565.7 20.73 *
##      5) lstat > 14.405 128  2551.0 15.18
##       10) crim < 6.91188 77   922.7 17.48 *
##       11) crim > 6.91188 51   605.5 11.70 *
##    3) rm > 6.797 72  6076.0 35.31
##      6) rm < 7.47 51  2299.0 31.27
##       12) crim < 7.39342 46   996.9 32.83 *
##       13) crim > 7.39342 5   166.9 16.96 *
##      7) rm > 7.47 21   923.2 45.12
##       14) ptratio < 17.6 16   139.0 47.76 *
##       15) ptratio > 17.6 5   318.5 36.70 *
```

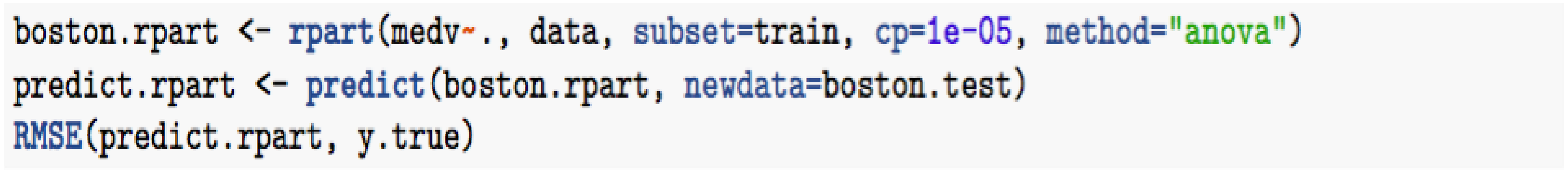
Visualize Tree Information - 3

```
# let's produce an annotated tree
plot(tree.boston)
text(tree.boston, pretty=0)
```



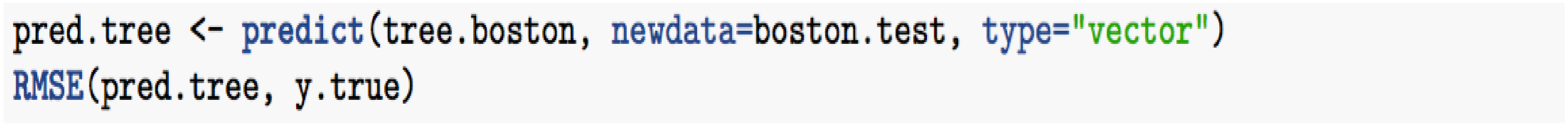
Building a Tree - rpart

- Open-source implementation of CART (Breiman et al. 1983)
- Same basic signature as tree



```
boston.rpart <- rpart(medv~., data, subset=train, cp=1e-05, method="anova")
predict.rpart <- predict(boston.rpart, newdata=boston.test)
RMSE(predict.rpart, y.true)
```

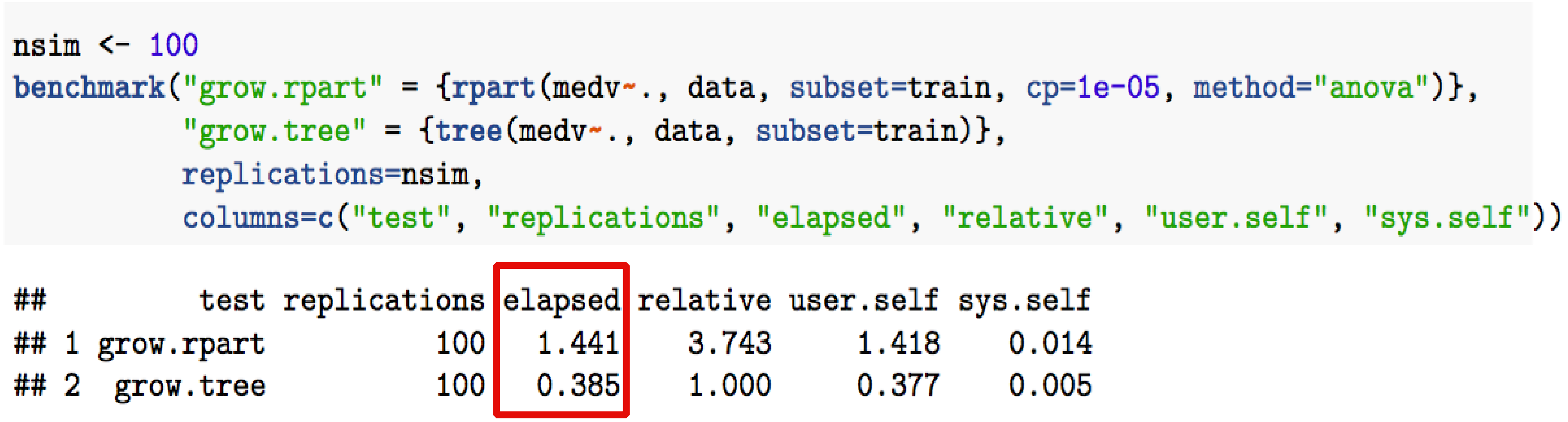
```
## [1] 4.118872
```



```
pred.tree <- predict(tree.boston, newdata=boston.test, type="vector")
RMSE(pred.tree, y.true)
```

```
## [1] 4.262707
```

Time Comparison



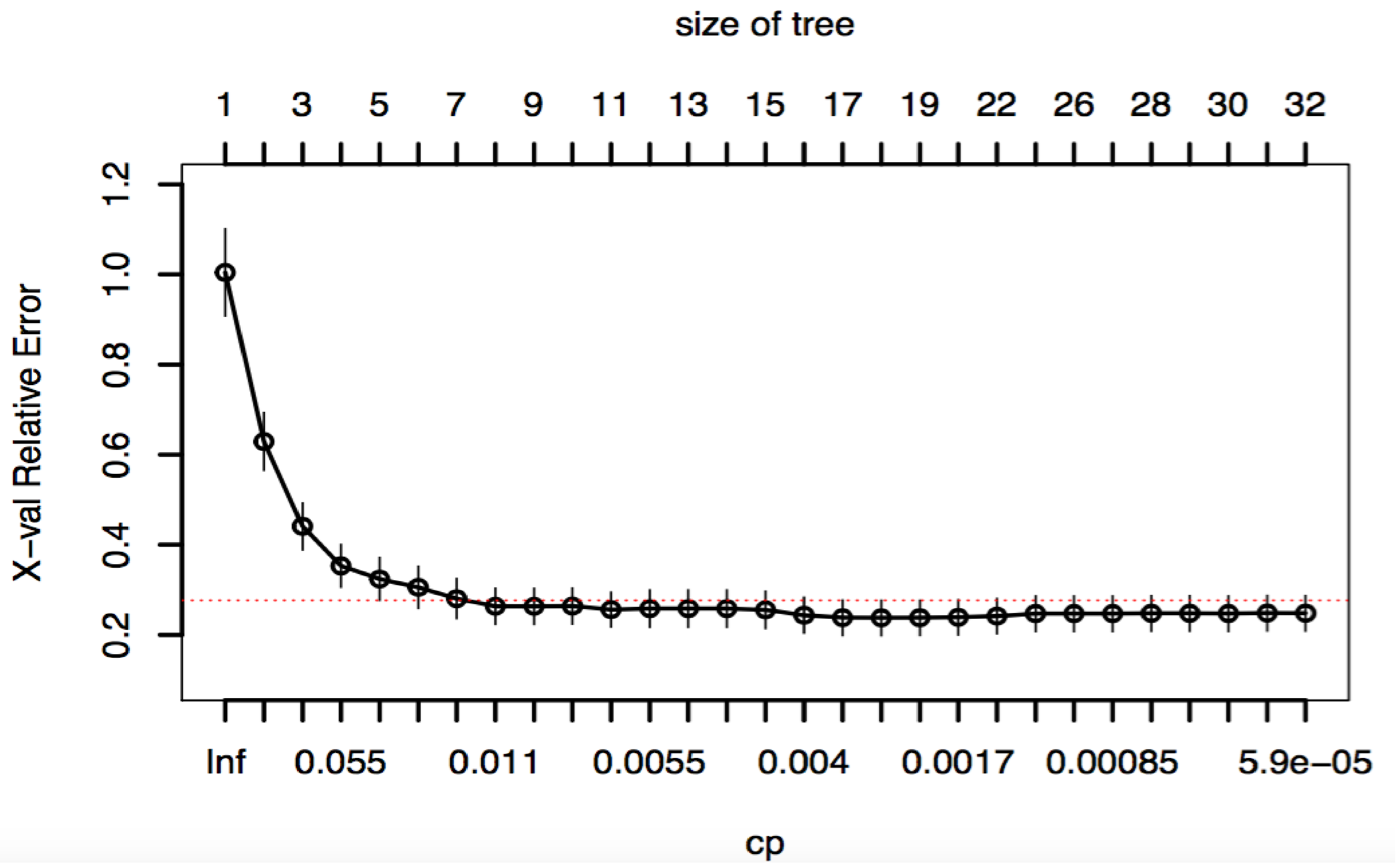
```r
nsim <- 100
benchmark("grow.rpart" = {rpart(medv~., data, subset=train, cp=1e-05, method="anova")},
          "grow.tree" = {tree(medv~., data, subset=train)},
          replications=nsim,
          columns=c("test", "replications", "elapsed", "relative", "user.self", "sys.self"))
```

```
##         test replications elapsed relative user.self sys.self
## 1 grow.rpart          100   1.441    3.743     1.418    0.014
## 2  grow.tree          100   0.385    1.000     0.377    0.005
```

Reason: rpart performs cross-validation

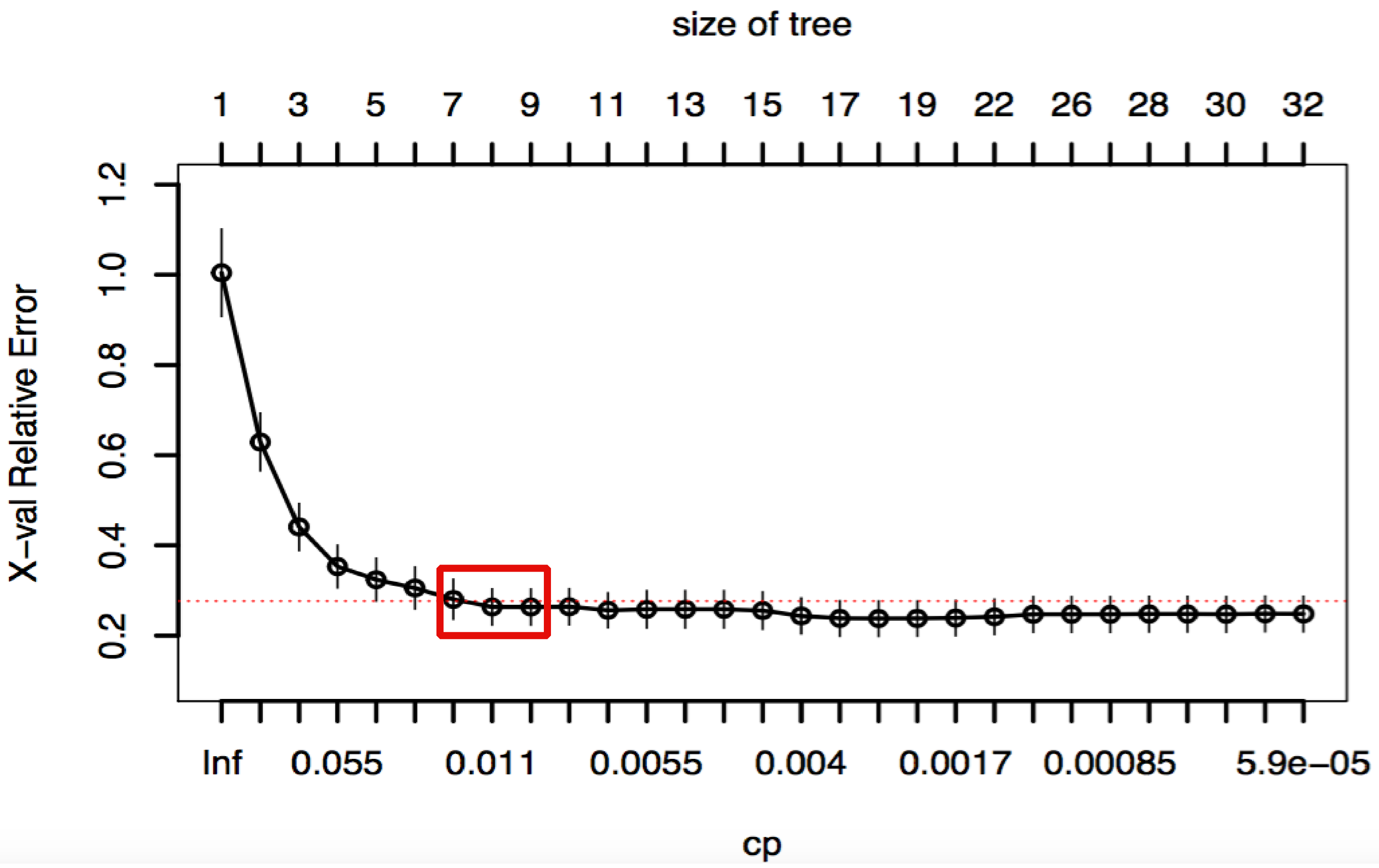
rpart – visualizing cv results

```
plotcp(boston.rpart, minline=TRUE, col="red", lwd=2)
```

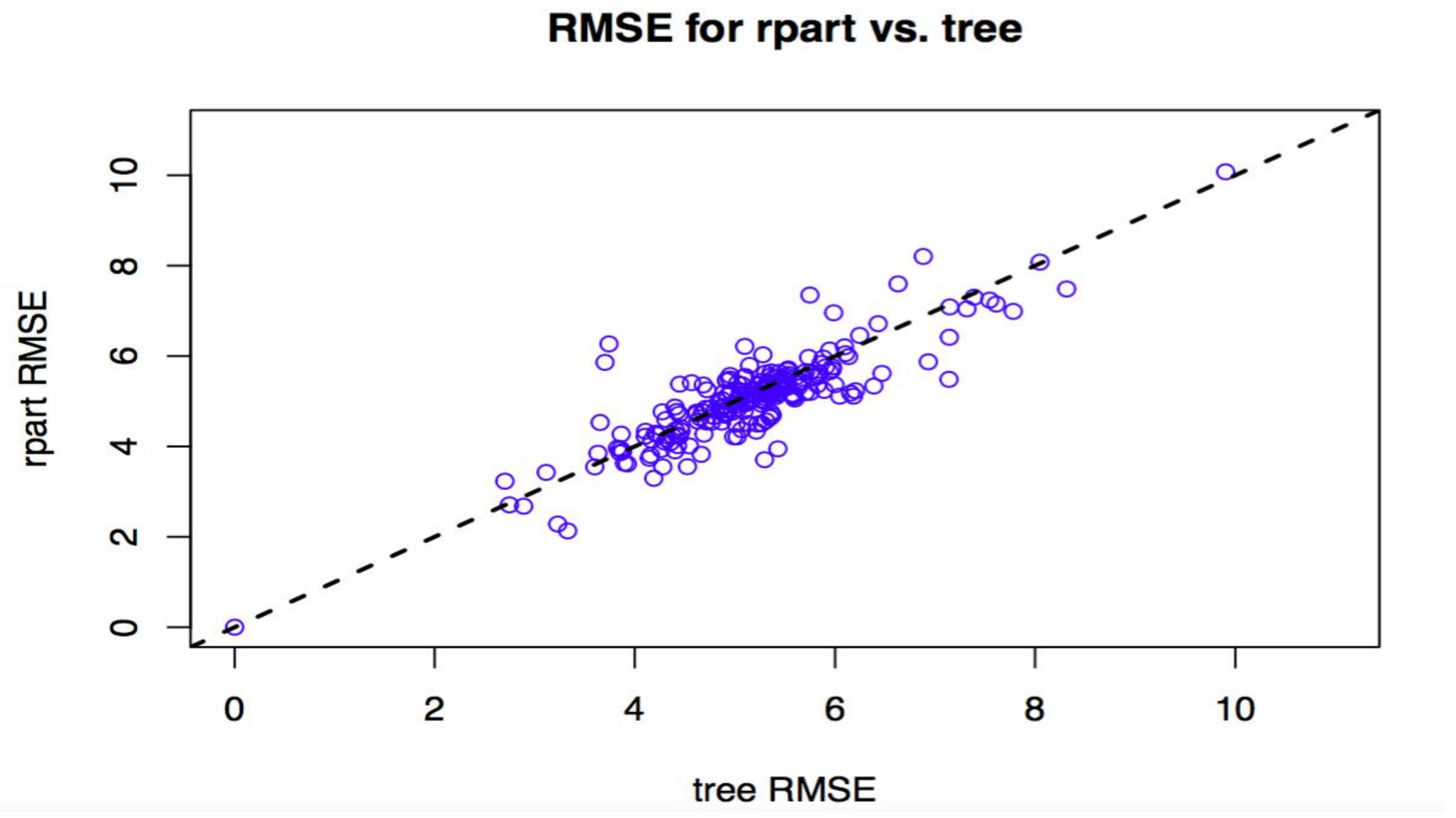


rpart – visualizing cv results

```
plotcp(boston.rpart, minline=TRUE, col="red", lwd=2)
```



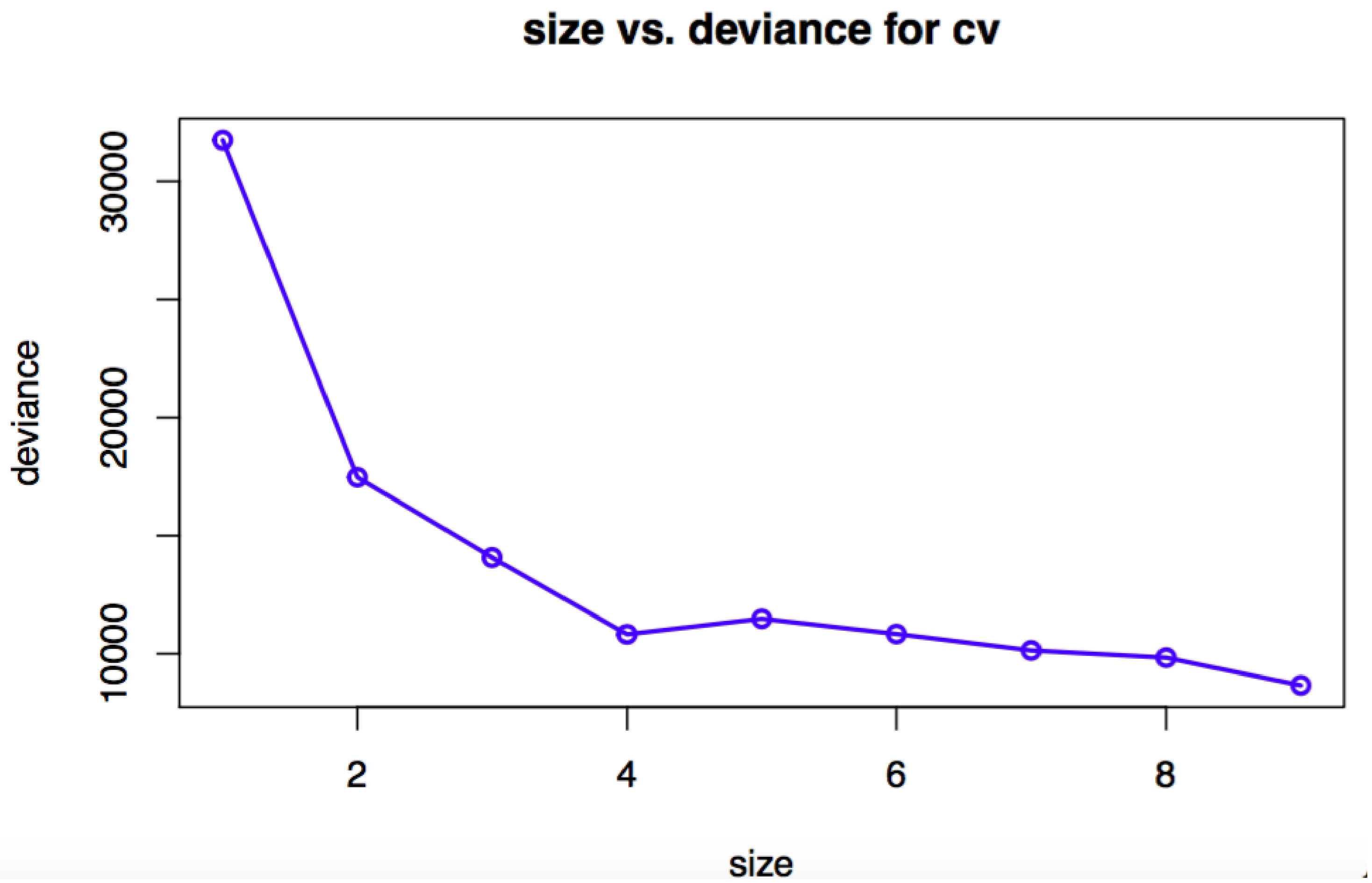
Performance Comparison



tree – cross-validation

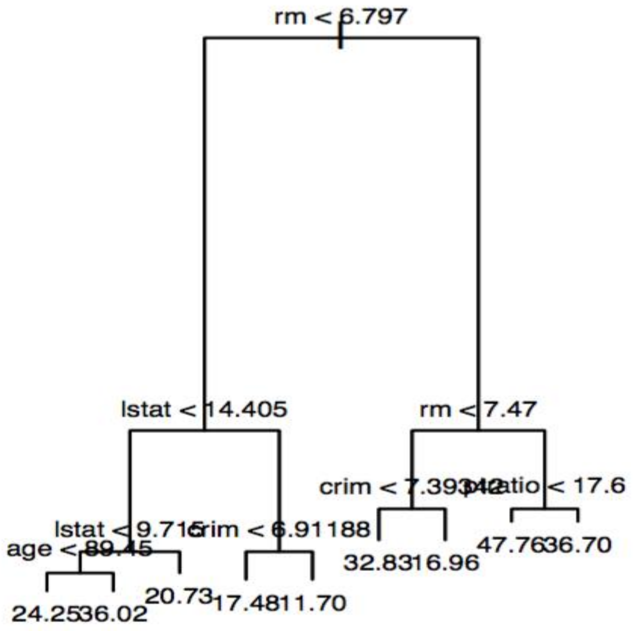
```
cv.boston <- cv.tree(tree.boston)
```

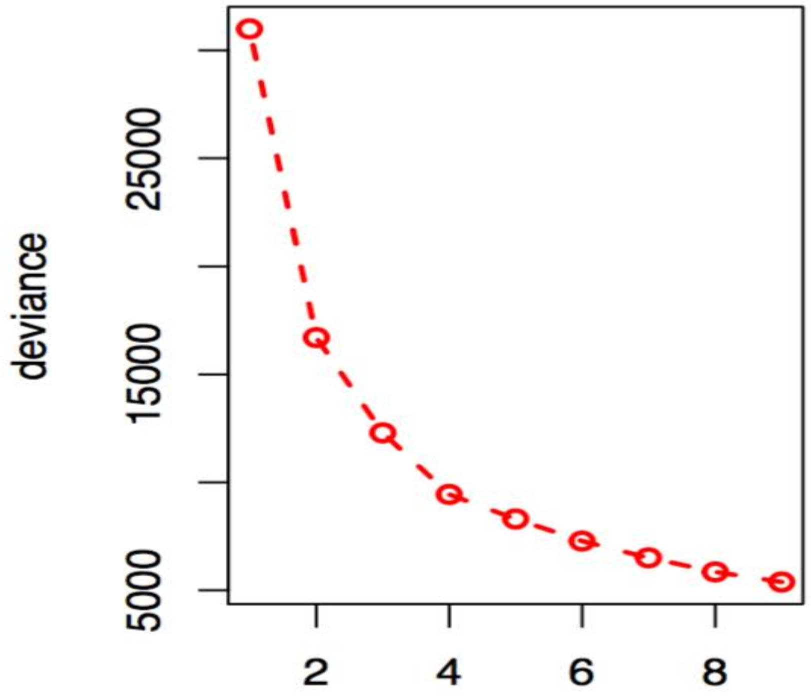
```
plot(cv.boston$size, cv.boston$dev, type='o', col=4, lwd=2, main="size vs. deviance f
```



tree – pruning

```
prune.tree <- prune.tree(tree.boston, best=9)
```



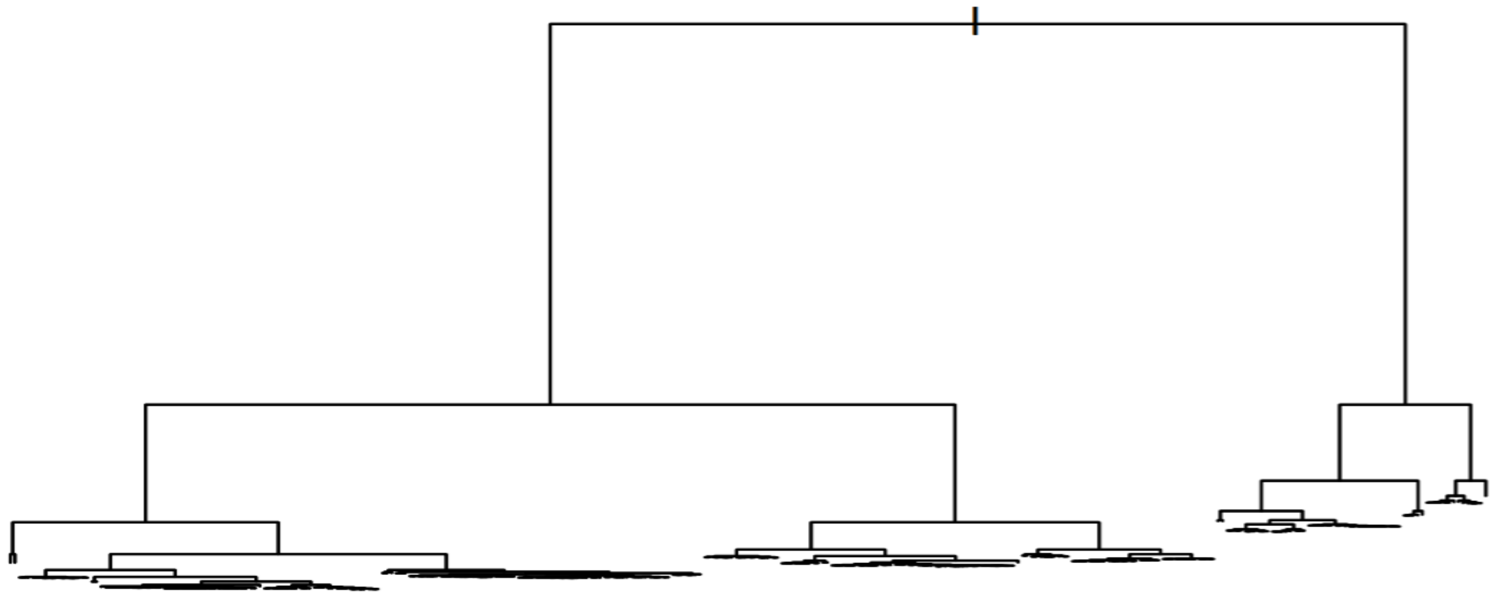


```
predict.tree <- predict(prune.tree, newdata=boston.test)
RMSE(predict.tree, y.true)
```

```
## [1] 4.262707
```

Pre-Pruning - tree

- mincut (min # obs. per node)
- minsize (max # obs. per node)
- mindev (within-node deviance must be at least this times that of the root for the node to be split)
- overfitting tree:



Pre-Pruning - rpart

- minsplit (min # obs. per node)
- minbucket (max # obs. per node)
- maxdepth (of the tree)
- no check on deviance, but yes on depth

“partition.tree”

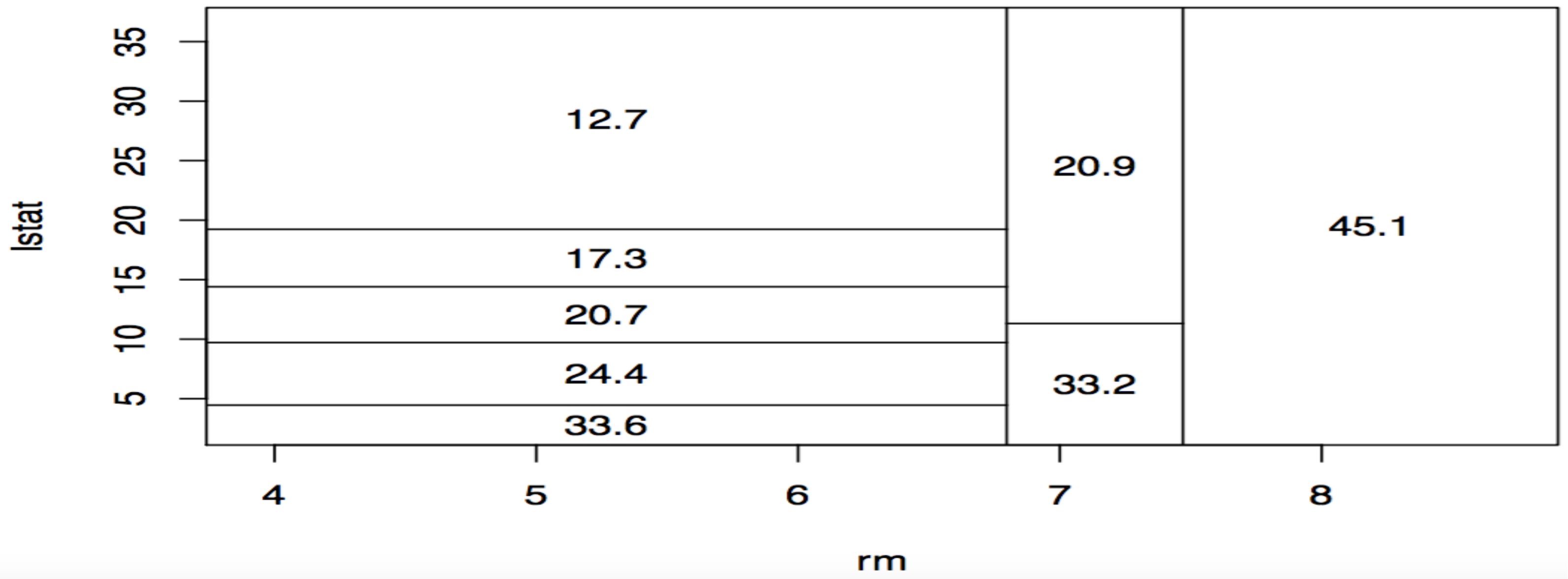
```
keep <- c("lstat", "rm", "medv")
data2 <- data[, names(data) %in% keep]
nrow(data2)
```

```
## [1] 506
```

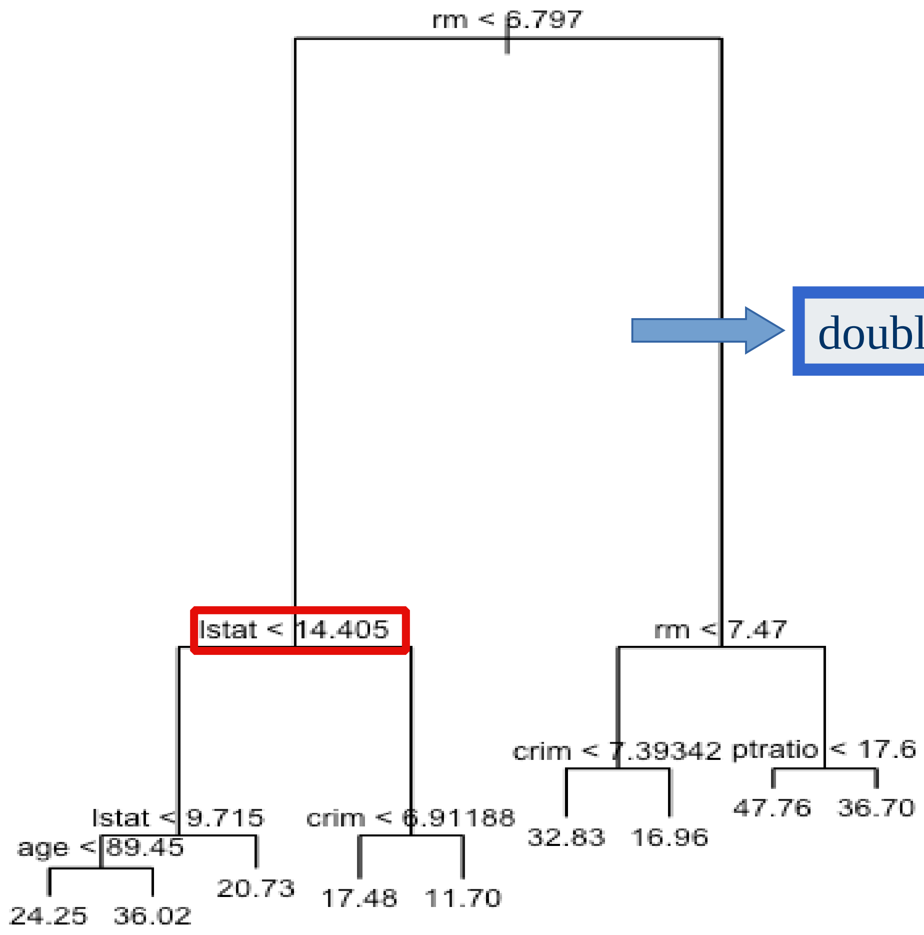
```
names(data2)
```

```
## [1] "rm"     "lstat" "medv"
```

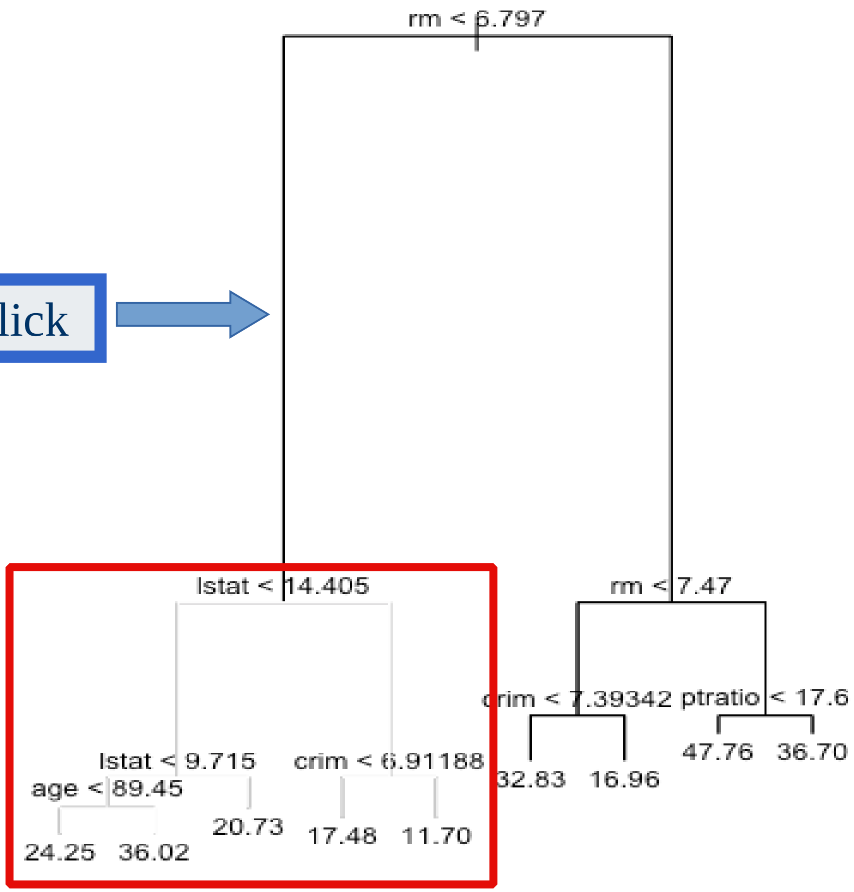
```
new.tree <- tree(medv~., data2, subset=train)
partition.tree(new.tree)
```



“snip.tree”



double click



Handling Missing Values

rpart:

- surrogate variables

tree:

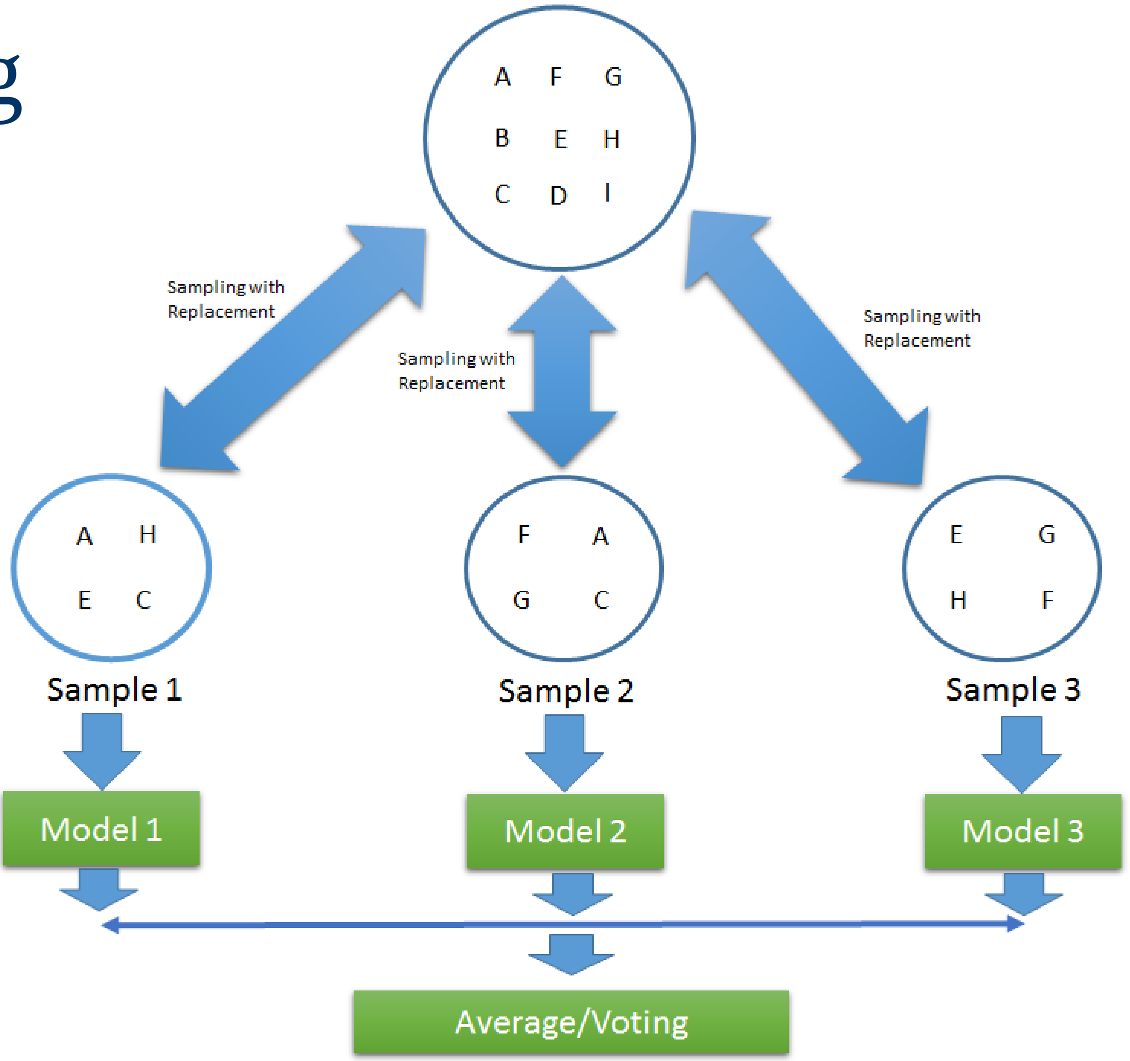
- drop as far as possible in the tree

In any case, see “na.action” argument

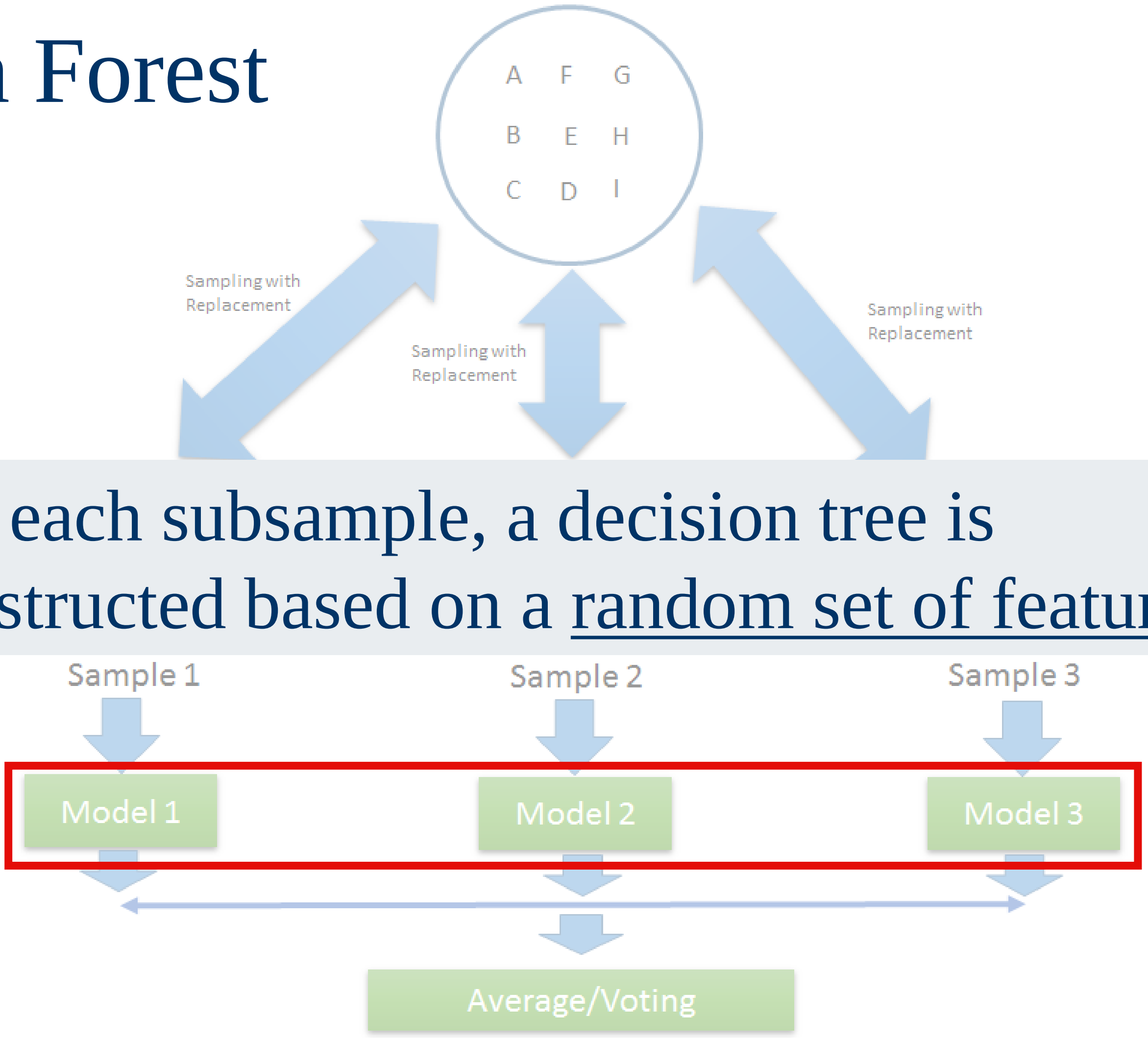
Bagging and Random Forest

in R

Bagging


A F G
B E H
C D I
Sampling with Replacement
Sampling with Replacement
Sampling with Replacement
A H
E C
Sample 1
F A
G C
Sample 2
E G
H F
Sample 3
Model 1
Model 2
Model 3
Average/Voting

Random Forest


A F G
B E H
C D I
Sampling with Replacement
Sampling with Replacement
Sampling with Replacement
Sample 1
Sample 2
Sample 3
Model 1
Model 2
Model 3
Average/Voting

For each subsample, a decision tree is constructed based on a random set of features

Random Forest

Decorrelates the trees in the forest

Bagging x Random Forest

The main difference is the features considered in each tree

randomForest R package

Bagging
Random Forest

Regression
Classification

Package 'randomForest'

March 25, 2018

Title Breiman and Cutler's Random Forests for Classification and Regression

Version 4.6-14

Date 2018-03-22

Depends R (>= 3.2.2), stats

Suggests RColorBrewer, MASS

Author Fortran original by Leo Breiman and Adele Cutler, R port by Andy Liaw and Matthew Wiener.

Description Classification and regression based on a forest of trees using random inputs, based on Breiman (2001) <DOI:10.1023/A:1010933404324>.

Maintainer Andy Liaw <andy_liaw@merck.com>

License GPL (>= 2)

URL https://www.stat.berkeley.edu/~breiman/RandomForests/

NeedsCompilation yes

Repository CRAN

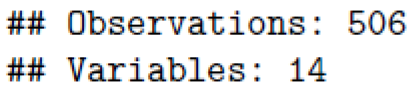
Date/Publication 2018-03-25 15:00:24 UTC

R topics documented:

1

randomForest package - Regression Boston data set

```
library(randomForest)
library(MASS)
library(dplyr)
set.seed (2019)

# data
data <- Boston
glimpse(data)

# Train set
train = sample (1: nrow(data), nrow(data)*0.75)

# Test set
boston.test = data[-train ,"medv"]
```



```
## Observations: 506
## Variables: 14
```

randomForest package - Regression

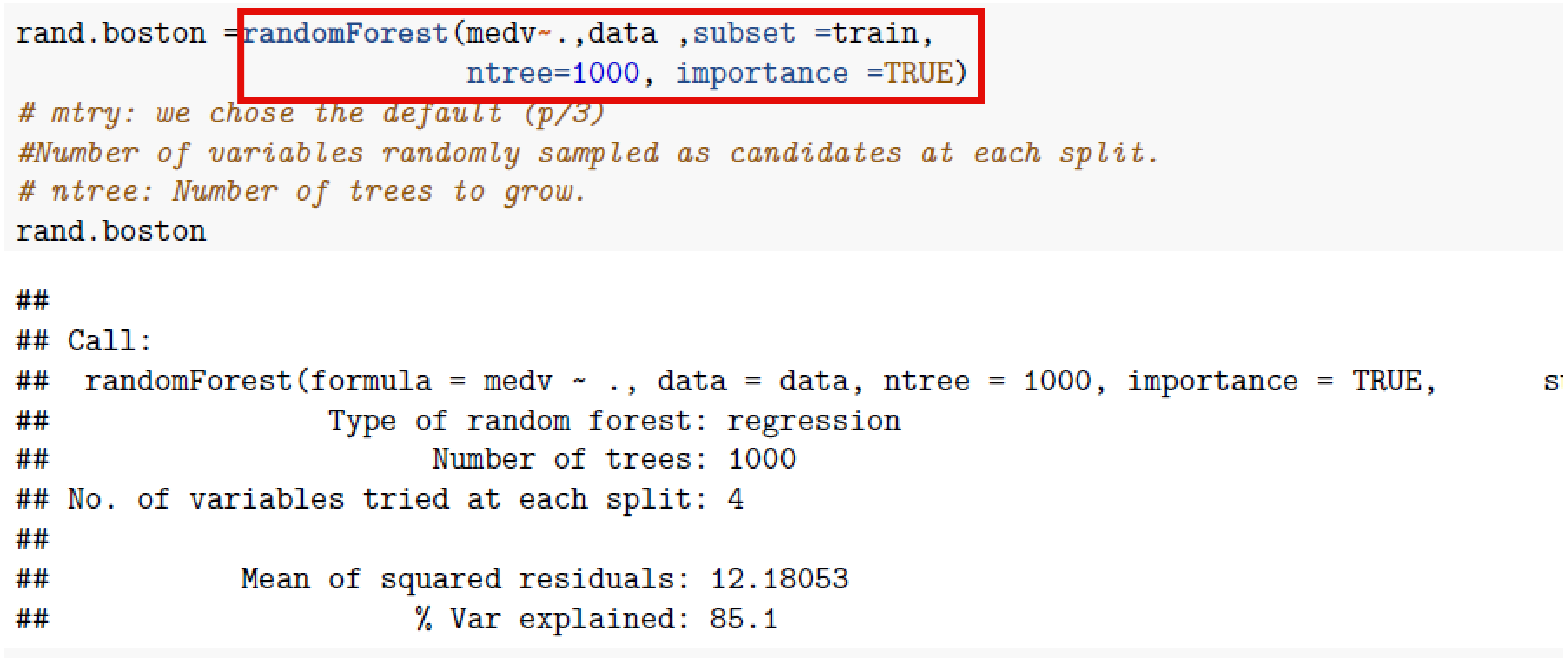
Random Forest

```r
rand.boston =randomForest(medv~.,data ,subset =train,
                          ntree=1000, importance =TRUE)
# mtry: we chose the default (p/3)
#Number of variables randomly sampled as candidates at each split.
# ntree: Number of trees to grow.
rand.boston
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = data, ntree = 1000, importance = TRUE,      s
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 12.18053
##                     % Var explained: 85.1
```

randomForest package - Regression

Random Forest



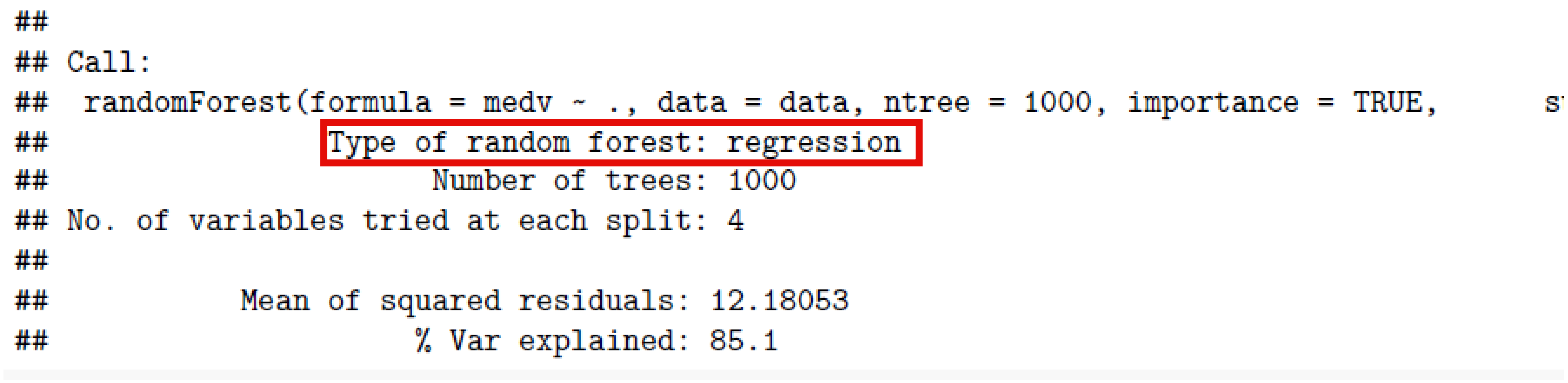
```
rand.boston =randomForest(medv~.,data ,subset =train,
                          ntree=1000, importance =TRUE)
# mtry: we chose the default (p/3)
#Number of variables randomly sampled as candidates at each split.
# ntree: Number of trees to grow.
rand.boston
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = data, ntree = 1000, importance = TRUE,      s
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 12.18053
##                     % Var explained: 85.1
```

randomForest package - Regression

Random Forest

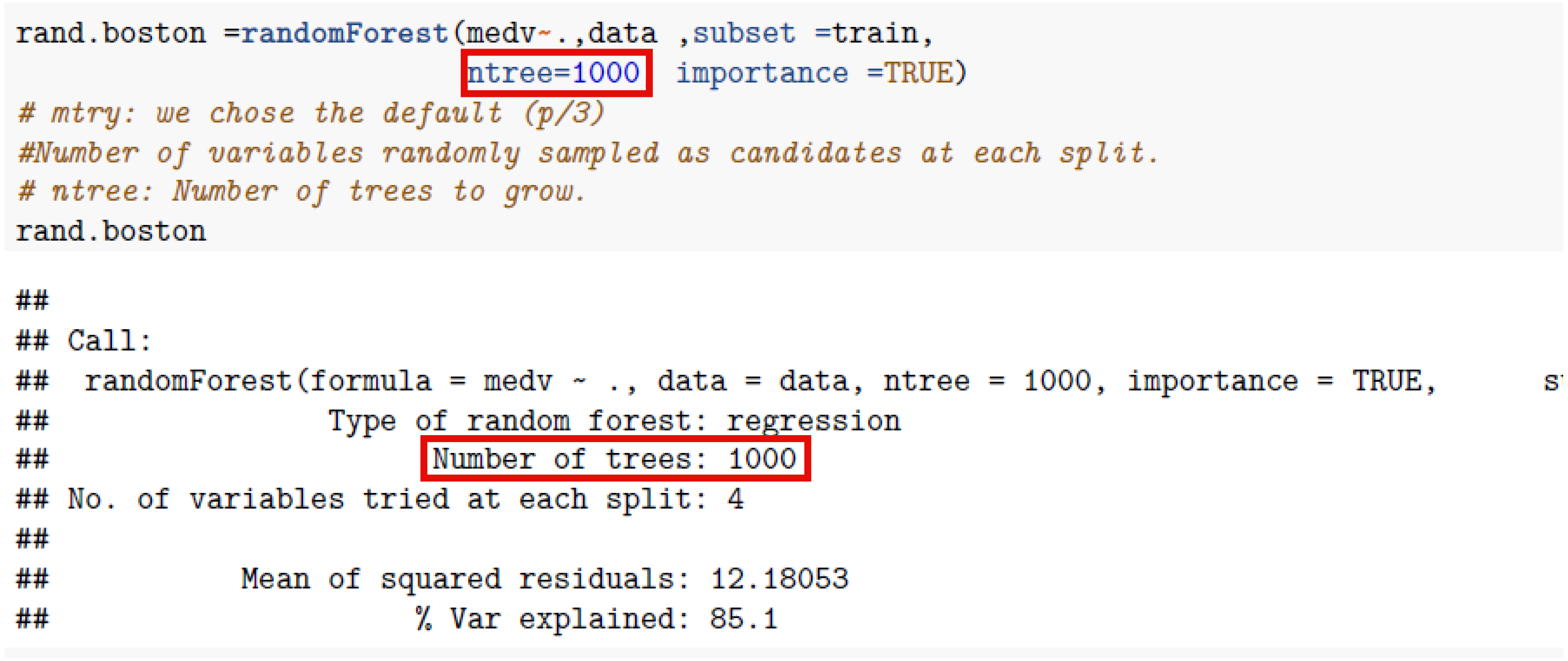
```
rand.boston =randomForest(medv~.,data ,subset =train,
                          ntree=1000, importance =TRUE)
# mtry: we chose the default (p/3)
#Number of variables randomly sampled as candidates at each split.
# ntree: Number of trees to grow.
rand.boston
```



```
##
## Call:
##  randomForest(formula = medv ~ ., data = data, ntree = 1000, importance = TRUE,      s
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 12.18053
##                     % Var explained: 85.1
```

randomForest package - Regression

Random Forest



```r
rand.boston =randomForest(medv~.,data ,subset =train,
                          ntree=1000  importance =TRUE)
# mtry: we chose the default (p/3)
#Number of variables randomly sampled as candidates at each split.
# ntree: Number of trees to grow.
rand.boston
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = data, ntree = 1000, importance = TRUE,      s
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 12.18053
##                     % Var explained: 85.1
```

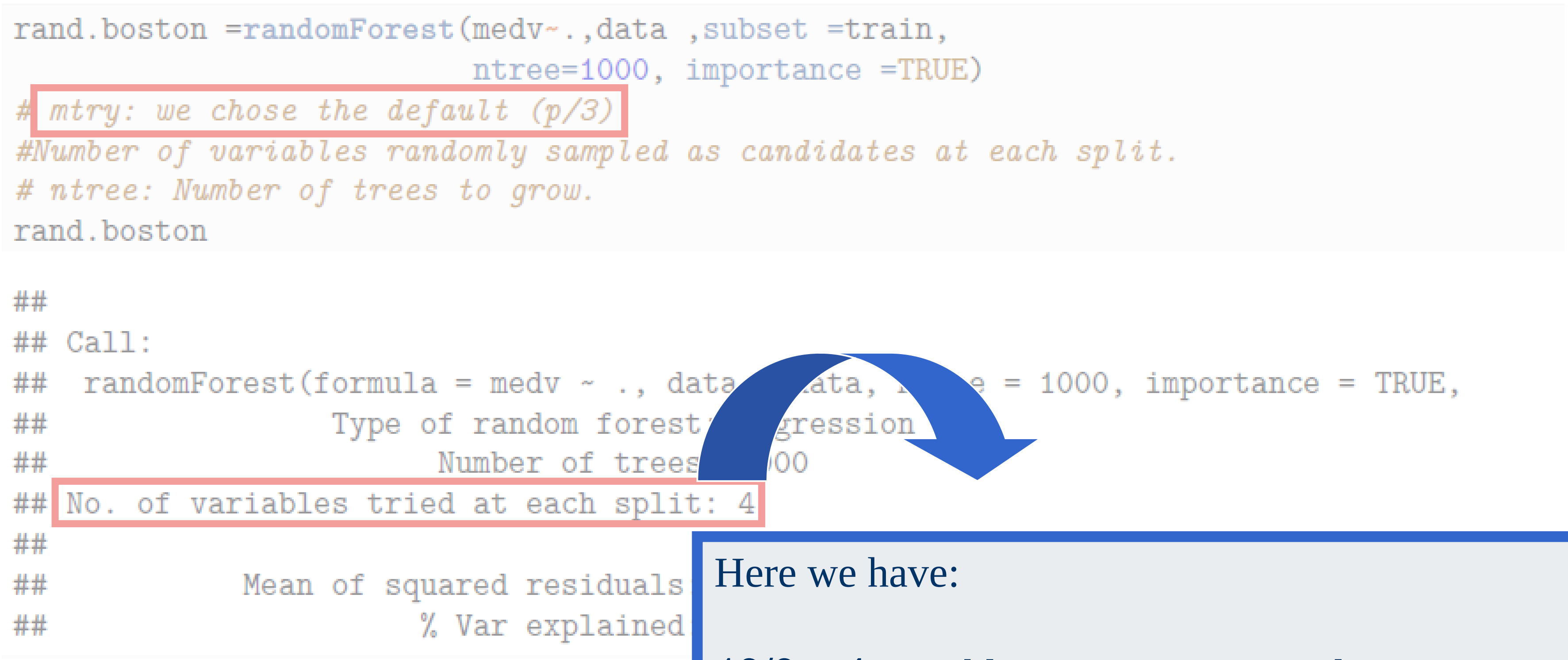
randomForest package - Regression

Random Forest

```
rand.boston =randomForest(medv~.,data ,subset =train,
                          ntree=1000, importance =TRUE)
# mtry: we chose the default (p/3)
#Number of variables randomly sampled as candidates at each split.
# ntree: Number of trees to grow.
rand.boston
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = data, ntree = 1000, importance = TRUE,      s
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 12.18053
##                     % Var explained: 85.1
```

randomForest package - Regression



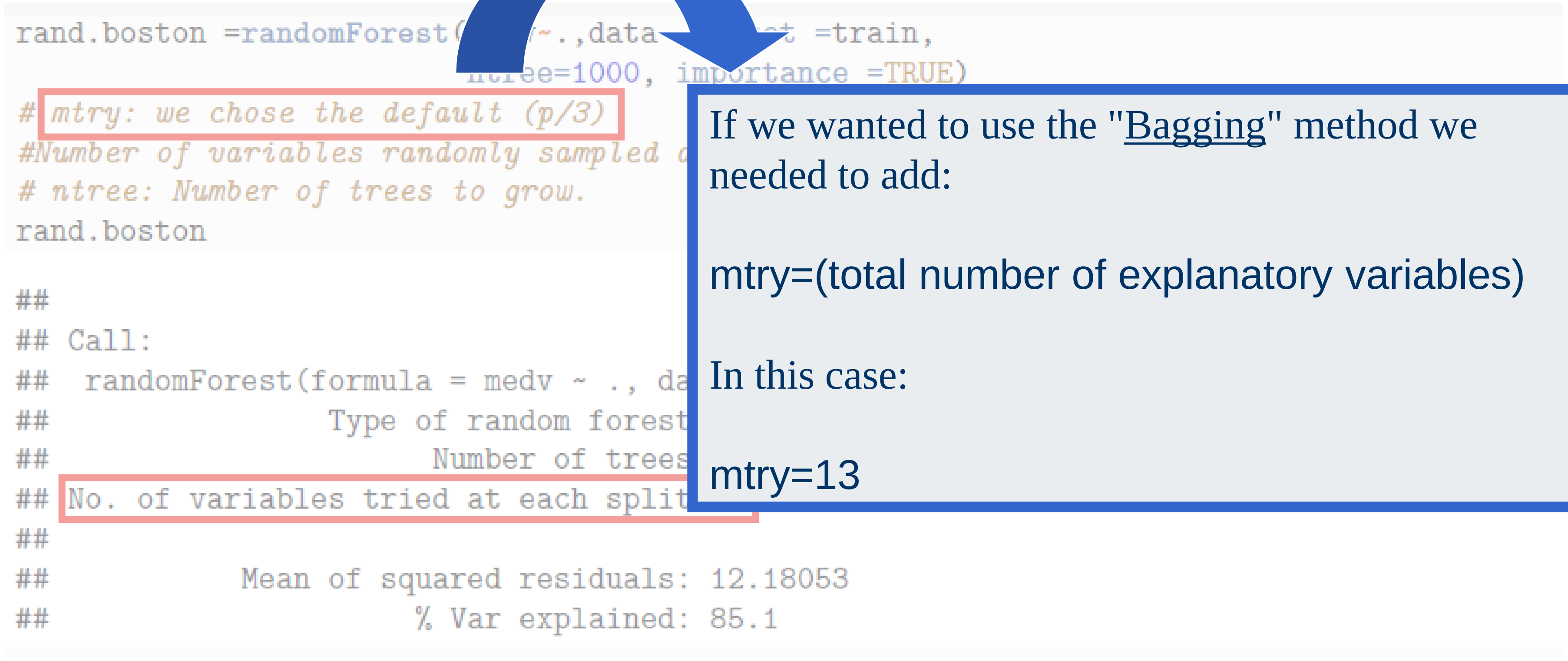
```
rand.boston =randomForest(medv~.,data ,subset =train,
                          ntree=1000, importance =TRUE)
# mtry: we chose the default (p/3)
#Number of variables randomly sampled as candidates at each split.
# ntree: Number of trees to grow.
rand.boston
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data    ata,      e = 1000, importance = TRUE,
##                Type of random forest:    gression
##                      Number of trees     00
## No. of variables tried at each split: 4
##
##           Mean of squared residuals
##                     % Var explained
```

Here we have:

13/3 ≈ 4 variables, since it is a Random Forest

randomForest package - Regression



```
rand.boston =randomForest(  ~.,data        et =train,
                              ntree=1000, importance =TRUE)
# mtry: we chose the default (p/3)
#Number of variables randomly sampled c
# ntree: Number of trees to grow.
rand.boston

##
## Call:
##  randomForest(formula = medv ~ ., da
##                Type of random forest
##                      Number of trees
## No. of variables tried at each split
##
##           Mean of squared residuals: 12.18053
##                     % Var explained: 85.1
```

If we wanted to use the "Bagging" method we needed to add:

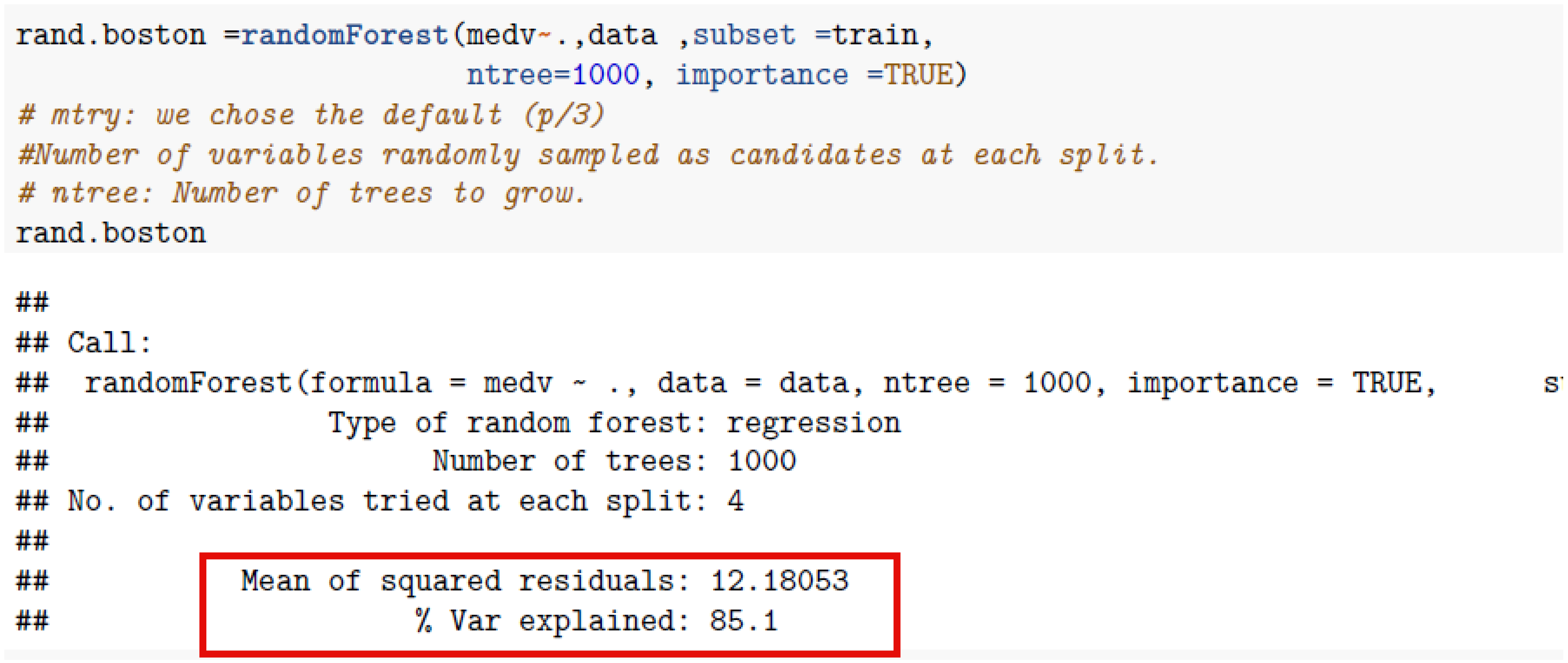
mtry=(total number of explanatory variables)

In this case:

mtry=13

randomForest package - Regression

Random Forest

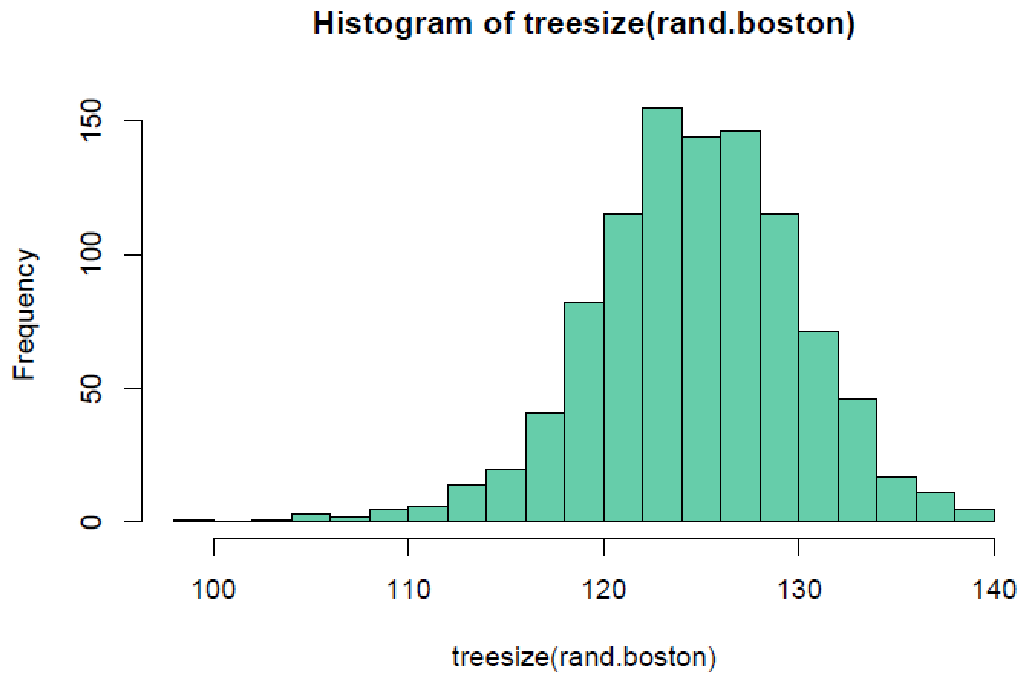


```
rand.boston =randomForest(medv~.,data ,subset =train,
                          ntree=1000, importance =TRUE)
# mtry: we chose the default (p/3)
#Number of variables randomly sampled as candidates at each split.
# ntree: Number of trees to grow.
rand.boston
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = data, ntree = 1000, importance = TRUE,      s
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 12.18053
##                     % Var explained: 85.1
```

randomForest package - Regression

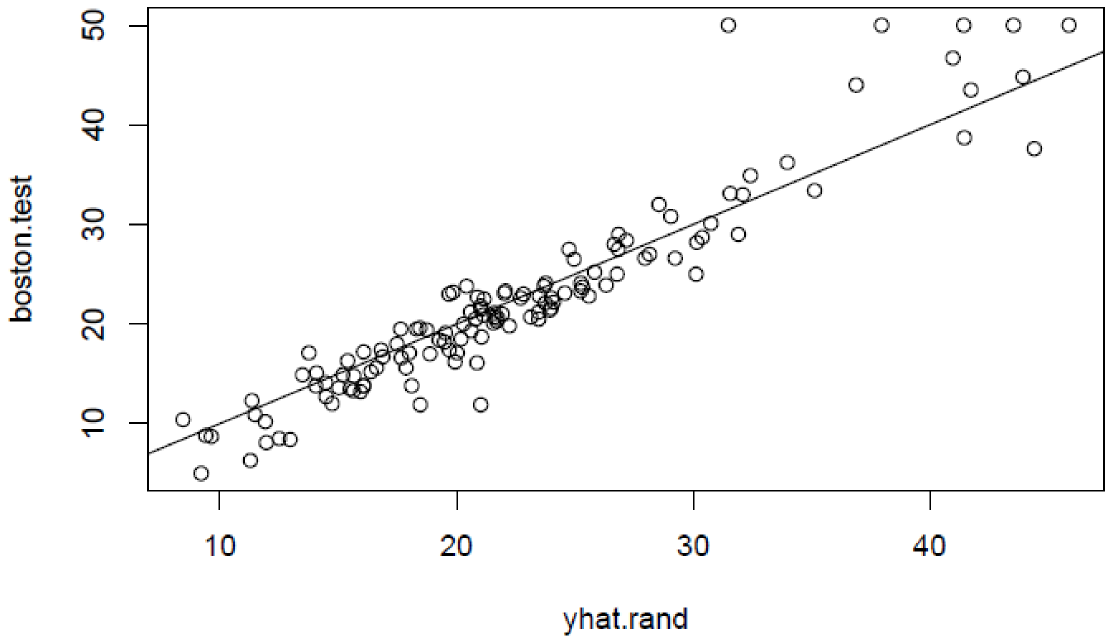
```
# Histogram of trees size
hist(treesize(rand.boston), breaks=20, col="aquamarine3")
```



tree size = number of nodes

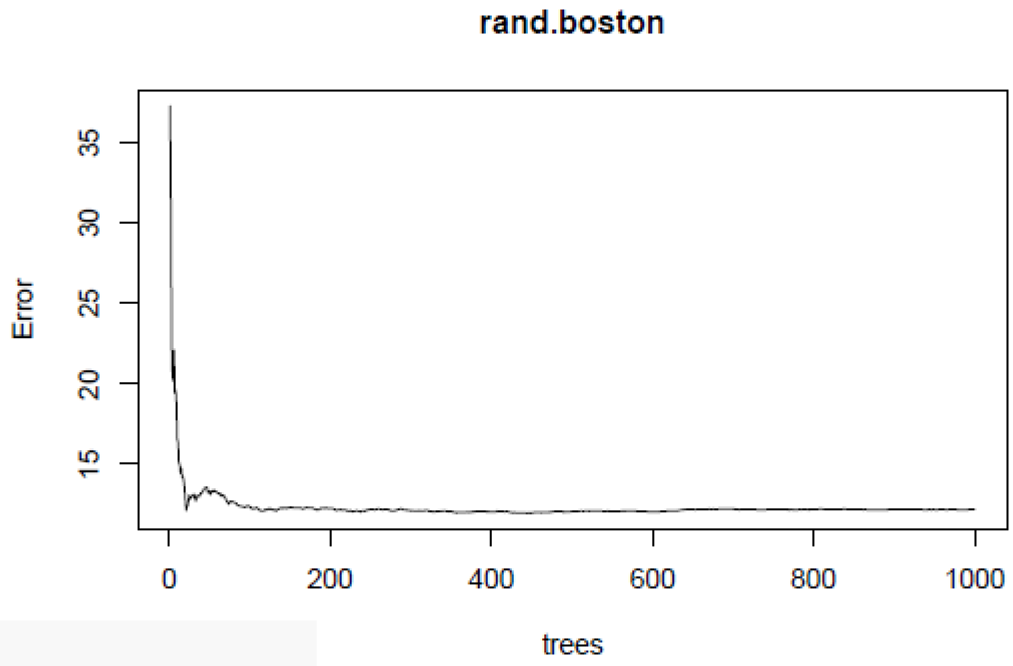
randomForest package - Regression

```
# Predictions
yhat.rand = predict (rand.boston , newdata = data[-train ,])
plot(yhat.rand , boston.test)
abline (0,1)
```



randomForest package - Regression

```
# Plot the error rates or MSE of a randomForest object x number of trees
plot(rand.boston, type="l")
```

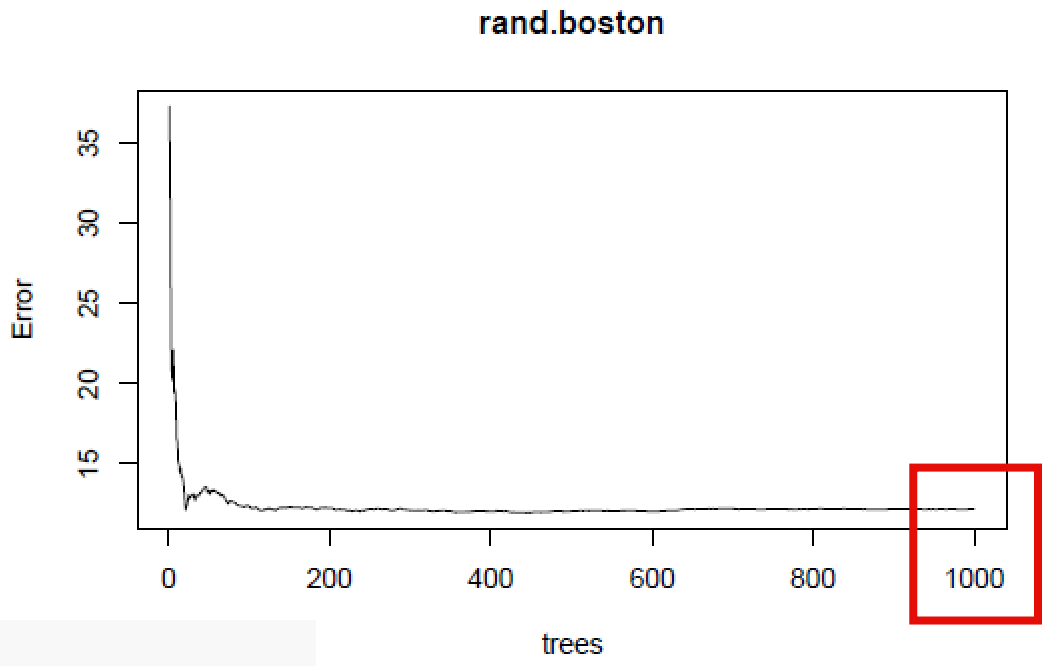


```
# MSE
mean(( yhat.rand -boston.test)^2)
```

```
## [1] 10.84324
```

randomForest package - Regression

```
# Plot the error rates or MSE of a randomForest object x number of trees
plot(rand.boston, type="l")
```



```
# MSE
mean(( yhat.rand -boston.test)^2)
```

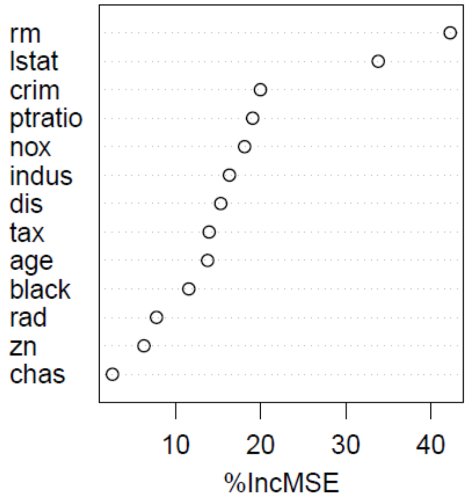
```
## [1] 10.84324
```

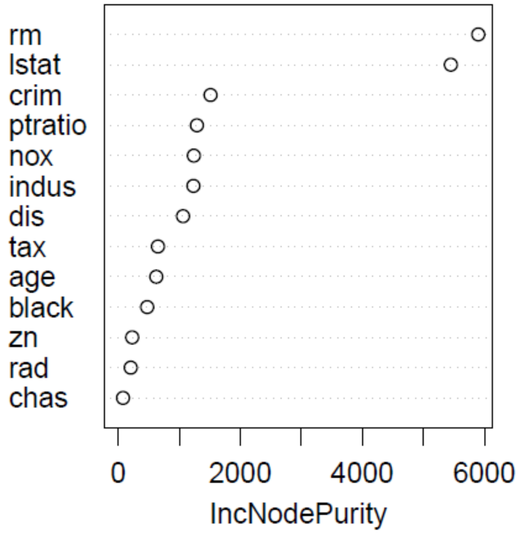
randomForest package - Regression

```r
# Importance of variables
importance (rand.boston)
```

```
##          %IncMSE IncNodePurity
## crim    21.378868     1849.6242
## zn       5.327068      327.3743
## indus   15.023468     2064.4995
## chas     4.422907      161.2110
## nox     22.588932     1997.7708
## rm      48.005356     8812.4332
## age     18.105861      866.5470
## dis     21.759105     1784.2645
## rad      7.799535      253.0358
## tax     17.556229     1116.2127
## ptratio 22.875649     2033.8358
## black   11.845736      577.0695
## lstat   38.052427     8568.2762
```

```r
varImpPlot (rand.boston)
```





randomForest package - Classification Iris data set

```
library(randomForest)
library(caret)
library(dplyr)
set.seed (2019)

# data
data <- iris
glimpse(iris)

# Train set
train = sample (1: nrow(data), nrow(data)*0.75)

# Test set
iris.test <- data[-train, "Species"]
```

```
## Observations: 150
## Variables: 5
```

randomForest package - Classification

Random Forest

```
rand.iris = randomForest(Species~.,data, subset=train,
                        ntree=1000, proximity=TRUE)
# we chose the default number of variables (sqrt(p))
rand.iris
```

```
##
## Call:
##  randomForest(formula = Species ~ ., data = data, ntree = 1000,      proximity = TRUE,
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 6.25%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         40          0         0  0.00000000
## versicolor      0         31         4  0.11428571
## virginica       0          3        34  0.08108108
```

randomForest package - Classification

Random Forest

```
rand.iris = randomForest(Species~.,data, subset=train,
                        ntree=1000, proximity=TRUE)
# we chose the default number of variables (sqrt(p))
rand.iris
```

```
##
## Call:
##  randomForest(formula = Species ~ ., data = data, ntree = 1000,      proximity = TRUE,
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 6.25%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         40          0         0  0.00000000
## versicolor      0         31         4  0.11428571
## virginica       0          3        34  0.08108108
```

randomForest package - Classification

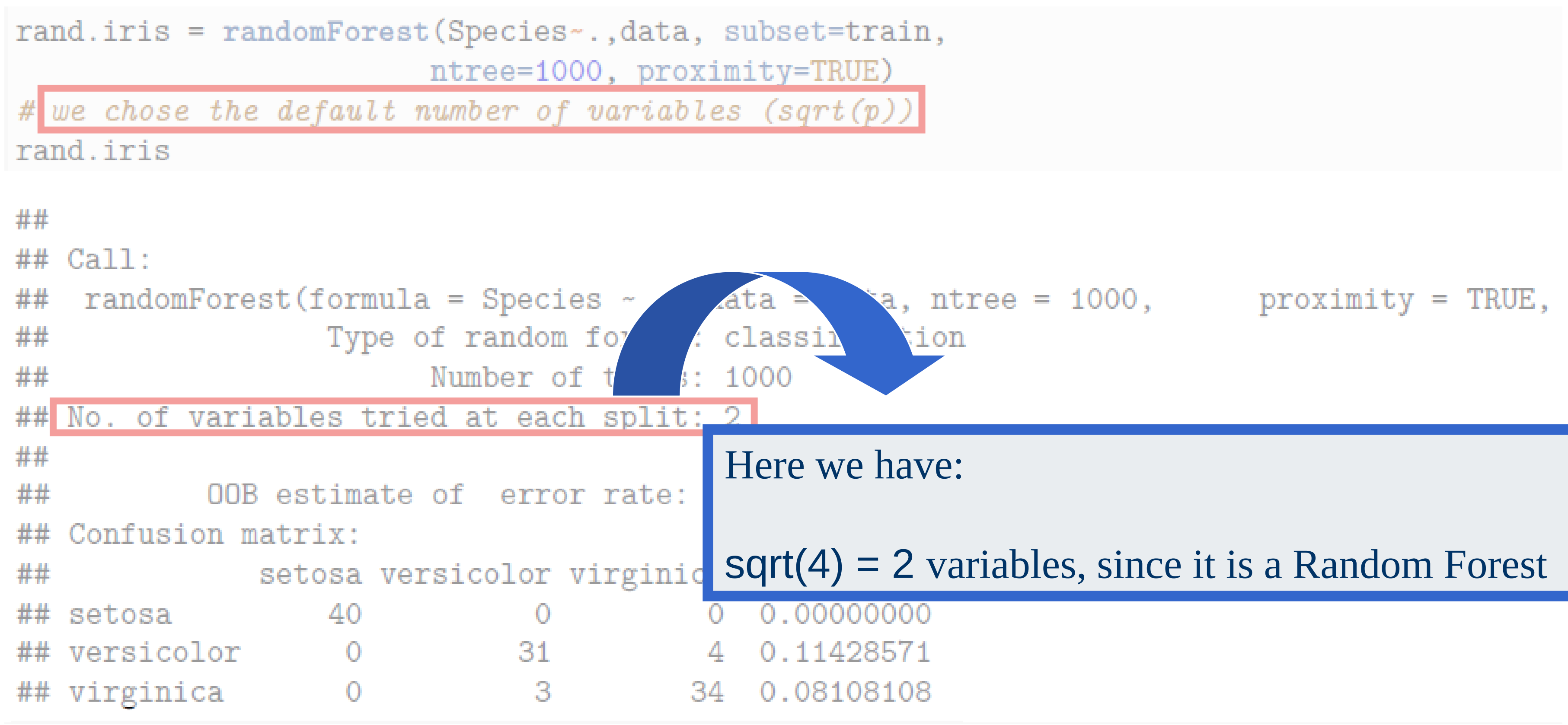
Random Forest

```
rand.iris = randomForest(Species~.,data, subset=train,
                        ntree=1000, proximity=TRUE)
# we chose the default number of variables (sqrt(p))
rand.iris
```

```
##
## Call:
##  randomForest(formula = Species ~ ., data = data, ntree = 1000,      proximity = TRUE,
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 6.25%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         40          0         0  0.00000000
## versicolor      0         31         4  0.11428571
## virginica       0          3        34  0.08108108
```

randomForest package - Classification

Random Forest



```
rand.iris = randomForest(Species~.,data, subset=train,
                         ntree=1000, proximity=TRUE)
# we chose the default number of variables (sqrt(p))
rand.iris
```

```
##
## Call:
##  randomForest(formula = Species ~     ata =     a, ntree = 1000,      proximity = TRUE,
##                Type of random fo      : classi     ion
##                      Number of t    s: 1000
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate:
## Confusion matrix:
##            setosa versicolor virginic
## setosa         40          0          0  0.00000000
## versicolor      0         31          4  0.11428571
## virginica       0          3         34  0.08108108
```

Here we have:

sqrt(4) = 2 variables, since it is a Random Forest

randomForest package - Classification

Random Forest

```r
rand.iris = randomForest(Species~.,data, subset=train,
                        ntree=1000, proximity=TRUE)
# we chose the default number of variables (sqrt(p))
rand.iris
```

```
##
## Call:
##  randomForest(formula = Species ~ ., data = data, ntree = 1000,      proximity = TRUE,
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 6.25%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         40          0         0  0.00000000
## versicolor      0         31         4  0.11428571
## virginica       0          3        34  0.08108108
```

randomForest package - Classification

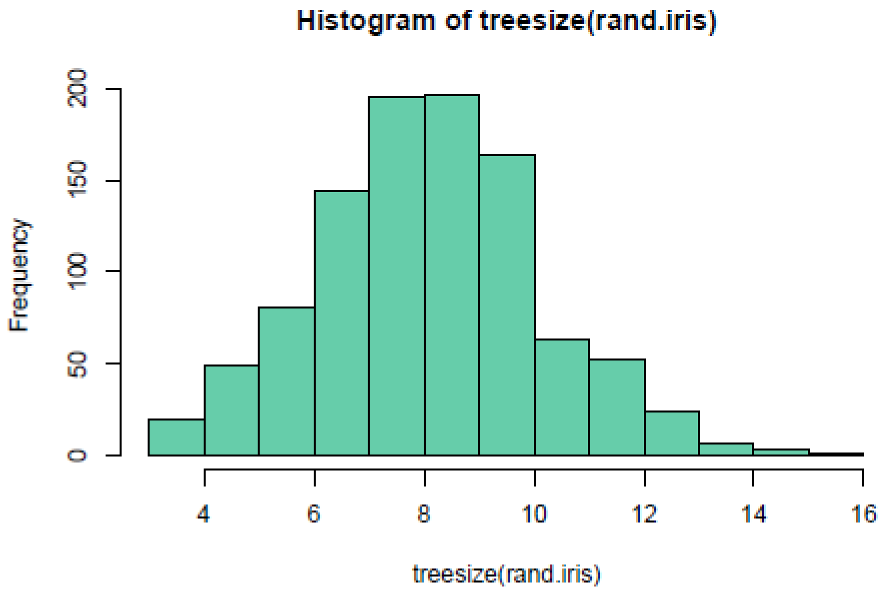
Random Forest

```
rand.iris = randomForest(Species~.,data, subset=train,
                        ntree=1000, proximity=TRUE)
# we chose the default number of variables (sqrt(p))
rand.iris
```

```
##
## Call:
##  randomForest(formula = Species ~ ., data = data, ntree = 1000,      proximity = TRUE,
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 6.25%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         40          0         0  0.00000000
## versicolor      0         31         4  0.11428571
## virginica       0          3        34  0.08108108
```

randomForest package - Classification

```
# Histogram of trees size
hist(treesize(rand.iris), breaks=10, col="aquamarine3")
```

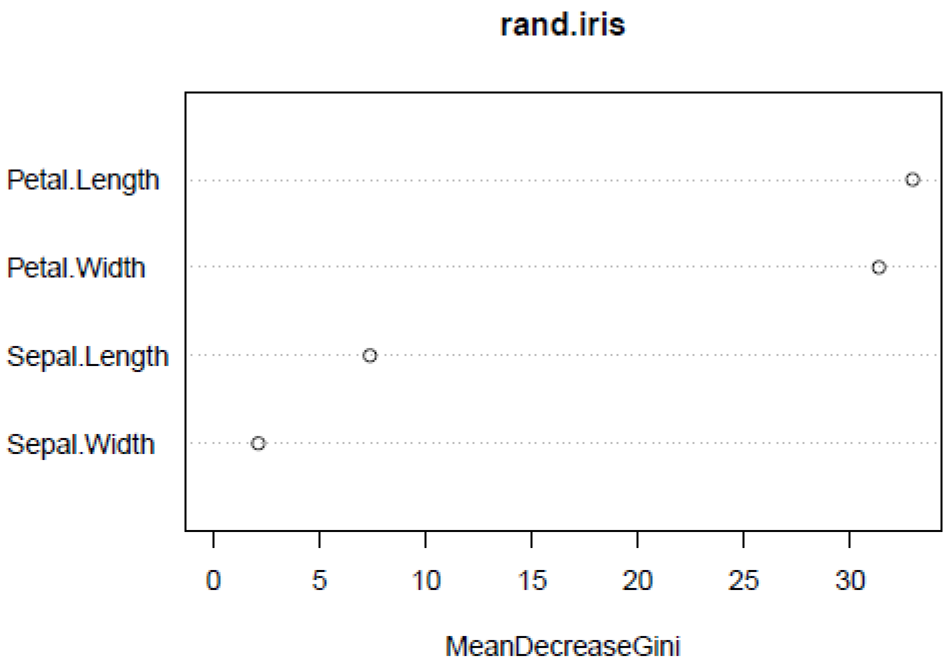


randomForest package - Classification

```
# Importance of variables
importance(rand.iris)
```

```
##              MeanDecreaseGini
## Sepal.Length         7.380465
## Sepal.Width          2.121128
## Petal.Length        32.945481
## Petal.Width         31.354294
```

```
varImpPlot(rand.iris)
```



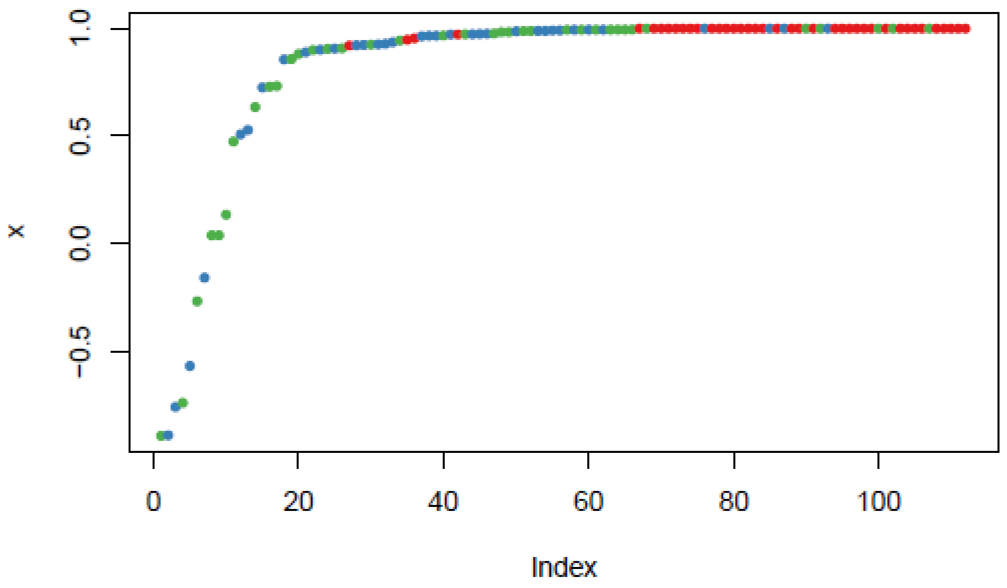
randomForest package - Classification

```r
# Prediction
irisPred <- predict(rand.iris , newdata = data[-train ,])
table(irisPred, iris.test)
```

```
##              iris.test
## irisPred     setosa versicolor virginica
##   setosa         10          0         0
##   versicolor      0         15         0
##   virginica       0          0        13
```

randomForest package - Classification

```
# Margin
# Compute or plot the margin of predictions from a randomForest classifier
margin.iris <- randomForest::margin(rand.iris,iris.test)
plot(margin.iris)
```



The margin of a data point is defined as the proportion of votes for the correct class minus maximum proportion of votes for the other classes. Thus under majority votes, positive margin means correct classification, and vice versa.

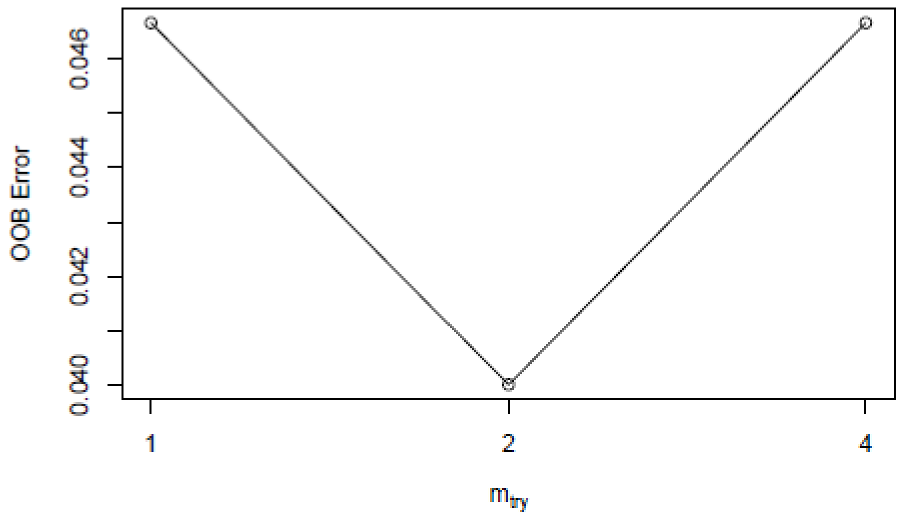
randomForest package - Classification

```r
# Tuning
# Tune randomForest for the optimal mtry parameter
tune.rf <- tuneRF(iris[,-5], iris[,5], stepFactor=0.5)
```

```
## mtry = 2  OOB error = 4%
## Searching left ...
## mtry = 4      OOB error = 4.67%
## -0.1666667 0.05
## Searching right ...
## mtry = 1      OOB error = 4.67%
## -0.1666667 0.05
```

```r
tune.rf
```

```
##       mtry   OOBError
## 1.OOB    1 0.04666667
## 2.OOB    2 0.04000000
## 4.OOB    4 0.04666667
```



Feature Selection

Generalized Boosted Regression Models
gbm package

Generalized Boosted Regression Models
gbm package

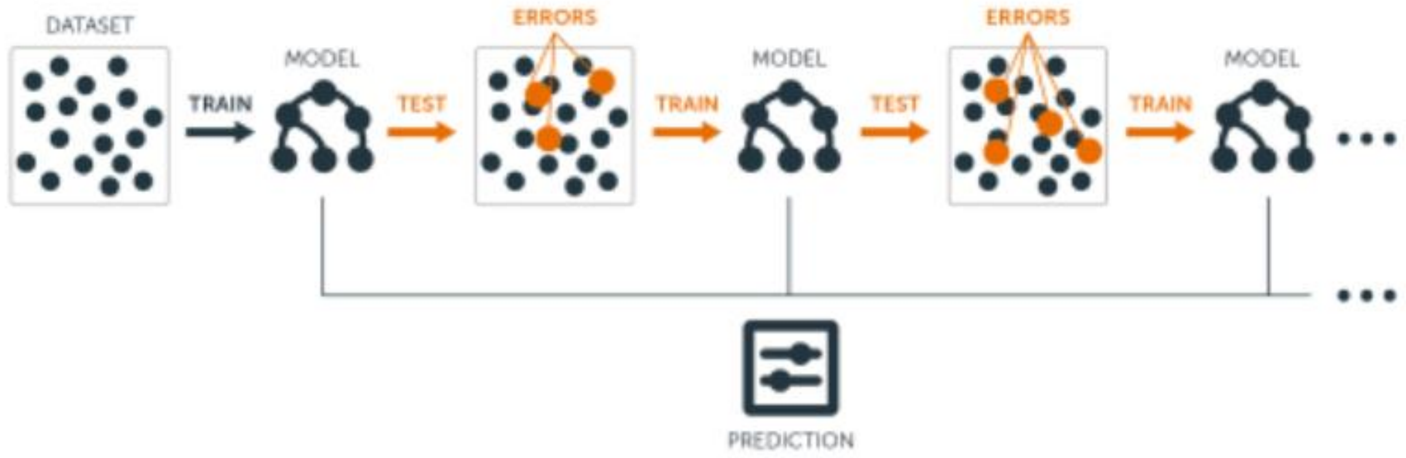
- Gradient Boosting trains many models in a gradual, additive and sequential manner.

- The major difference between AdaBoost and Gradient Boosting Algorithm is how the two algorithms identify the shortcomings of weak learners (eg. decision trees).

- While the AdaBoost model identifies the shortcomings by using high weight data points, gradient boosting performs the same by using gradients in the loss function

Boosting

- The main idea of boosting is to add new models to the ensemble ***sequentially***. At each particular iteration, a new weak, base-learner model is trained with respect to the error of the whole ensemble learnt so far.



Boosting

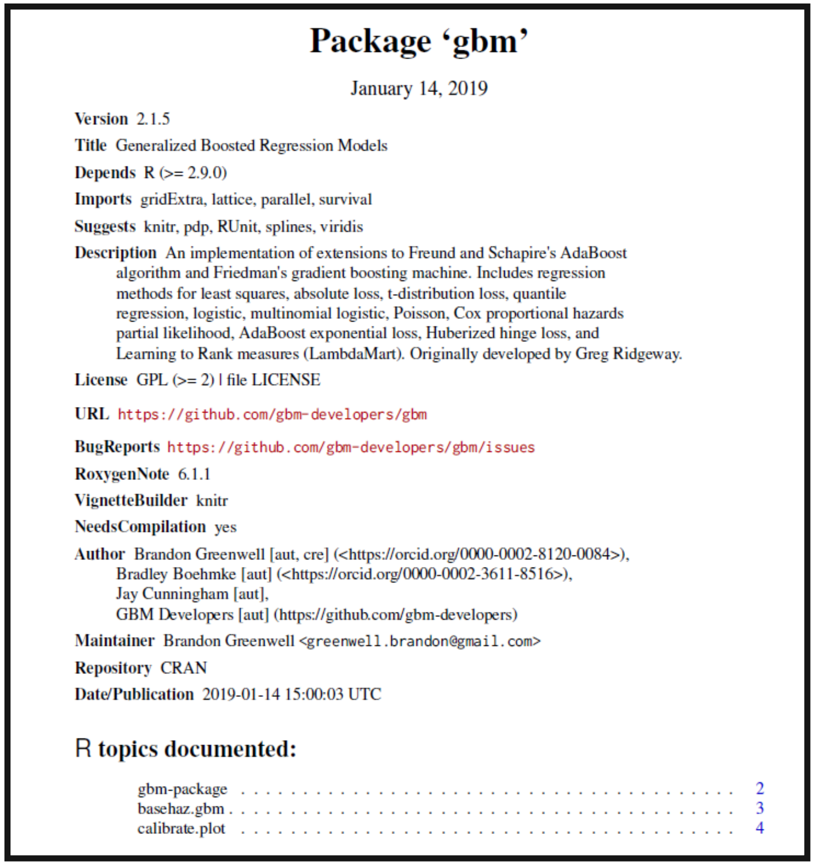
Boosting has three tuning parameters

•**Number of trees:** The total number of trees to fit. GBMs often require many trees; however, unlike random forests GBMs can overfit so the goal is to find the optimal number of trees that minimize the loss function of interest with cross validation.

•**Depth of trees:** The number *d* of splits in each tree, which controls the complexity of the boosted ensemble.

•**Learning rate:** The shrinkage parameter. It controls the rate in which boosting learns.

gbm R package



Package 'gbm'

January 14, 2019

Version 2.1.5

Title Generalized Boosted Regression Models

Depends R (>= 2.9.0)

Imports gridExtra, lattice, parallel, survival

Suggests knitr, pdp, RUnit, splines, viridis

Description An implementation of extensions to Freund and Schapire's AdaBoost algorithm and Friedman's gradient boosting machine. Includes regression methods for least squares, absolute loss, t-distribution loss, quantile regression, logistic, multinomial logistic, Poisson, Cox proportional hazards partial likelihood, AdaBoost exponential loss, Huberized hinge loss, and Learning to Rank measures (LambdaMart). Originally developed by Greg Ridgeway.

License GPL (>= 2) | file LICENSE

URL https://github.com/gbm-developers/gbm

BugReports https://github.com/gbm-developers/gbm/issues

RoxygenNote 6.1.1

VignetteBuilder knitr

NeedsCompilation yes

Author Brandon Greenwell [aut, cre] (<https://orcid.org/0000-0002-8120-0084>), Bradley Boehmke [aut] (<https://orcid.org/0000-0002-3611-8516>), Jay Cunningham [aut], GBM Developers [aut] (https://github.com/gbm-developers)

Maintainer Brandon Greenwell <greenwell.brandon@gmail.com>

Repository CRAN

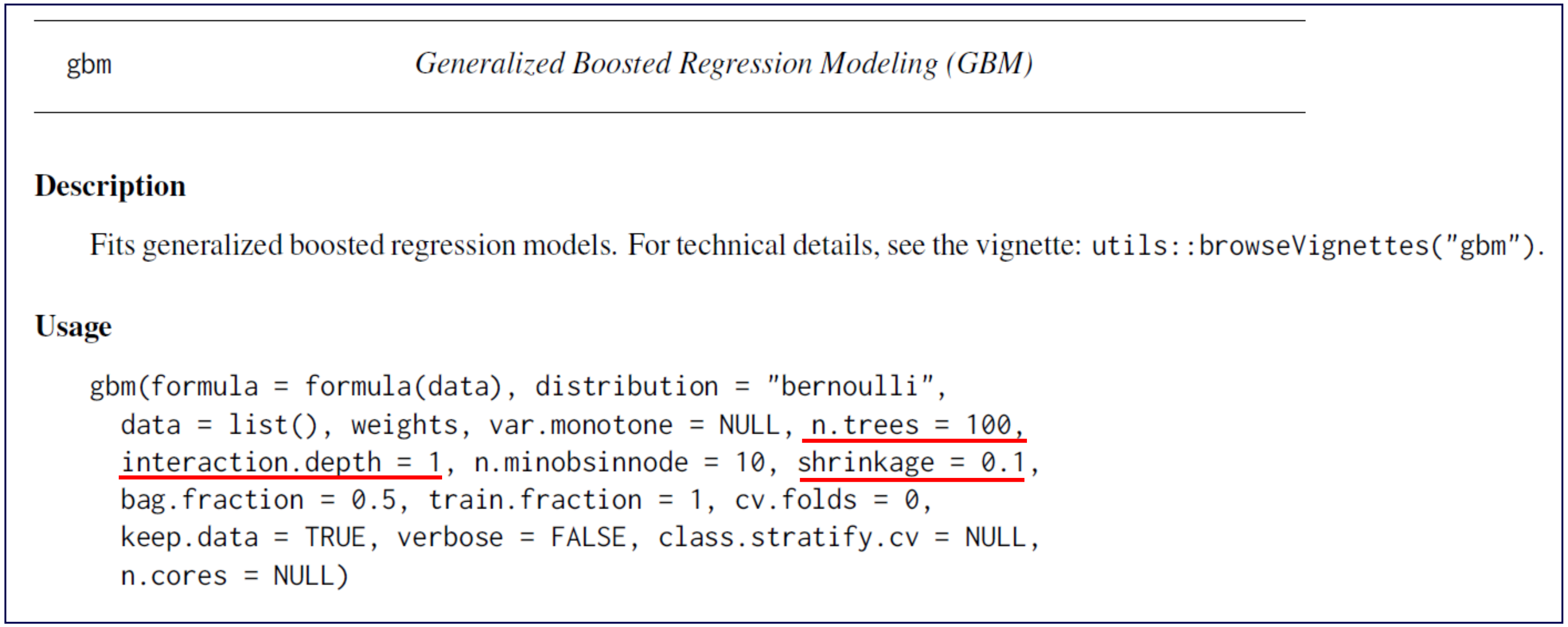
Date/Publication 2019-01-14 15:00:03 UTC

R topics documented:

The gbm R package is an implementation of extensions to Freund and Schapire's AdaBoost algorithm and Friedman's gradient boosting machine. This is the original R implementation of GBM.

URL https://github.com/gbm-developers/gbm

gbm R package



gbm *Generalized Boosted Regression Modeling (GBM)*

Description

Fits generalized boosted regression models. For technical details, see the vignette: `utils::browseVignettes("gbm")`.

Usage

```
gbm(formula = formula(data), distribution = "bernoulli",
  data = list(), weights, var.monotone = NULL, n.trees = 100,
  interaction.depth = 1, n.minobsinnode = 10, shrinkage = 0.1,
  bag.fraction = 0.5, train.fraction = 1, cv.folds = 0,
  keep.data = TRUE, verbose = FALSE, class.stratify.cv = NULL,
  n.cores = NULL)
```

Arguments:

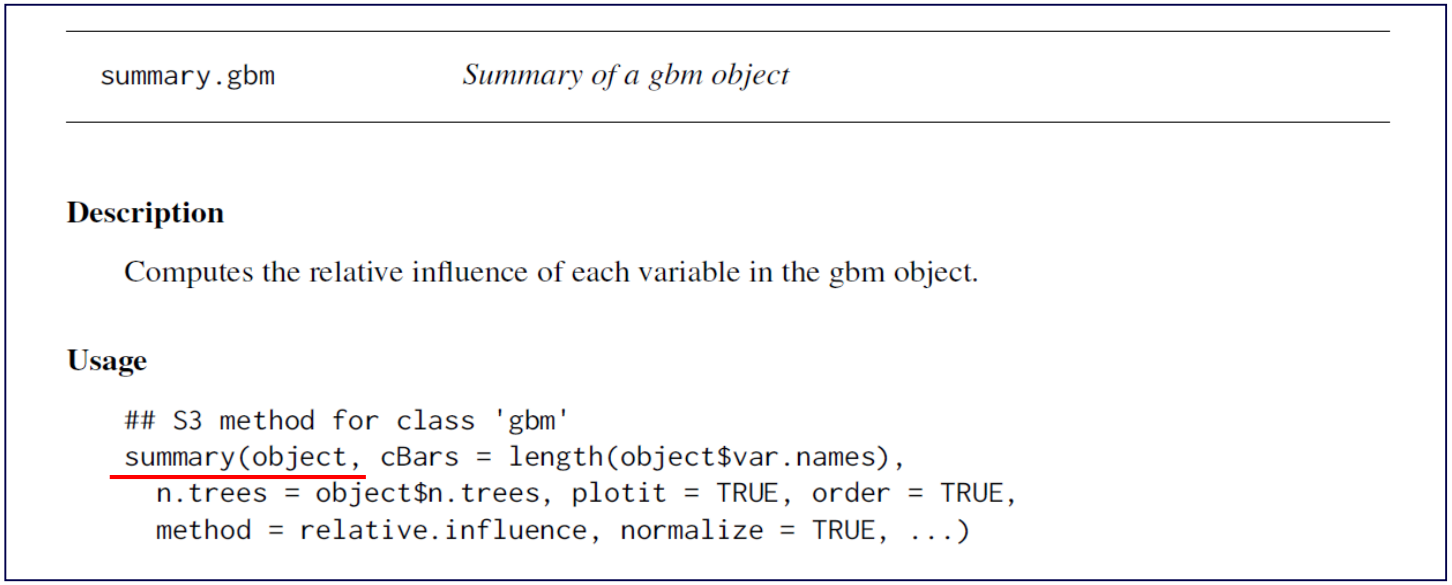
- •formula - A symbolic description of the model to be fit
- •distribution - name of the distribution
- •n.trees - Integer specifying the total number of trees to fit
- •interaction.depth - the maximum depth of each tree
- •shrinkage - the learning rate or step-size reduction

gbm R package

```
train <- sample(1: nrow(Boston), nrow(Boston) * 0.75)
# Test and Train set
Boston.train <- Boston[train,]
Boston.test <- Boston[-train,]
```

```
# Model for the Regression problem
# Default settings in gbm
boston.boost1<- gbm(medv~., data = Boston.train, cv.folds = 5)

## Distribution not specified, assuming gaussian ...

print(boston.boost1)

## gbm(formula = medv ~ ., data = Boston.train, cv.folds = 5)
## A gradient boosted model with gaussian loss function.
## 100 iterations were performed.
## The best cross-validation iteration was 99.
## There were 13 predictors of which 10 had non-zero influence.
```

gbm R package



summary.gbm *Summary of a gbm object*

Description

Computes the relative influence of each variable in the gbm object.

Usage

```
## S3 method for class 'gbm'
summary(object, cBars = length(object$var.names),
  n.trees = object$n.trees, plotit = TRUE, order = TRUE,
  method = relative.influence, normalize = TRUE, ...)
```

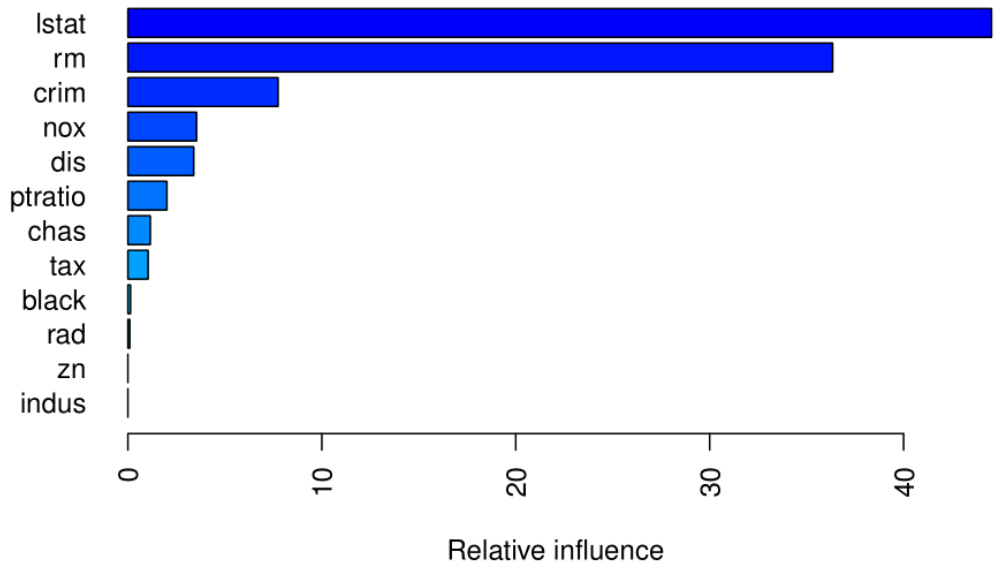
Arguments:

•object - a gbm object created from an initial call to gbm.
•cBars - number of bars to plot
•method - function used to compute the relative influence.
The permutation.test.gbm: Randomly permutes each predictor variable at a time and computes the associated reduction in predictive performance.

gbm R package

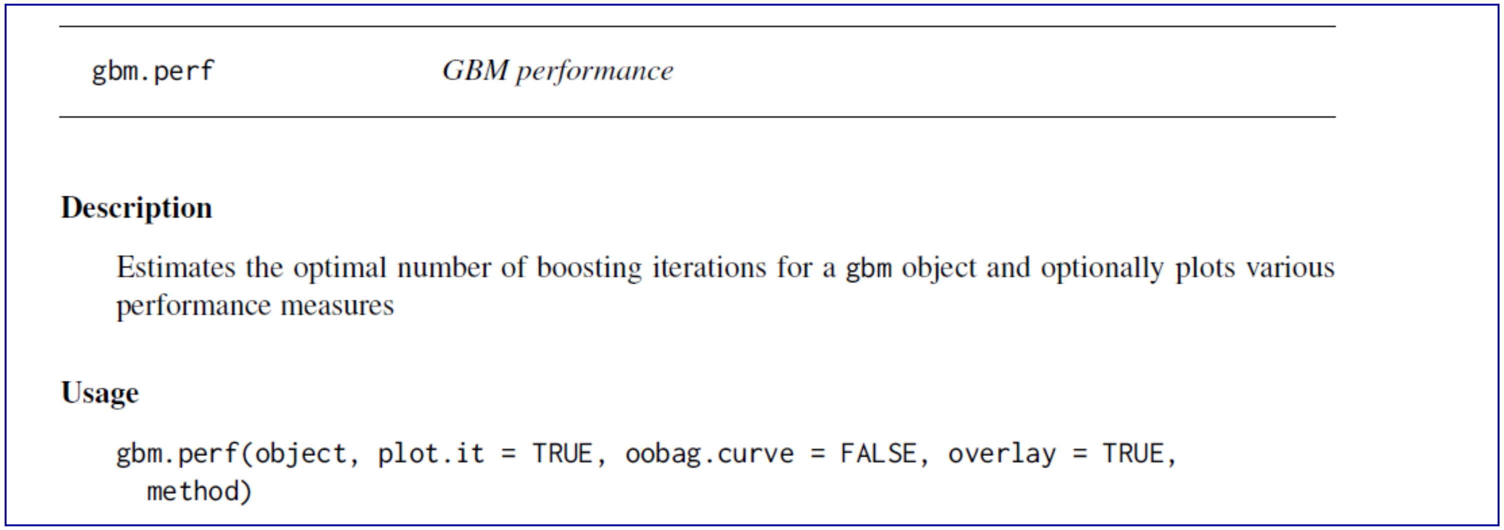
Eduardo Gonnelli

```r
# Variable importance

summary(boston.boost1,cBars = 12,
        method = relative.influence, # also can use permutation.test.gbm
        las = 2)
```



```
##                 var     rel.inf
## lstat         lstat 44.53540713
## rm               rm 36.34203002
## crim           crim  7.74309272
## nox             nox  3.54144074
## dis             dis  3.39437715
## ptratio     ptratio  2.01057975
## chas           chas  1.15397212
## tax             tax  1.04375450
## black         black  0.13674443
## rad             rad  0.09860143
## zn               zn  0.00000000
## indus         indus  0.00000000
## age             age  0.00000000
```

gbm R package



gbm.perf *GBM performance*

Description

Estimates the optimal number of boosting iterations for a gbm object and optionally plots various performance measures

Usage

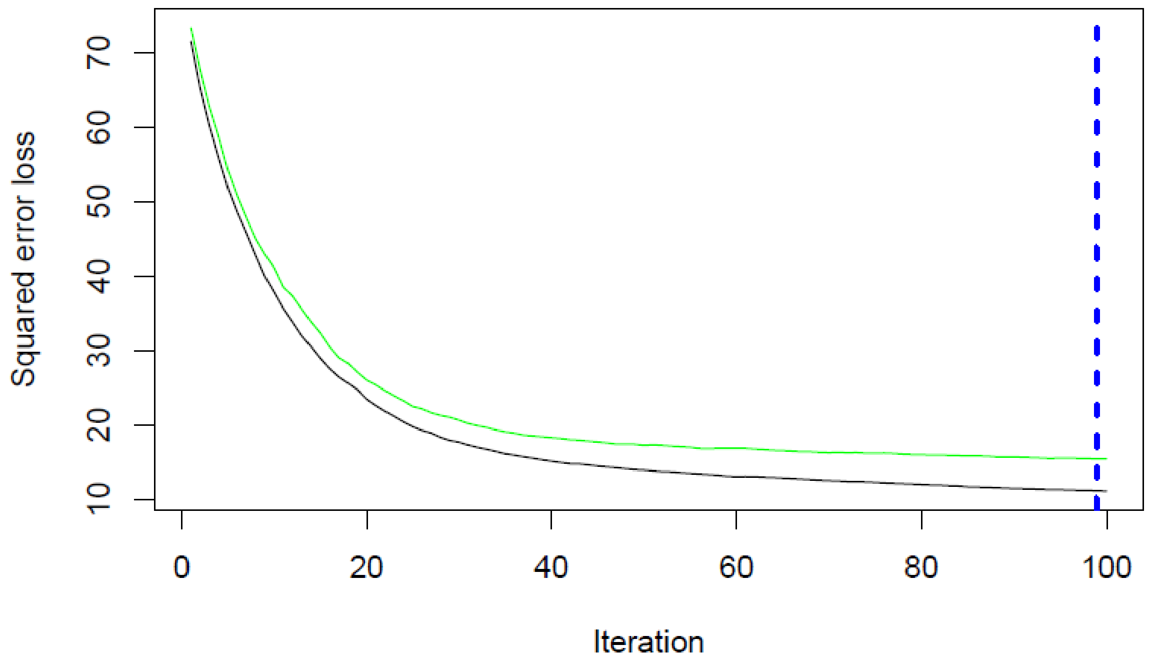
```
gbm.perf(object, plot.it = TRUE, oobag.curve = FALSE, overlay = TRUE,
  method)
```

Arguments:

• method - used to estimate the optimal number of boosting iterations.
method = "OOB" computes the out-of-bag estimate
method = "cv" extracts the optimal number of iterations using cross-validation if gbm was called with cv.folds > 1.

gbm R package

```r
# plot loss function as a result of n trees added to the ensemble
gbm.perf(boston.boost1, method = "cv")
```



```
## [1] 99
```

gbm R package

Tuning parameters

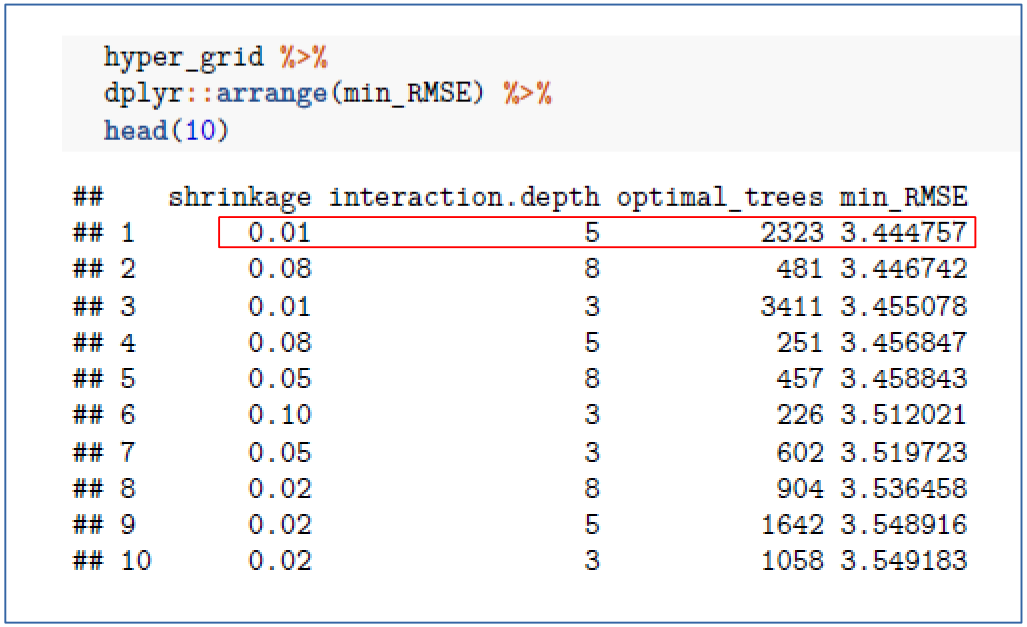
```
# Tuning
# Create hyperparameter grid
hyper_grid <- expand.grid(
  shrinkage = c(0.01, 0.02, 0.05, 0.08, 0.1),
  interaction.depth = c(3, 5, 8),
  optimal_trees = 0,               # a place to dump results
  min_RMSE = 0                     # a place to dump results
)

# total number of combinations
nrow(hyper_grid)

## [1] 15
```

```
for(i in 1:nrow(hyper_grid)) {

  # Train GBM model
  gbm.tune <- gbm(formula = medv ~ .,distribution = "gaussian",data = Boston.train,
    n.trees = 5000,interaction.depth = hyper_grid$interaction.depth[i],
    shrinkage = hyper_grid$shrinkage[i], cv.folds = 5
  )


  # add min training error and trees to grid
  hyper_grid$optimal_trees[i] <- which.min(gbm.tune$cv.error)
  hyper_grid$min_RMSE[i] <- sqrt(min(gbm.tune$cv.error))
}
```

gbm R package

Eduardo Gonnelli

Tuning parameters

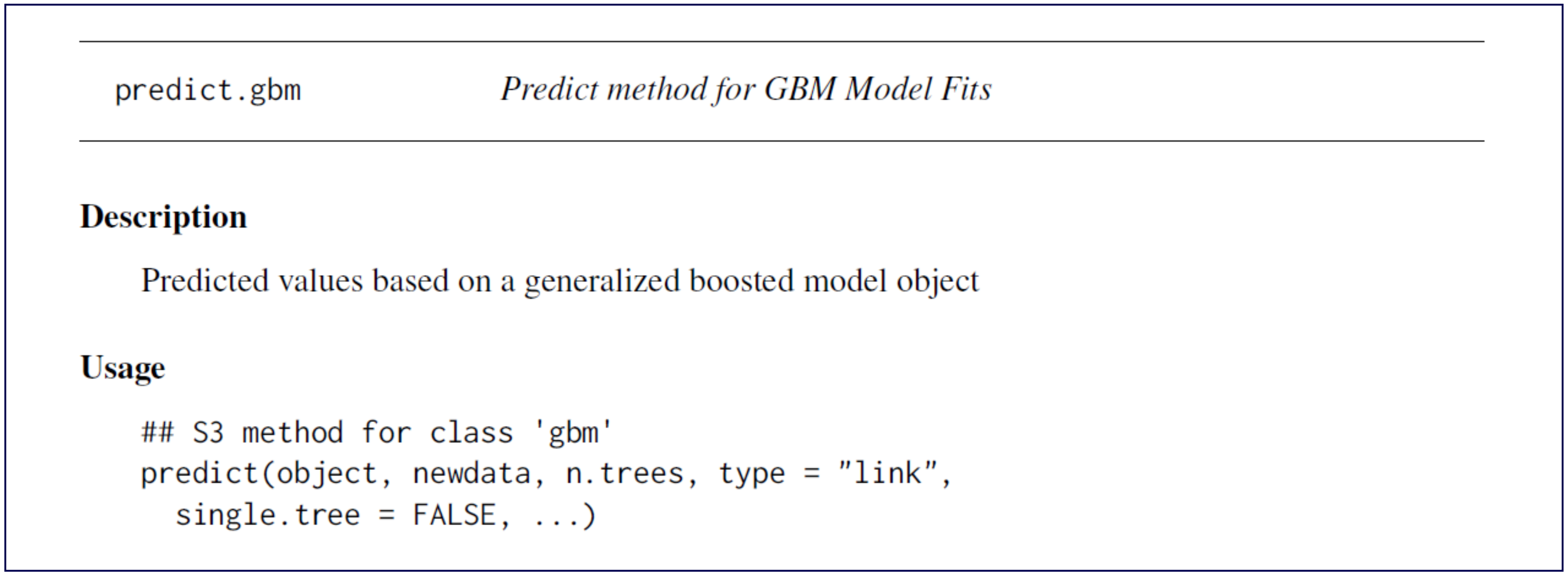


```r
hyper_grid %>%
dplyr::arrange(min_RMSE) %>%
head(10)
```

```
##    shrinkage interaction.depth optimal_trees min_RMSE
## 1       0.01                 5          2323 3.444757
## 2       0.08                 8           481 3.446742
## 3       0.01                 3          3411 3.455078
## 4       0.08                 5           251 3.456847
## 5       0.05                 8           457 3.458843
## 6       0.10                 3           226 3.512021
## 7       0.05                 3           602 3.519723
## 8       0.02                 8           904 3.536458
## 9       0.02                 5          1642 3.548916
## 10      0.02                 3          1058 3.549183
```

```r
boston.opt <-  gbm(formula = medv ~ .,distribution = "gaussian",data = Boston.train,
  n.trees = 2323, interaction.depth = 5,shrinkage = 0.01,cv.folds = 5)
```

gbm R package



predict.gbm *Predict method for GBM Model Fits*

Description

Predicted values based on a generalized boosted model object

Usage

```
## S3 method for class 'gbm'
predict(object, newdata, n.trees, type = "link",
  single.tree = FALSE, ...)
```

The predict.gbm produces predicted values for each observation in newdata using the first n.trees iterations of the boosting sequence.

gbm R package

```
#Predicting values for test data

boston.opt.test.pred.tree = predict(boston.opt, Boston.test, n.trees = boston.opt$n.trees)
MSE.test.boosting.opt <- mean((boston.opt.test.pred.tree - Boston.test$medv)^2)
MSE.test.boosting.opt
```

```
## [1] 9.335418
```

```
library(MLmetrics) # for MSE
```

```
MSE(boston.opt.test.pred.tree, Boston.test$medv)
```

```
## [1] 9.335418
```

gbm R package

```
#Predicting values for train data
boston.opt.train.pred.tree = predict(boston.opt, n.trees = boston.opt$n.trees)
MSE.train.boosting.opt <- mean((boston.opt.train.pred.tree - Boston.train$medv)^2)
MSE.train.boosting.opt
```

```
## [1] 1.641844
```

```
MSE(boston.opt.train.pred.tree,Boston.train$medv)
```

```
## [1] 1.641844
```

gbm R package

Methods	MSE – Test Error
Tree	18.17
rpart	16.97
Random Forest	10.84
Boosting	9.34

Thank you very much for your attention

Grazie mille per la vostra attenzione

Muito obrigado pela atenção