Different Random Forest Packages in R

Thiyanga Talagala

14 November 2017

```
library(knitr)
opts_chunk$set(tidy = TRUE)
```

randomForest Package

Data pre-processing

Split iris data to Training data and testing data

```
ind <- sample(2, nrow(iris), replace = TRUE, prob = c(0.7, 0.3))
trainData <- iris[ind == 1, ]
testData <- iris[ind == 2, ]
```

1. Load randomForest

```
library(randomForest)
```

```
## randomForest 4.6-12
```

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

2. Generate Random Forest learning treee

```
iris_rf <- randomForest(Species ~ ., data = trainData, ntree = 100, proximity = TRUE)
table(predict(iris_rf), trainData$Species)
```

```
##
##              setosa versicolor virginica
##   setosa         32          0         0
##   versicolor      0         37         5
##   virginica       0          2        33
```

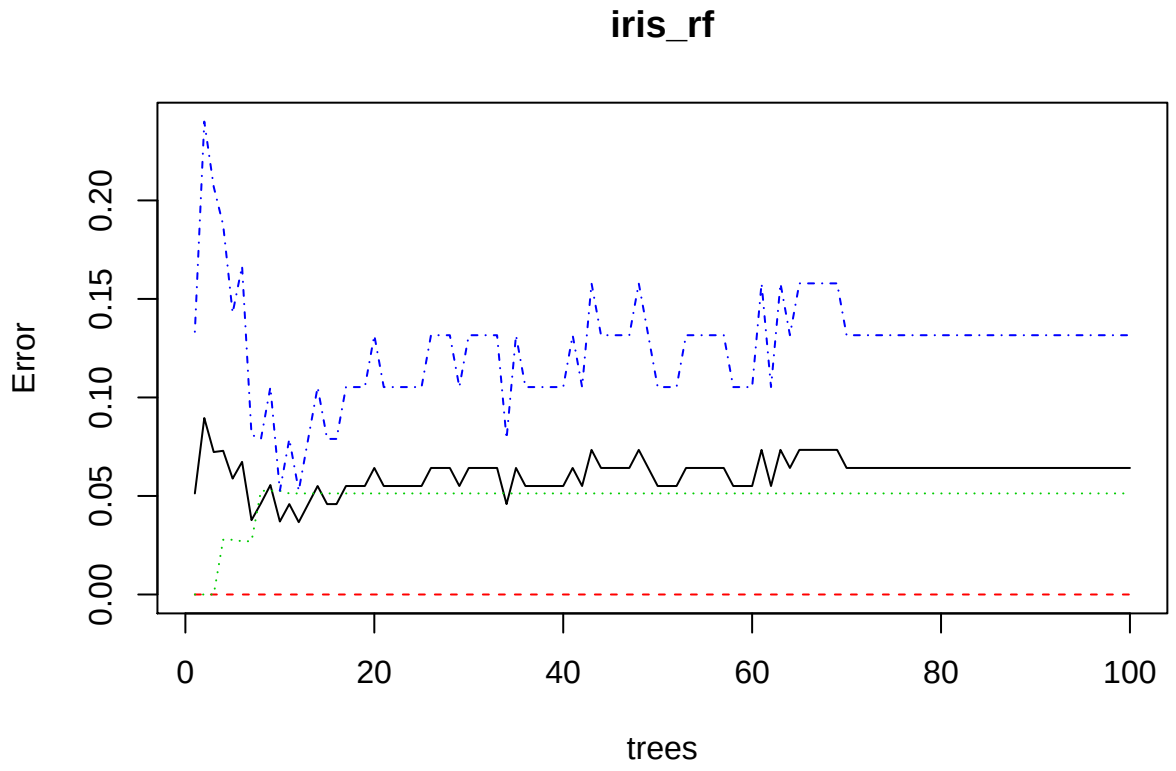
3. Try to print Random Forest model and see the importance features

```
print(iris_rf)
```

```
##
## Call:
##  randomForest(formula = Species ~ ., data = trainData, ntree = 100,      proximity = TRUE)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 6.42%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         32          0         0  0.00000000
## versicolor      0         37         2  0.05128205
```

```
## virginica       0          5         33  0.13157895
```

```
plot(iris_rf)
```



4. Assessing model fit

```
importance(iris_rf)
```

```
##              MeanDecreaseGini
## Sepal.Length         7.024487
## Sepal.Width          1.953323
## Petal.Length        32.304676
## Petal.Width         30.462743
```

```
varImpPlot(iris_rf)
```

5. Predict the class labels for test data

```
irisPred <- predict(iris_rf, newdata = testData)
irisPred
```

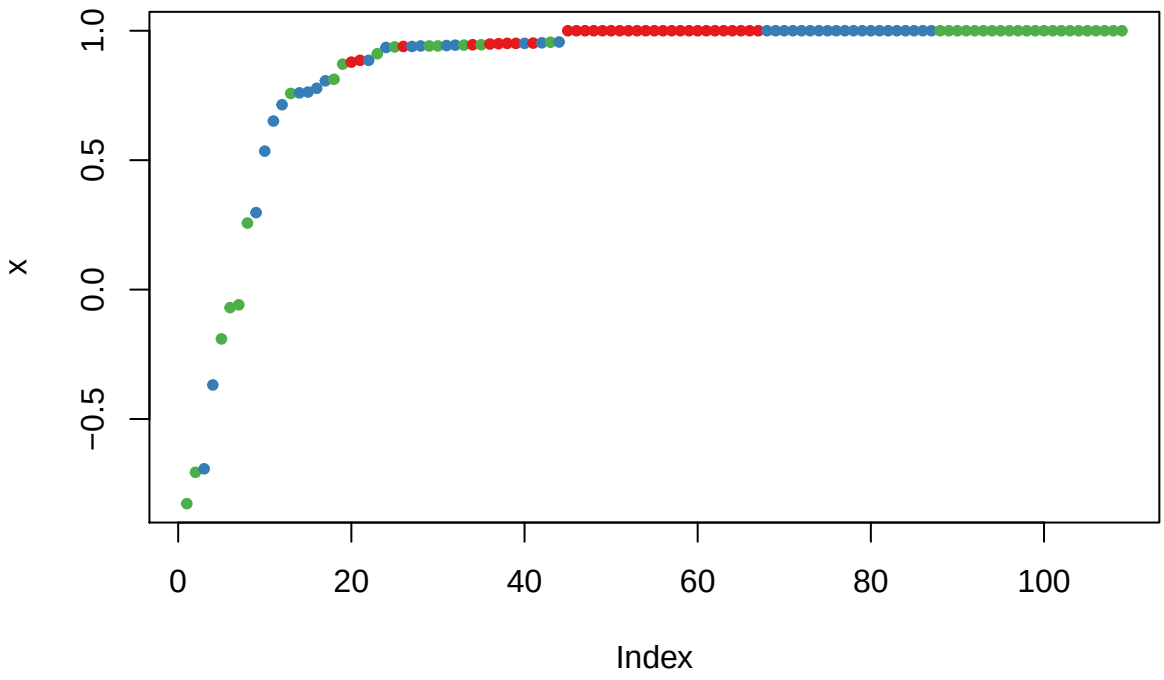
```
##          1          2          4          5          8         12
##     setosa     setosa     setosa     setosa     setosa     setosa
##         13         17         22         26         28         30
##     setosa     setosa     setosa     setosa     setosa     setosa
##         31         33         34         39         40         49
##     setosa     setosa     setosa     setosa     setosa     setosa
##         54         56         58         62         64         71
## versicolor versicolor versicolor versicolor versicolor  virginica
##         74         86         88         94         98        101
## versicolor versicolor versicolor versicolor versicolor  virginica
##        103        104        111        114        122        126
##  virginica  virginica  virginica  virginica  virginica  virginica
##        127        140        141        145        149
##  virginica  virginica  virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

```
table(irisPred, testData$Species)
```

```
##
## irisPred     setosa versicolor virginica
##   setosa         18          0         0
##   versicolor      0         10         0
##   virginica       0          1        12
```

6. Try to see the margin, positive or negative, if positif it means correct classification

```
plot(margin(iris_rf, testData$Species))
```



7. Tune randomForest for the optimal mtry parameter

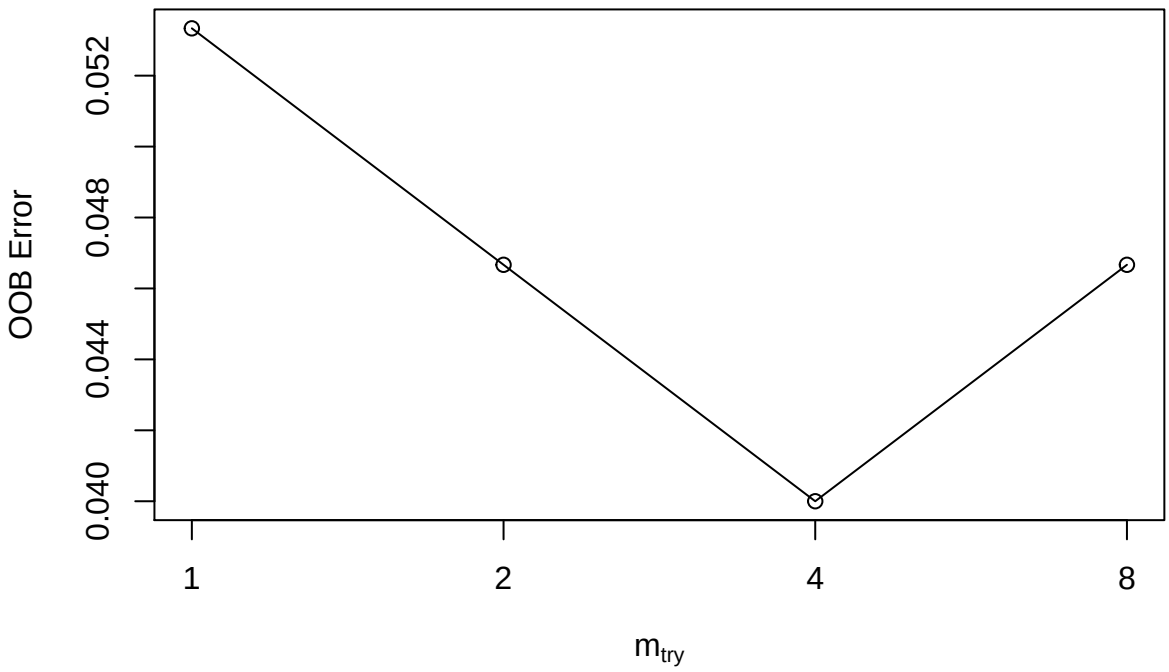
method 1

```
tune.rf <- tuneRF(iris[, -5], iris[, 5], stepFactor = 0.5)
```

```
## mtry = 2  OOB error = 4.67%
## Searching left ...
## mtry = 4     OOB error = 4%
## 0.1428571 0.05
```

```
## Warning in randomForest.default(x, y, mtry = mtryCur, ntree = ntreeTry, :
## invalid mtry: reset to within valid range
```

```
## mtry = 8     OOB error = 4.67%
## -0.1666667 0.05
## Searching right ...
## mtry = 1     OOB error = 5.33%
## -0.3333333 0.05
```




```
print(tune.rf)
```

```
##       mtry   OOBError
## 1.OOB    1 0.05333333
## 2.OOB    2 0.04666667
## 4.OOB    4 0.04000000
## 8.OOB    8 0.04666667
```

method 2

We can also tune the structure, ie, finding the best hyperparameters of the method via grid search

```
library(e1071)
tuned.r <- tune(randomForest, train.x = Species ~ ., data = trainData, validation.x = testData)
best.model <- tuned.r$best.model
best.model
```

```
##
## Call:
##  best.tune(method = randomForest, train.x = Species ~ ., data = trainData,      validation.x = testDa
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 5.5%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         32          0         0  0.00000000
```

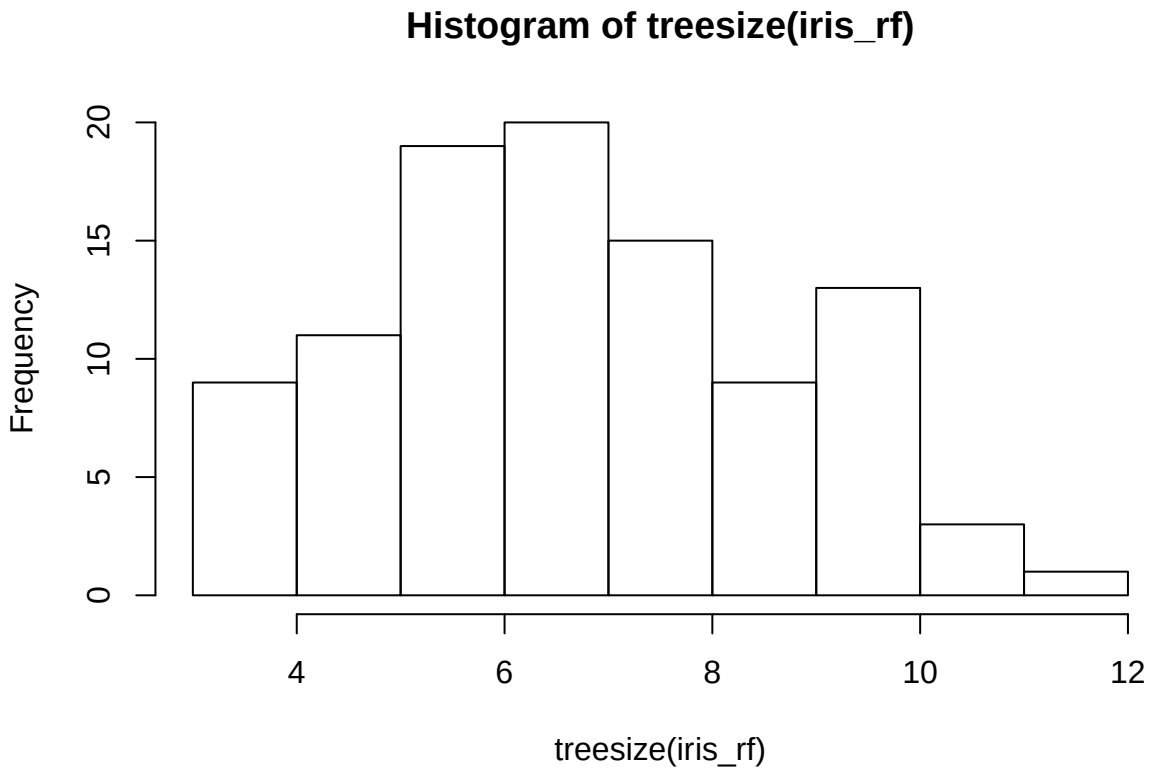
```
## versicolor      0         37         2  0.05128205
## virginica       0          4        34  0.10526316
```

```
predictions <- predict(best.model, testData)
table.random.forest <- table(testData$Species, predictions)
table.random.forest
```

```
##             predictions
##              setosa versicolor virginica
##   setosa         18          0         0
##   versicolor      0         10         1
##   virginica       0          0        12
```

8. Tree size

```
hist(treesize(iris_rf))
```



The randomForestSRC Package

```
library(randomForestSRC)
```

```
##
##  randomForestSRC 2.5.1
##
##  Type rfsrc.news() to see new features, changes, and bug fixes.
##
```

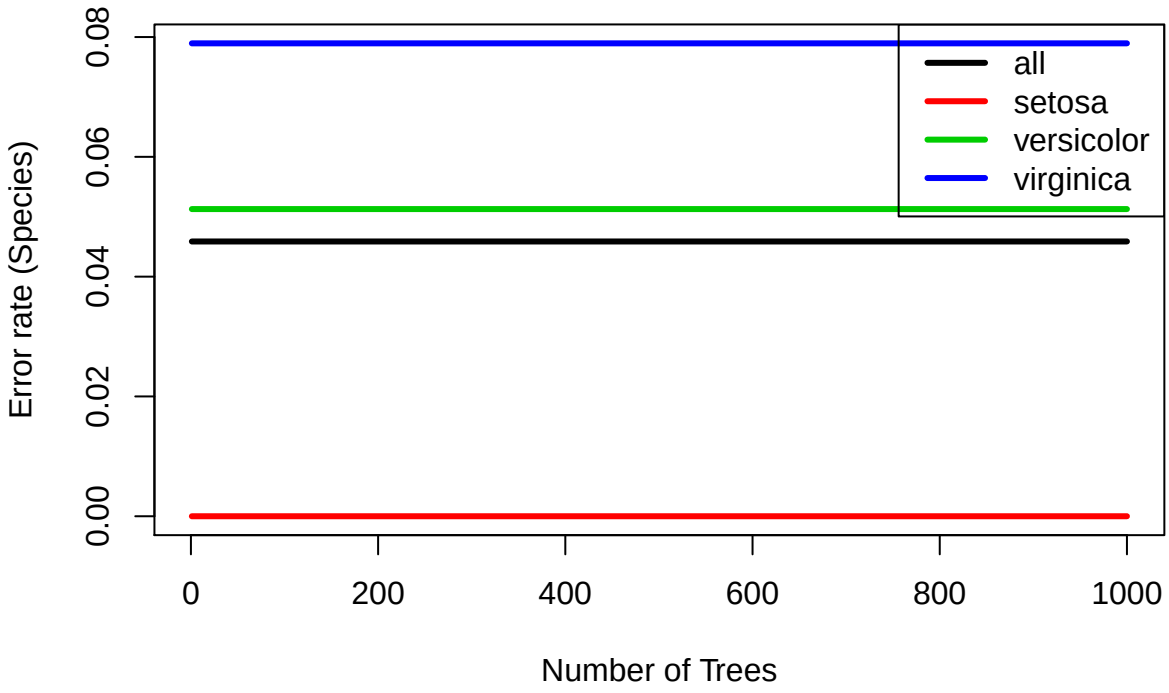
```
##
## Attaching package: 'randomForestSRC'
## The following object is masked from 'package:e1071':
##
##     impute
```

1. Fitting a random forest

```
## Edgar Anderson's iris data
iris.obj <- rfsrc(Species ~ ., data = trainData)
iris.obj
```

```
##                          Sample size: 109
##            Frequency of class labels: 32, 39, 38
##                      Number of trees: 1000
##            Forest terminal node size: 1
##        Average no. of terminal nodes: 7.308
## No. of variables tried at each split: 2
##               Total no. of variables: 4
##                             Analysis: RF-C
##                               Family: class
##                       Splitting rule: gini
##               Normalized Brier score: 12.65
##                           Error rate: 0.05, 0, 0.05, 0.08
##
## Confusion matrix:
##
##             predicted
##   observed   setosa versicolor virginica class.error
##   setosa         32          0         0      0.0000
##   versicolor      0         37         2      0.0513
##   virginica       0          3        35      0.0789
##
##  Overall error rate: 4.59%
```

```
plot(iris.obj)
```



2. predict based on the results of rfsrc

```
predict(iris.obj, testData)
```

```
##   Sample size of test (predict) data: 41
##                 Number of grow trees: 1000
##   Average no. of grow terminal nodes: 7.308
##           Total no. of grow variables: 4
##                               Analysis: RF-C
##                                 Family: class
##        Test set Normalized Brier score: 6.82
##                   Test set error rate: 0.02, 0, 0.09, 0
##
## Confusion matrix:
##
##              predicted
##   observed   setosa versicolor virginica class.error
##   setosa         18          0         0      0.0000
##   versicolor      0         10         1      0.0909
##   virginica       0          0        12      0.0000
##
##  Overall error rate: 2.44%
```

Parallel execution with random forest

```
library(randomForest)
library(doParallel)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
workers <- detectCores()
workers
```

```
## [1] 8
```

```
cl <- makePSOCKcluster(workers)
registerDoParallel(cl)

x <- matrix(runif(500), 100)
y <- gl(2, 50)
ntree <- 1000

rf <- foreach(n = rep(ceiling(ntree/workers), workers), .combine = combine,
    .multicombine = TRUE, .packages = "randomForest") %dopar% {
    randomForest(x, y, ntree = n)
}
```