Introduction to Random Forest

Kaiqian Zhang & Dennis Liu

7/15/2017

Packages Installed: randomForest, caret, and pROC

Random forests are an ensemble learning method for classification (and regression) that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes output by individual trees (Wikipedia)

# randomForest is a package for creating random forests  
library(randomForest)

## randomForest 4.6-12

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

# caret is a package we use for data splitting  
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

##   
## Attaching package: 'ggplot2'

## The following object is masked from 'package:randomForest':  
##   
## margin

# pROC is a package for plotting ROC curve  
library(pROC)

## Type 'citation("pROC")' for a citation.

##   
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':  
##   
## cov, smooth, var

### General Algorithm

We can first build a random forest simply by using a R package ‘randomForest’. The data we use here is ‘iris’ data. And we start with a simple random forest with only three trees.

# Data summary  
head(iris)

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
## 1 5.1 3.5 1.4 0.2 setosa  
## 2 4.9 3.0 1.4 0.2 setosa  
## 3 4.7 3.2 1.3 0.2 setosa  
## 4 4.6 3.1 1.5 0.2 setosa  
## 5 5.0 3.6 1.4 0.2 setosa  
## 6 5.4 3.9 1.7 0.4 setosa

# Create a single 2/3 split of the iris data  
set.seed(1011)  
trainIndex <- createDataPartition(iris$Species, p = 2/3,   
 list = FALSE,   
 times = 1)   
# list: should the results be in a list or not; times: the number of partitions to create;   
irisTrain <- iris[trainIndex,]  
irisTest <- iris[-trainIndex,]  
  
# Build a random forest   
# importance: should importance of predictors be assessed?  
# ntree: number of trees in the forest  
# mtry: number of veriables randomly sampled at each split when building the tree  
# do.trace: whether to give a more detailed output as randomForest is run  
forest1 <- randomForest(Species ~ ., data = irisTrain, importance=TRUE, ntree=3, mtry = 2, do.trace=TRUE)

## ntree OOB 1 2 3  
## 1: 8.33% 0.00% 7.69% 20.00%  
## 2: 6.45% 0.00% 4.35% 15.00%  
## 3: 5.33% 0.00% 4.00% 12.50%

forest1

##   
## Call:  
## randomForest(formula = Species ~ ., data = irisTrain, importance = TRUE, ntree = 3, mtry = 2, do.trace = TRUE)   
## Type of random forest: classification  
## Number of trees: 3  
## No. of variables tried at each split: 2  
##   
## OOB estimate of error rate: 5.33%  
## Confusion matrix:  
## setosa versicolor virginica class.error  
## setosa 26 0 0 0.000  
## versicolor 0 24 1 0.040  
## virginica 0 3 21 0.125

Notice that OOB error estimate is called out-of-bag error rate. This is used to measure the prediction error of random forests. OOB is the mean prediction error on each training sample xᵢ, using only the trees that did not have xᵢ in their bootstrap sample. Further, confusion matrix is produced based on OOB. It is calculated at a specific point determined by the cutoff on the votes.

The algorithm for producing the random forest model ‘forest1’ is:

* sample three times from the training set since we set ntree = 3.
* build each of three classification trees individually.
* randomly select two variables at each split when constructing each individual tree since we set mtry = 2.

### Visualize Individual Trees

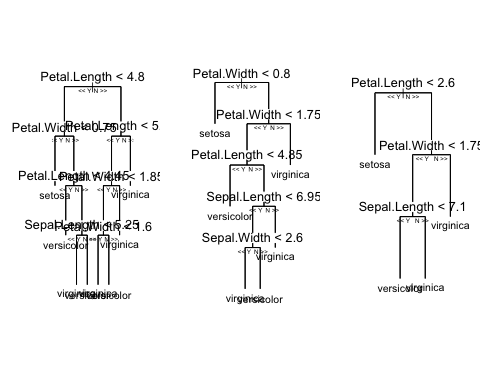
Even though random forest is like a blackbox, we are still able to visualize individual tress by using a R package ‘reprtree’.

Please first instll the package by using the following code:

# Install reprtree package  
options(repos='http://cran.rstudio.org')  
have.packages <- installed.packages()  
cran.packages <- c('devtools','plotrix','randomForest','tree')  
to.install <- setdiff(cran.packages, have.packages[,1])  
if(length(to.install)>0) install.packages(to.install)  
library(devtools)  
if(!('reprtree' %in% installed.packages())){  
 install\_github('araastat/reprtree')  
}  
for(p in c(cran.packages, 'reprtree')) eval(substitute(library(pkg), list(pkg=p)))

Now we can visualize exactly what three sampled trees are.

library(reprtree)  
par(mfrow=c(1,3))  
reprtree:::plot.getTree(forest1, k = 1)  
reprtree:::plot.getTree(forest1, k = 2)  
reprtree:::plot.getTree(forest1, k = 3)



### Predictions on Testing

Next, we can test our data by using our model forest1. We observe that the random forest model predicts that observation 1 is setosa, observation 2 is setosa, and etc.

iris.pred <- predict(forest1, irisTest)  
iris.pred

## 1 2 8 9 12 17   
## setosa setosa setosa setosa setosa setosa   
## 18 19 21 23 27 28   
## setosa setosa setosa setosa setosa setosa   
## 30 40 43 45 52 53   
## setosa setosa setosa setosa versicolor versicolor   
## 55 56 58 60 64 71   
## versicolor versicolor versicolor versicolor versicolor virginica   
## 73 77 83 85 93 95   
## virginica versicolor versicolor versicolor versicolor versicolor   
## 97 99 103 104 108 109   
## versicolor versicolor virginica virginica virginica virginica   
## 110 111 118 123 127 128   
## virginica virginica virginica virginica virginica virginica   
## 133 134 135 137 147 148   
## virginica versicolor virginica virginica virginica virginica   
## Levels: setosa versicolor virginica

We can obtain predicted results of the remaining 48 testing data. We will examine in detail on one test input to illustrate how the forest1 model work for prediction.

Say we choose observation 64 (Sepal.Length = 6.1, Sepal.Width = 2.9, Petal.Length = 4.7, Petal.Width = 1.4) and observation 73 (Sepal.Length = 6.3, Sepal.Width = 2.5, Petal.Length = 4.9, Petal.Width = 1.5) . Applying these information on each of three trees, we get a table of predictions for observation.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Observation | Tree 1 | Tree 2 | Tree 3 | Prediction | Actual |
| 64 | versicolor | versicolor | versicolor | versicolor | versicolor |
| 73 | virginica | virginica | versicolor | virginica | versicolor |

Notice that the final prediction takes majority votes from three trees. The prediction for observation 64 is versicolor since versicolor gets three votes. Likewise, the prediction for observation 73 is virginica since this type gets two out of three votes. We could also check prediction in the ‘iris.pred’ above.

To obtain a summary of all 48 testing data under the random forest model, we can form a table:

table(observed = irisTest[, "Species"], predicted = iris.pred)

## predicted  
## observed setosa versicolor virginica  
## setosa 16 0 0  
## versicolor 0 14 2  
## virginica 0 1 15

We notice that numbers on the diagonal are well-predicted. So we can compute the prediction accuracy by adding diagonals together and dividing by 48: accuracy = (16+14+15) / 48 = 93.75%.

### Using Caret to Create Random Forests

Caret is also powerful in developing random forests. In this section, we will use train() function in caret to create a random forest. To make it simple, we remove setosa data. We only have two responses: virginica and versicolor in this case.

# Clear our data  
rm(iris)

## Warning in rm(iris): object 'iris' not found

# Remove setosa data from iris  
iris <- iris[iris$Species == "virginica" | iris$Species == "versicolor", ]  
iris$Species <- factor(iris$Species)   
# Set up training and testing sets as before   
set.seed(1013)  
trainIndex <- createDataPartition(iris$Species, p = 2/3,   
 list = FALSE,   
 times = 1)   
irisTrain <- iris[trainIndex,]  
irisTest <- iris[-trainIndex,]  
# Create a random forest using train()  
forest.model <- train(Species ~., data = irisTrain)  
forest.model$finalModel

##   
## Call:  
## randomForest(x = x, y = y, mtry = param$mtry)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 2  
##   
## OOB estimate of error rate: 7.35%  
## Confusion matrix:  
## versicolor virginica class.error  
## versicolor 32 2 0.05882353  
## virginica 3 31 0.08823529

Now we proceed to evaluate final model by applying final model on our testing set.

iris.pred <- predict(forest.model, irisTest)  
with(irisTest, table(iris.pred, Species))

## Species  
## iris.pred versicolor virginica  
## versicolor 15 0  
## virginica 1 16

When discussing the predicted outcomes for a binary response data set, we use the following types of error to describe this.

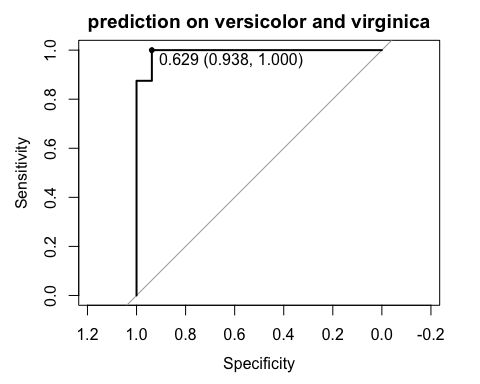
* True positive: correctly identitified (predicted true when true)
* False positive: incorrectly identified (predicted true when false)
* True negative: correctly rejected (predicted false when false)
* False negative: incorrectly rejected (predicted false when true)

In our example, suppose versicolor is true and virginica is false. We have 15 true positives, 0 false positive, 16 true negatives, and 1 false negative. We can compute accuracy by using the formula:

accuracy = (true positive + true negative) / total = (15+16) / (15+16+1+0) = 96.875%.

We can further visualize the accuracy for this binary data by plotting ROC curve. The ROC (receiver operating characteristic) curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

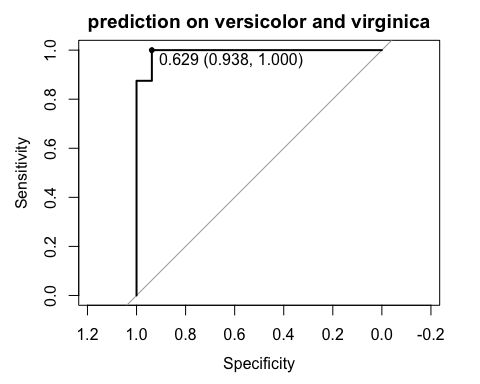
# Predict probability   
iris.pred.prob <- predict(forest.model, irisTest, type="prob")  
# Draw ROC curve  
result.roc <- roc(irisTest$Species, iris.pred.prob$versicolor)  
plot(result.roc, print.thres="best", print.thres.best.method="closest.topleft", main = "prediction on versicolor and virginica")



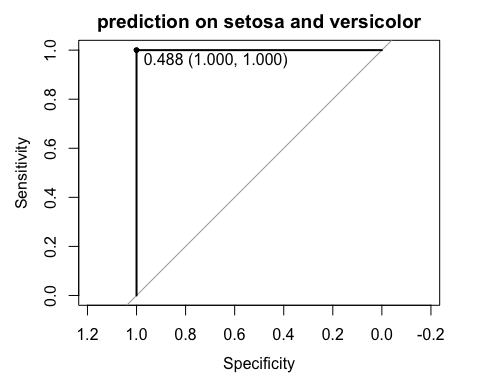
The area under ROC curve (AUC) indicates how well the prediction peforms.

We can run two other analyses on setosa and virsicolor, and virginica and setosa. Two more ROC curves can be plotted. The following code just repeats the above process for our questions of interests.

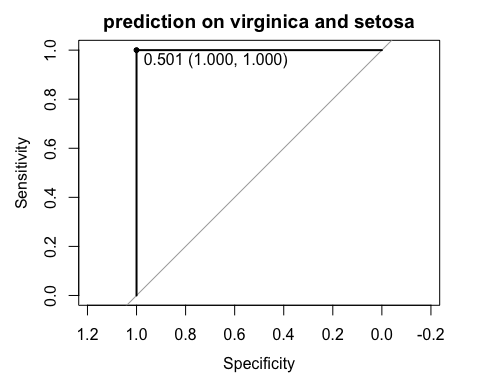
# Analysis on setosa and versicolor  
rm(iris)  
iris <- iris[iris$Species == "setosa" | iris$Species == "versicolor", ]  
iris$Species <- factor(iris$Species)   
set.seed(1015)  
trainIndex <- createDataPartition(iris$Species, p = 2/3,   
 list = FALSE,   
 times = 1)   
irisTrain <- iris[trainIndex,]  
irisTest <- iris[-trainIndex,]  
forest.model1 <- train(Species ~., data = irisTrain)  
iris.pred.prob1 <- predict(forest.model1, irisTest, type="prob")  
result.roc1 <- roc(irisTest$Species, iris.pred.prob1$setosa)  
  
# Analysis on virginica and setosa  
rm(iris)  
iris <- iris[iris$Species == "setosa" | iris$Species == "virginica", ]  
iris$Species <- factor(iris$Species)   
set.seed(1015)  
trainIndex <- createDataPartition(iris$Species, p = 2/3,   
 list = FALSE,   
 times = 1)   
irisTrain <- iris[trainIndex,]  
irisTest <- iris[-trainIndex,]  
forest.model2 <- train(Species ~., data = irisTrain)  
iris.pred.prob2 <- predict(forest.model2, irisTest, type="prob")  
result.roc2 <- roc(irisTest$Species, iris.pred.prob2$virginica)  
  
# Put three ROC curves together  
plot(result.roc, print.thres="best", print.thres.best.method="closest.topleft", main = "prediction on versicolor and virginica")



plot(result.roc1, print.thres="best", print.thres.best.method="closest.topleft", main = "prediction on setosa and versicolor")



plot(result.roc2, print.thres="best", print.thres.best.method="closest.topleft", main = "prediction on virginica and setosa")



### References

* <https://www.r-bloggers.com/in-depth-introduction-to-machine-learning-in-15-hours-of-expert-videos>
* <https://en.wikipedia.org/wiki/Random_forest>
* <https://github.com/araastat/reprtree/tree/master/man>