Post 2: randomForest An Intro To Machine Learning

Danielle Chen

November 27, 2017

Using the randomForest Package: An Intro to Machine Learning

1. Introduction

A. Purpose

The purpose of this post is to educate all proficient or learning R developers on how to use randomForest an ensemble learning technique. According to the article, it is one of the commonly used predictive modelling and machine learning technique.

Quick notice, if you don't know what a decision tree is, check out this article. A non-technical description of an decision tree is a tree like representation of decisions which uses recursive partitioning of input data (called nodes) to meet specified conditions. The reason for recursive partitioning is to improve the measure of child nodes.

B. Motivation

My personal motivation on choosing randomForest in particular is that it gives a mini intro to machine learning, and decision trees in general. Since in this course we mainly focused on the graphical representation of statistics and data wrangling, I feel like this package is a healthy mix of using graphics while incorporating machine learning via decision trees.

C. Background

In this posting the main package we will be using is 'randomForest' the elaborated PDF documentation of the package and its information is here

The package was last updated in October 7th 2015 and is maintained by Andy Liaw. I will be using the latest version 4.6-12. The package PDF also suggests installing RColorBrewer and MASS.

The history of Random Forest is well described in this article. Tim Kam Ho was the first one to create the algorithm for random decision forests. He did that by using the random subspace method. Leo Breiman and Adele Cutler were the ones that developed the trademark "Random Forests," which is an extension to Ho's algorithm.

2. Examples

For replication purposes here is my verion of R. Moreover, this is on a Dell, Windows laptop.

```
version
```

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          3
## minor          4.2
## year           2017
## month          09
## day            28
## svn rev        73368
## language       R
## version.string R version 3.4.2 (2017-09-28)
## nickname       Short Summer
```

Moreover, ALL of the packages are the latest versions as of 12/3

You'll need to install randomForest first:

```
#install randomForest the package we'll be using
install.packages("randomForest")
#package should be version 4.6-12.
```

```
#load all packages into the console
#some of the packages may or may not be used
#more packages will be used and explained in the examples below
library(randomForest)
```

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

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

```
library(MASS)
library(RColorBrewer)
```

So what exactly is randomForest?

randomForest is a nice learning method for classification and many other things like regression as it creates a large number of decision trees that are created under the different parameters and subsets from the original data set. It then combines the output of trees, which is the ensemble method, to improve generalization ability of the model. Ensembling is the combination of weak learners (individual trees) to produce a strong learner.

A real life example of the basics of randomForest (mainly the ensemble method) is if you are deciding on if a class is good or not. To help decide you ask ten people on how they liked the class. If say 8 of them say the class is, “amazing and extremely insightful” the majority favors the class; as a result, you choose to take that class.

Building many decision trees results in a forest. A random forest works the following way:

- Pulled from this article

> - First, it uses the Bagging (Bootstrap Aggregating) algorithm to create random samples. Given a data set D1 (n rows and p columns), it creates a new dataset (D2) by sampling n cases at random with replacement from the original data. About 1/3 of the rows from D1 are left out, known as Out of Bag(OOB) samples.
> - Then, the model trains on D2. OOB sample is used to determine unbiased estimate of the error.
> - Out of p columns, P << p columns are selected at each node in the data set. The P columns are selected at random. Usually, the default choice of P is p/3 for regression tree and P is sqrt(p) for classification tree. -Unlike a tree, no pruning takes place in random forest; i.e, each tree is grown fully. In decision trees, pruning is a method to avoid overfitting. Pruning means selecting a subtree that leads to the lowest test errror rate. We can use cross validation to determine the test error rate of a subtree.
> - Several trees are grown and the final prediction is obtained by averaging or voting.
> - Each tree is grown on a different sample of original data. Since random forest has the feature to calculate OOB error internally, cross validation doesn’t make much sense in random forest.

Before we start coding for random Forest, we first need to understand how to use it. The random Forest package has a TON of functions. We won’t be doing all of them as that would take this post to be **super** long; however, I will go through the majority of the functions. To start off here is a list of all of the functions and a short description that was presented in the PDF document linked above.

> - classCenter: prototypes of groups
> - combine: combines ensembles of trees
> - getTree: extract a single tree from a forest
> - grow: add trees to an assembly
> - importance: extract variable importance measure
> - import85: the automobile data
> - margin: margins of randomForest classifier
> - MDSplot: multidimensional scaling plot of proximity matrix from randomForest
> - na.roughfix: rough imputation of missing values
> - outlier: compute outlying measure
> - partialPlot: partial dependence plot
> - plot.randomForest: plot method for randomforest objects
> - predict.randomForest: predict method for random forest objects
> - randomForest: classification and regression with random forest
> - rfcv: random forest cross-validation for feature selection
> - rfImpute: missing value imputations by randomForest
> - rfNews: show the NEWS files
> - treesize: size of trees in an ensemble
> - tuneRF: tune randomForest for the optimal mtry parameter
> - varImpPlot: variable importance plot
> - varUsed: variable used in a random forest

Some key features of Random Forests Includes:

*Pulled from this website

> - It runs efficiently on large data bases.
> - It can handle thousands of input variables without variable deletion.

- It gives estimates of what variables are important in the classification.
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- It has methods for balancing error in class population unbalanced data sets. Generated forests can be saved for future use on other data. Prototypes are computed that give information about the relation between the variables and the classification.
- It computes proximities between pairs of cases that can be used in clustering, locating outliers, or (by scaling) give interesting views of the data. The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.
- It offers an experimental method for detecting variable interactions.
- It is unexcelled in accuracy among current algorithms.

With all of the advantages of randomForest, there are some disadvantages of randomForest as described in the article:

- It surely does a good job at classification but not as good as for regression problem as it does not give precise continuous nature predictions. In case of regression, it doesn't predict beyond the range in the training data, and that they may over-fit data sets that are particularly noisy.
- Random Forest can feel like a black box approach for statistical modelers - you have very little control on what the model does. You can at best - try different parameters and random seeds!

Let's go over a regression example from thisblog post.

The code in this example is pulled straight from the blog.If you want more in depth explanation for each step refer to the blog post.

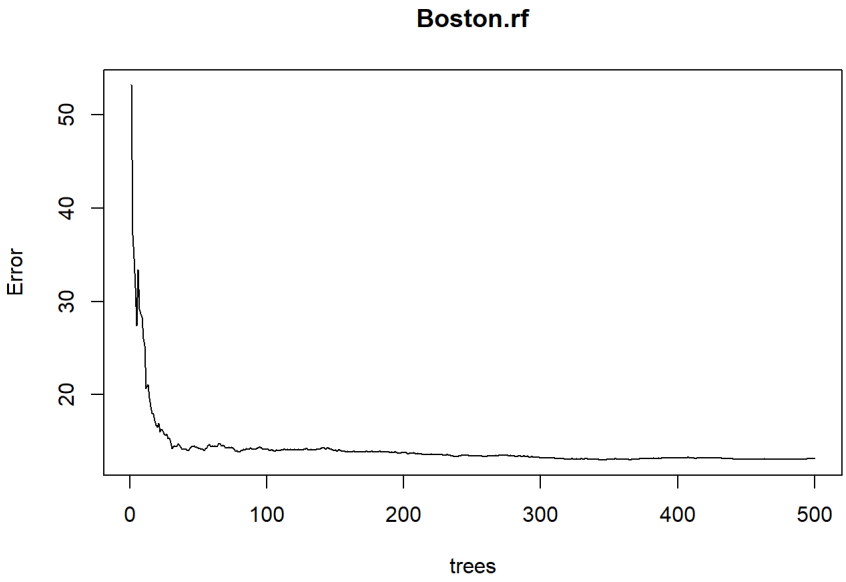
```
# step 1: Implementation in R
# loading the required packages
require(randomForest)
require(MASS) #this contains Boston housing dataset

attach(Boston) #dataset we will work with
set.seed(100) #generate 100 random nums
dim(Boston) #get the dimensions of Boston dataset
```

```
## [1] 506  14
```

```
# Step 2: Separating Training and Test Sets
#training Sample with 300 observations
train <- sample(1:nrow(Boston),300)


# Step 3:Fitting the Random Forest
Boston.rf <- randomForest(medv ~ . , data = Boston , subset = train)
Boston.rf #.rf stands for datasets after randomforest
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = Boston, subset = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 13.15948
##                     % Var explained: 84.39
```

```
# Step 4: Plotting the Error vs Number of Trees Graph.
plot(Boston.rf)
```




```
#This plot shows the Error and the Number of Trees.
#Notice how the Error is dropping as more trees are added


#Step 5: Compare the Out of Bag Sample Errors and Error on Test set

# Random Forest chose 4 variables randomly to be considered at each split.
# Now try all possible 13 predictors which can be found at each split.

oob.err=double(13) #oob stands for out of bag
test.err=double(13) # testing the 13 predicators

#mtry is no of Variables randomly chosen at each split
for(mtry in 1:13)
{
  rf=randomForest(medv ~ . , data = Boston , subset = train,mtry=mtry,ntree=400)
  oob.err[mtry] = rf$mse[400] #Error of all Trees fitted

  pred<-predict(rf,Boston[-train,]) #Predictions on Test Set for each Tree
  test.err[mtry]= with(Boston[-train,], mean( (medv - pred)^2)) #Mean Squared Test Error

  cat(mtry," ") #printing the output to the console
}
```

```
## 1  2  3  4  5  6  7  8  9  10  11  12  13
```

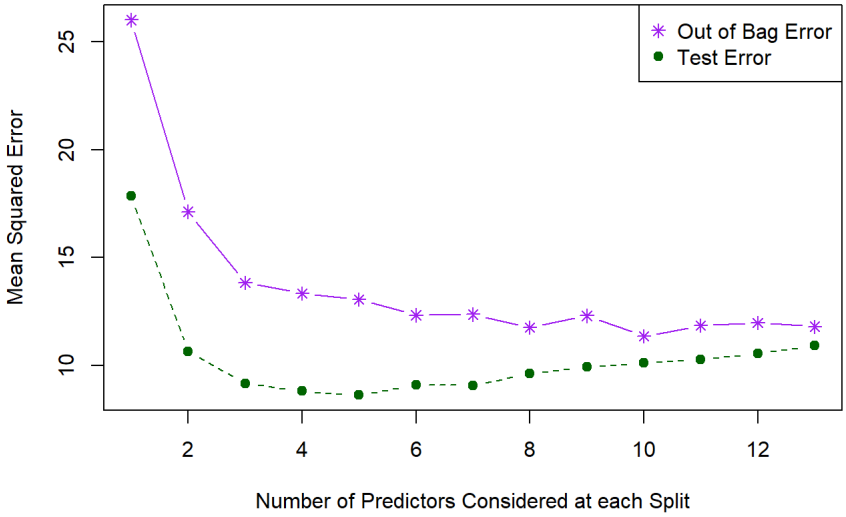
```
#Step 6: get the errors
#this returns the mean squared error for each of the 13 predicators

#Test Error
test.err
```

```
##  [1] 17.864469 10.634574  9.158720  8.801375  8.617153  9.097007  9.070912
##  [8]  9.607225  9.923911 10.101269 10.271418 10.557313 10.905275
```

```
#Out of Bag Error Estimation
oob.err
```

```
##  [1] 26.01397 17.11895 13.83049 13.33001 13.04343 12.32336 12.35522
##  [8] 11.75557 12.29802 11.34876 11.83531 11.96986 11.79109
```

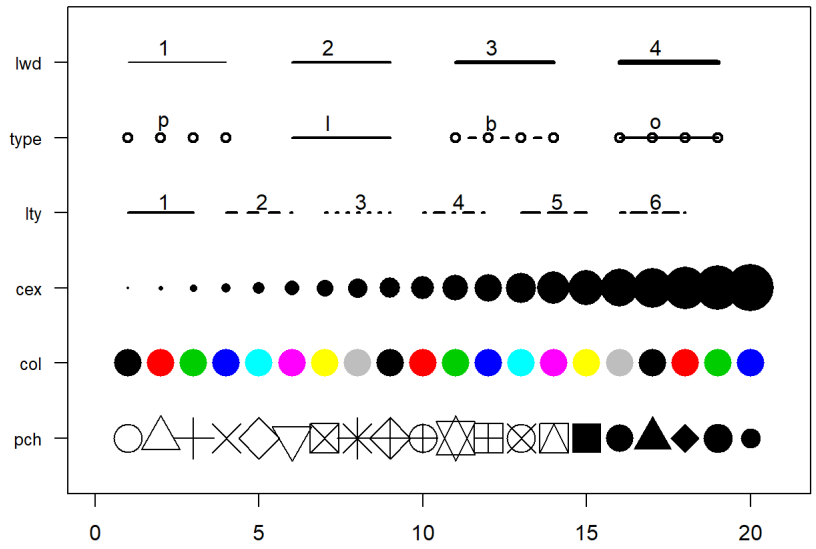
```
#Step 7:Plotting both Test Error and Out of Bag Error
matplot(1:mtry , cbind(oob.err,test.err), pch=c(8,19), col=c("purple","darkgreen"),
type="b",ylab="Mean Squared Error",xlab="Number of Predictors Considered at each Split")
legend("topright",legend=c("Out of Bag Error","Test Error"),pch=c(8,19), col=c("purple","darkgreen"))
```




```
#in the plot design feel free to change colors, pch, and
```

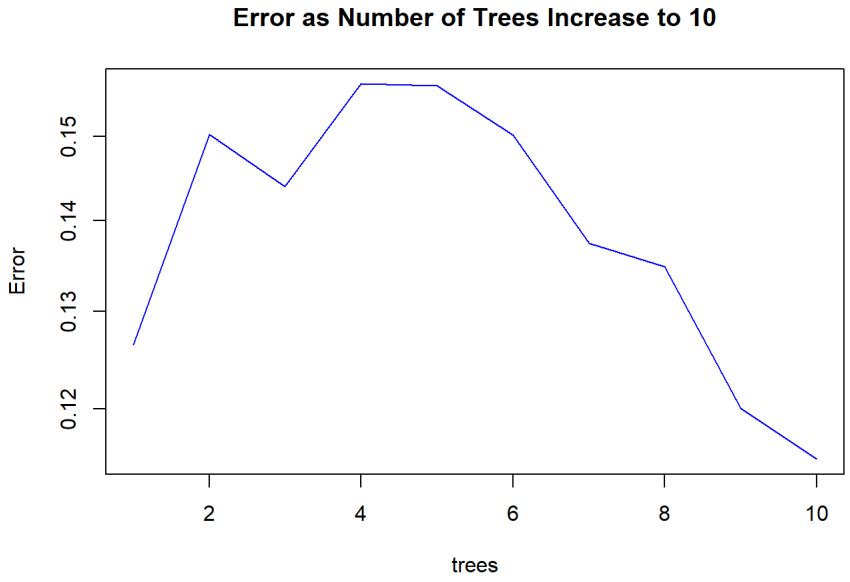
slightly off topic but here are the different parameters for graphs you DO NOT replicate this. It is just for you too see the graph to decide parameters of the graph plot example above. * code was pulled from this website

```
 #initialization
par(mar=c(3,3,3,3))
num=0 ; num1=0
plot(0,0 , xlim=c(0,21) , ylim=c(0.5,6.5), col="white" , yaxt="n" , ylab="" , xlab="")

#fill the graph
for (i in seq(1,20)){
  points(i,1 , pch=i , cex=3)
  points(i,2 , col=i , pch=16 , cex=3)
  points(i,3 , col="black" , pch=16 , cex=i*0.25)

  #lty
  if(i %in% c(seq(1,18,3))){
        num=num+1
    points(c(i,i+2), c(4,4) , col="black" , lty=num , type="l" , lwd=2)
        text(i+1.1 , 4.15 , num)
        }

  #type and lwd
  if(i %in% c(seq(1,20,5))){
    num1=num1+1
    points(c(i,i+1,i+2,i+3), c(5,5,5,5) , col="black"  , type=c("p","l","b","o")[num1] , lwd=2)
    text(i+1.1 , 5.2 , c("p","l","b","o")[num1] )
    points(c(i,i+1,i+2,i+3), c(6,6,6,6) , col="black"  , type="l",  lwd=num1)
    text(i+1.1 , 6.2 , num1 )

    }
  }

#add axis
axis(2, at = c(1,2,3,4,5,6), labels = c("pch" , "col" , "cex" , "lty", "type" , "lwd" ),
     tick = TRUE, col = "black", las = 1, cex.axis = 0.8)
```

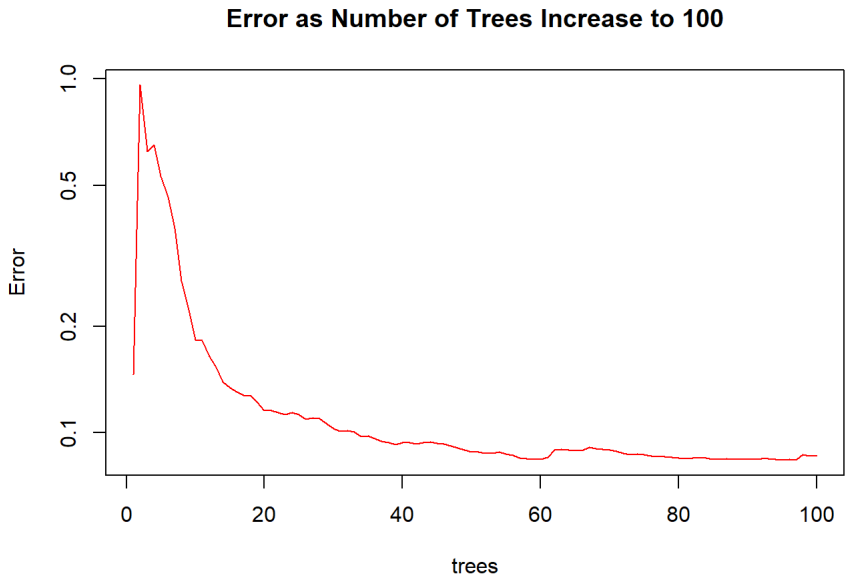


Here is another example using iris

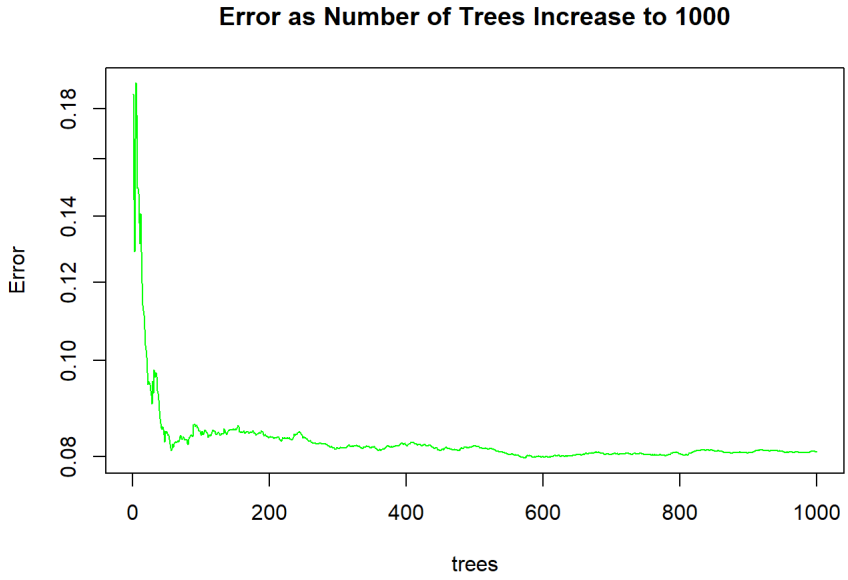
```
#load dataset iris
# install.packages("datasets") <-if you haven't already
library(datasets)
data(iris)

#see that as the num of trees increase the error levels out law of large num
plot(randomForest(Petal.Length ~ ., iris, keep.forest=FALSE, ntree=10),
     main = "Error as Number of Trees Increase to 10", log="y",
     col = "blue")
```



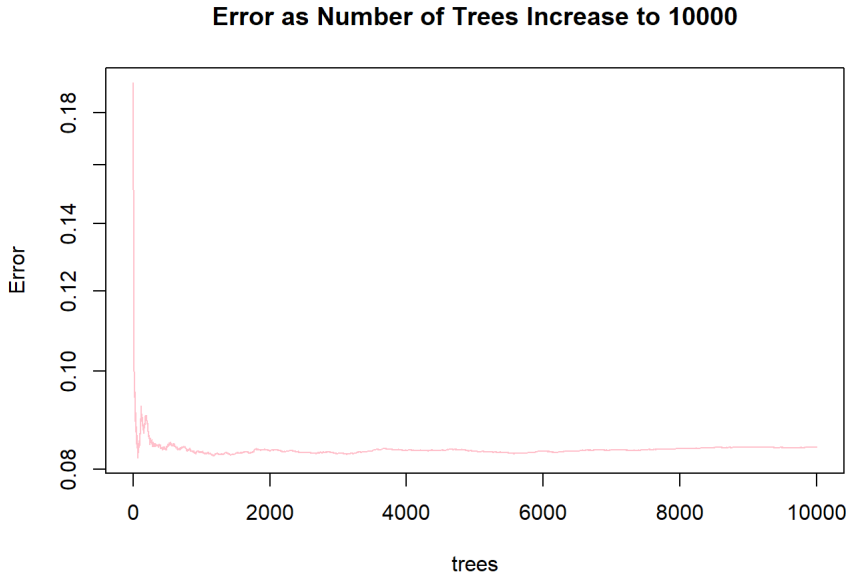

```
plot(randomForest(Petal.Length ~ ., iris, keep.forest=FALSE, ntree=100),
     main = "Error as Number of Trees Increase to 100", log="y",
     col = "red")
```




```
plot(randomForest(Petal.Length ~ ., iris, keep.forest=FALSE, ntree=1000),
     main = "Error as Number of Trees Increase to 1000", log="y",
     col = "green")
```




```
plot(randomForest(Petal.Length ~ ., iris, keep.forest=FALSE, ntree=10000),
     main = "Error as Number of Trees Increase to 10000", log="y",
     col = "pink")
```



The next two examples will be from this article

- for further info check out the article above

```
#A classification example to show how random forests work
# install.packages("MASS") <- if you haven't already
library(randomForest)
library(MASS) #load MASS for dataset

#using Measurements of Forensic Glass Fragments dataset aka fgl
data(fgl)

set.seed(100) #100 random num generated

#making dataset to be random forest
fgl.rf <- randomForest(type ~ ., data = fgl,
mtry = 2, importance = TRUE,
do.trace = 100)
```

```
## ntree      OOB      1      2      3      4      5      6
##   100:  22.43% 15.71% 23.68% 64.71% 23.08% 22.22% 10.34%
##   200:  20.56% 12.86% 19.74% 58.82% 30.77% 22.22% 13.79%
##   300:  20.56% 11.43% 19.74% 64.71% 30.77% 22.22% 13.79%
##   400:  19.16% 11.43% 17.11% 58.82% 38.46% 11.11% 13.79%
##   500:  20.09% 11.43% 18.42% 64.71% 38.46% 11.11% 13.79%
```

```
print(fgl.rf) #showing the function outputs
```

```
##
## Call:
##  randomForest(formula = type ~ ., data = fgl, mtry = 2, importance = TRUE,      do.trace = 100)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 20.09%
## Confusion matrix:
##       WinF WinNF Veh Con Tabl Head class.error
## WinF    62     7   1   0    0    0   0.1142857
## WinNF    9    62   1   1    1    2   0.1842105
## Veh      8     3   6   0    0    0   0.6470588
## Con      0     4   0   8    0    1   0.3846154
## Tabl     0     1   0   0    8    0   0.1111111
## Head     1     3   0   0    0   25   0.1379310
```

Now compare random forests with support vector machines by doing ten repetitions of 10-fold cross-validation, using the errorest functions in the ipred package:

```
#install.packages("ipred")
library(ipred) #install and load ipred
set.seed(131)
error.RF <- numeric(10)
for(i in 1:10) error.RF[i] <-
errorest(type ~ ., data = fgl,
model = randomForest, mtry = 2)$error
summary(error.RF)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1822  0.1939  0.2079  0.2065  0.2150  0.2383
```

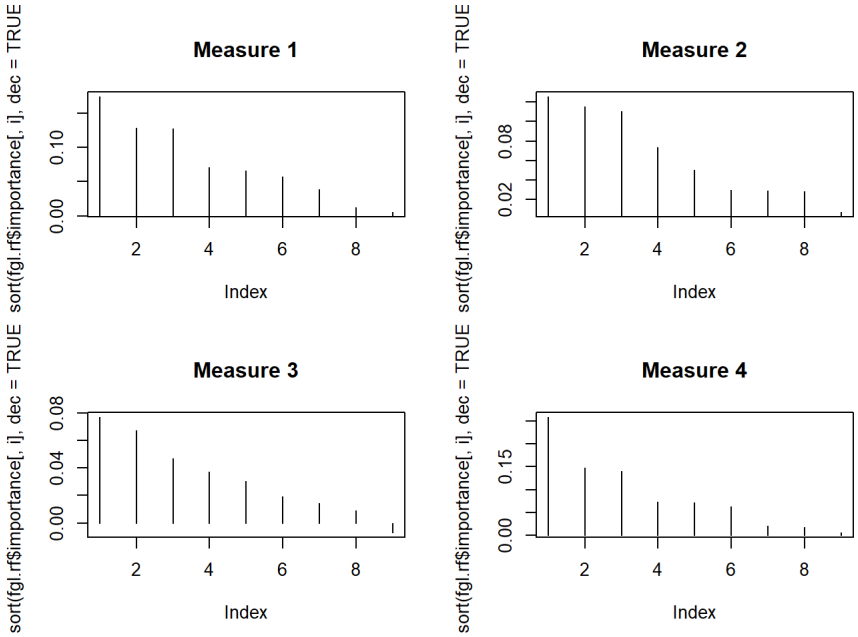
```
#install.packages("e1071")
library(e1071) #install and load e1071
set.seed(563)
error.SVM <- numeric(10)
for (i in 1:10) error.SVM[i] <-
errorest(type ~ ., data = fgl,
model = svm, cost = 10, gamma = 1.5)$error
summary(error.SVM)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.3645  0.3785  0.3808  0.3794  0.3832  0.3879
```

```
#notice that the random forest compares quite favorably with SVM.

#plotting variable importance for data
fgl.rf <- randomForest(type ~ ., data = fgl,
mtry = 2, importance = TRUE,
do.trace = 100)
```

```
## ntree      OOB      1      2      3      4      5      6
##   100:  23.36% 17.14% 19.74% 58.82% 46.15% 22.22% 17.24%
##   200:  19.63% 12.86% 14.47% 64.71% 38.46% 22.22% 13.79%
##   300:  21.03% 14.29% 17.11% 64.71% 38.46% 22.22% 13.79%
##   400:  19.16% 11.43% 15.79% 64.71% 38.46% 11.11% 13.79%
##   500:  20.09% 12.86% 17.11% 64.71% 38.46% 11.11% 13.79%
```

```
par(mfrow = c(2, 2))
for (i in 1:4)
plot(sort(fgl.rf$importance[,i], dec = TRUE),
type = "h", main = paste("Measure", i))
```



3. Discussion

Overall using the package randomForests is a nice algorithm to know and have in your repertoire as we progress closer to machine learning and big data in society. With its multitude of functions this algorithm is best for classification and regression. Make sure you do read some if not all of the articles and references listed as this post could only cover the basics. Also this topic is very tricky (it took me a LONG time to understand random forests and how to use it), so do further research as needed and don't feel discouraged if you get confused. All in all, randomForest is something that can either be very simple or extremely complex and it is useful for all types of coders.

4. Conclusion

Although there is many ways to go into machine learning, I feel like randomForests is a perfect way to get introduced to this growingly popular topic. This post only touched upon the mere basics of this package. The main takeaway is to learn more about machine learning in particular decision trees. I wrote this post to give a basic insight on machine learning and to spark an interest to learn more about machine learning and how to use it in R.

5. References

- http://dni-institute.in/blogs/random-forest-using-r-step-by-step-tutorial/
- http://dni-institute.in/blogs/decision-tree-a-statistical-and-analytical-tool-for-effective-decisions/
- https://en.wikipedia.org/wiki/Random_forest
- https://en.wikipedia.org/wiki/Random_subspace_method
- https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#features
- https://www.r-bloggers.com/random-forests-in-r/
- http://www.bios.unc.edu/~dzeng/BIOS740/randomforest.pdf
- https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/#nine
- https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/tutorial-random-forest-parameter-tuning-r/tutorial/
- https://www.rdocumentation.org/packages/randomForest/versions/4.6-12/topics/randomForest
- http://www.r-graph-gallery.com/6-graph-parameters-reminder/