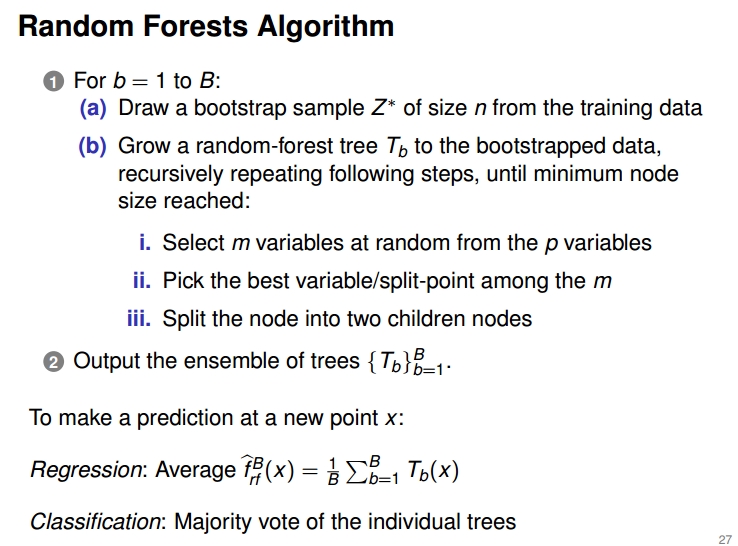
Random Forest of Handwritten Digits

## Basic Theory

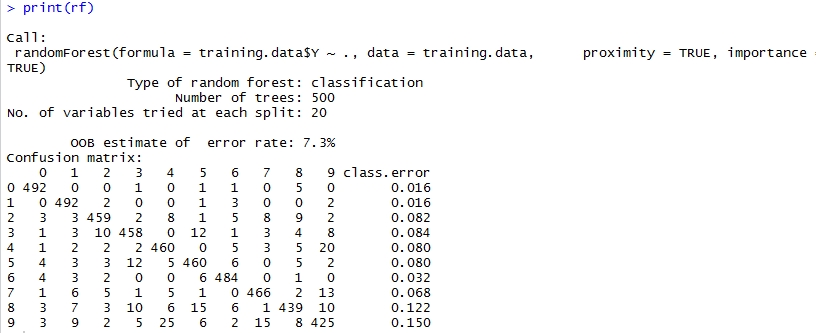


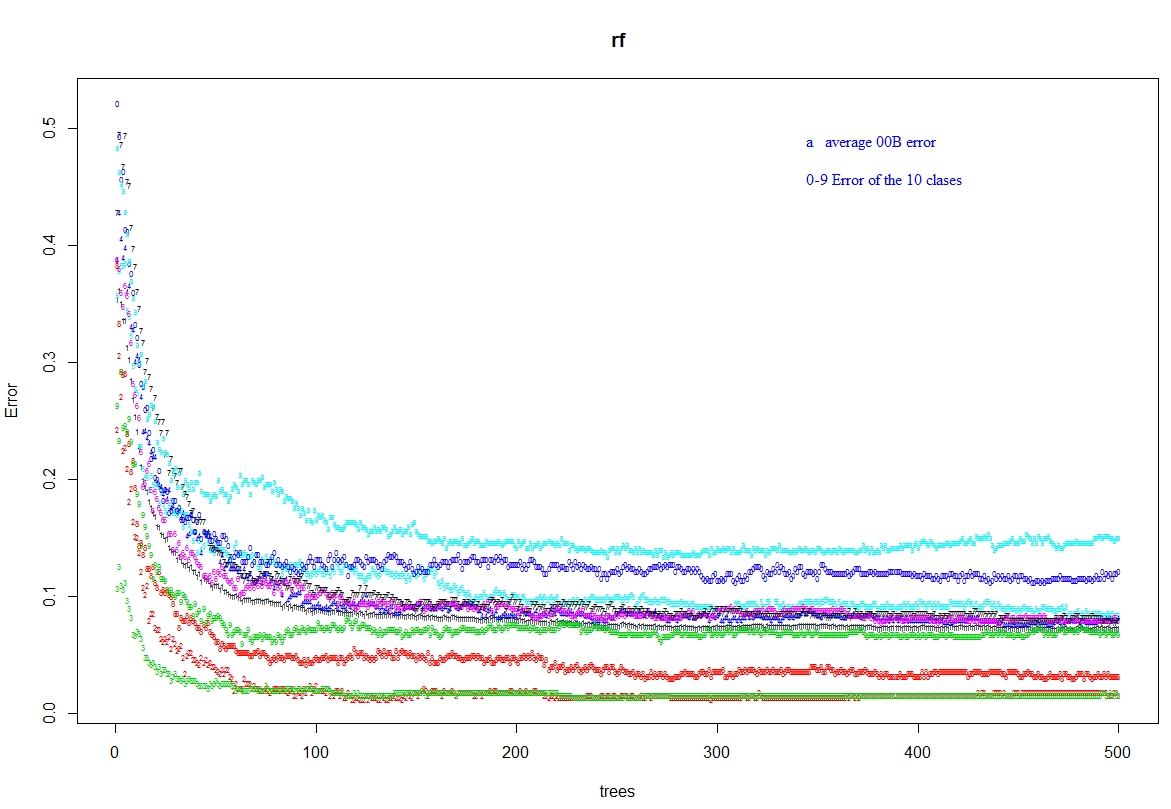
We use the following code to train the whole training data:

rf=randomForest(training.data$Y~.,training.data,proximity = TRUE,importance=TRUE)

plot(rf,type="p",cex=0.5)

print(rf)

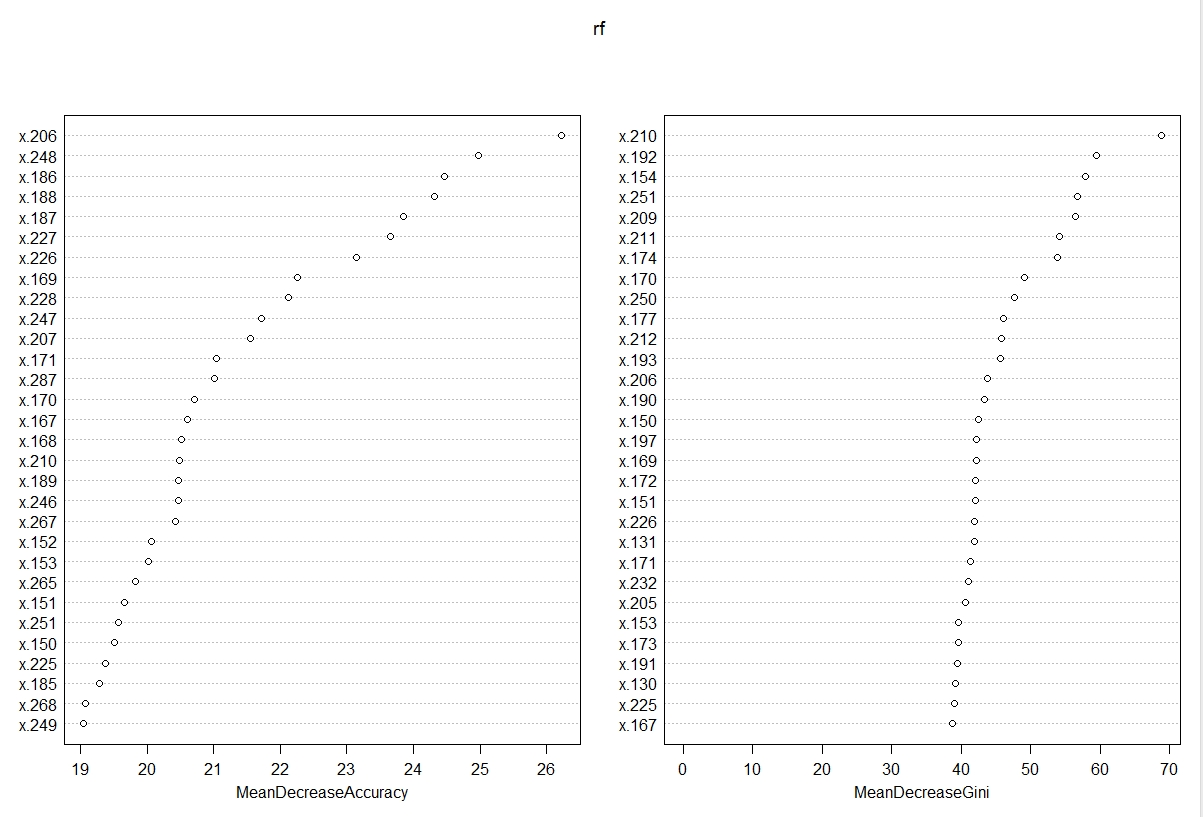




The importance of the variable plot :

imp=importance(rf)

varImpPlot(rf)



## Find Optimal Parameter

### The Right Way

**Tune randomForest for the optimal mtry parameter**

**Description**: Starting with the default value of mtry, search for the optimal value (with respect to Out-of-Bag error estimate) of mtry for randomForest.

**Usage:** tuneRF(x, y, mtryStart, ntreeTry=50, stepFactor=2, improve=0.05,

trace=TRUE, plot=TRUE, doBest=FALSE, ...)

**Arguments**

|  |  |
| --- | --- |
| x | matrix or data frame of predictor variables |
| y | response vector (factor for classification, numeric for regression) |
| mtryStart | starting value of mtry; default is the same as in [randomForest](http://127.0.0.1:46782/help/library/randomForest/help/randomForest) |
| ntreeTry | number of trees used at the tuning step |
| stepFactor | at each iteration, mtry is inflated (or deflated) by this value |
| improve | the (relative) improvement in OOB error must be by this much for the search to continue |
| trace | whether to print the progress of the search |
| plot | whether to plot the OOB error as function of mtry |
| doBest | whether to run a forest using the optimal mtry found |
| ... | options to be given to [randomForest](http://127.0.0.1:46782/help/library/randomForest/help/randomForest) |

> trf=tuneRF(training.data[,-1],training.data$Y,ntreeTry=500,n.var=400)

mtry = 20 OOB error = 7.36%

Searching left ...

mtry = 10 OOB error = 7.78%

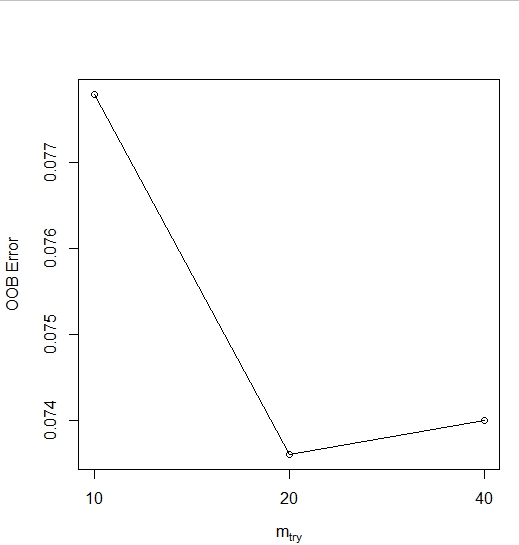
-0.05706522 0.05

Searching right ...

mtry = 40 OOB error = 7.4%

-0.005434783 0.05

**The optimal mtry is 20 which is same with the default mtry**

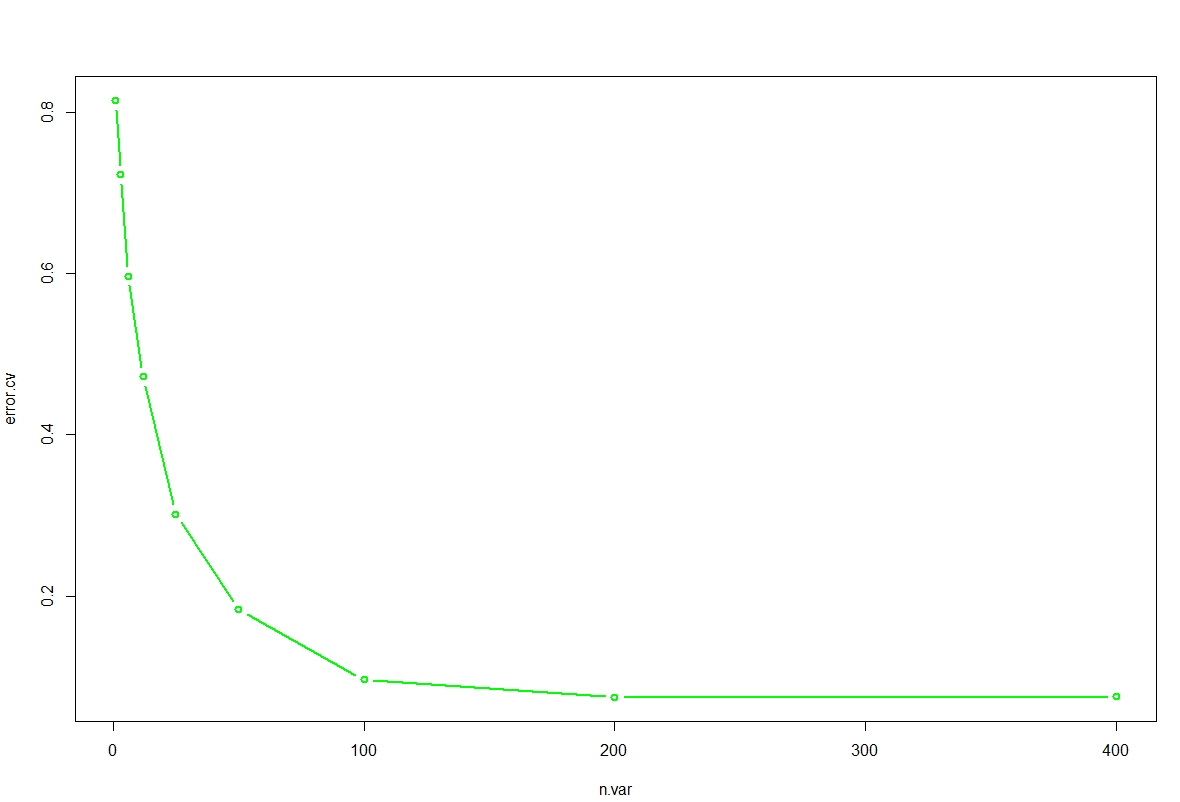


### The Wrong Way

Note that we cannot use the following code to find the optimal mtry. In the following process, they select the features according to the importance of the variables instead of randomly.

cv=rfcv(training.data[,-1],training.data$Y,cv.fold=5)

with(cv, plot(n.var, error.cv, log="x", type="b", lwd=2),col="green")



## Test on the Test Data

start\_time=proc.time()

pre\_test=predict(rf,test.data[,-1])

end\_time=proc.time()

run\_time=start\_time-end\_time

error\_rate=sum(pre\_test!=test.label)/nrow(test.data)

> run\_time

user system elapsed

1.61 0.08 1.78

> error\_rate

[1] 0.0655

summary of RF:

Random Forests algorithm is a classifier based on primarily two methods - bagging and random subspace method.

Suppose we decide to have S number of trees in our forest then we first create S datasets of "same size as original" created from random resampling of data in T with-replacement (n times for each dataset). This will result in {T1, T2, ... TS} datasets. Each of these is called a bootstrap dataset. Due to "with-replacement" every dataset Ti can have duplicate data records and Ti can be missing several data records from original datasets. This is called Bagging.

Now, RF creates S trees and uses m (=sqrt(M) or =floor(lnM+1)) random sub features out of M possible features to create any tree. This is called random subspace method.

So for each Ti bootstrap dataset you create a tree Ki. If you want to classify some input data D = {x1, x2, ..., xM} you let it pass through each tree and produce S outputs (one for each tree) which can be denoted by Y = {y1, y2, ..., ys}. Final prediction is a majority vote on this set.

Out-of-bag error:

After creating the classifiers (S trees), for each (xi,yi) in the original training set i.e. T, select all Tk which does not include (xi,yi). This subset, pay attention, is a set of boostrap datasets which does not contain a particular record from the original dataset. This set is called out-of-bag examples. There are n such subsets (one for each data record in original dataset T). OOB classifier is the aggregation of votes ONLY over Tk such that it does not contain (xi,yi).

## Code Appendix:

# classification of handwritten digits(Random Forest)

#Process the data

library("randomForest", lib.loc="~/R/win-library/3.2")

data=load("C:/Users/Christina/Desktop/digitsdata.RData")

image(t(1 - training.data[3,1,,])[,20:1],col=gray(seq(0, 1, length.out=256)),axes=FALSE, asp=1)

num.class <- dim(training.data)[1] # Number of classes

num.training <- dim(training.data)[2] # Number of training data per class

d <- prod(dim(training.data)[3:4]) # Dimension of each training image (rowsxcolumns)

num.test <- dim(test.data)[2] # Number of test data

dim(training.data) <- c(num.class \* num.training, d) # Reshape training data to 2-dim matrix

dim(test.data) <- c(num.class \* num.test, d) # Same for test.

training.label <- rep(0:9, num.training) # Labels of training data.

test.label <- rep(0:9, num.test) # Labels of test data

#transfer the data matix into data frame

training.data=as.data.frame(cbind(training.label,training.data))

training.data[,1]=as.factor(training.data[,1])

test.data=as.data.frame(cbind(test.label,test.data))

test.data[,1]=as.factor(test.data[,1])

colnames(training.data)=c("Y",paste("x.",1:400,sep=""))

colnames(test.data)=c("Y",paste("x.",1:400,sep=""))

# train the whole training data with random forest

rf=randomForest(training.data$Y~.,training.data,proximity = TRUE,importance=TRUE)

plot(rf,type="p",cex=0.5)

print(rf)

imp=importance(rf)

varImpPlot(rf)

#find the optimal mtry paraeter

trf=tuneRF(training.data[,-1],training.data$Y,ntreeTry=500)

#note we can not use the following code to find the optimal mtry

cv=rfcv(training.data[,-1],training.data$Y,cv.fold=5)

with(cv, plot(n.var, error.cv, log="x", type="b", lwd=2),col="green")

# Test on the test data

start\_time=proc.time()

pre\_test=predict(rf,test.data[,-1])

end\_time=proc.time()

run\_time=start\_time-end\_time

error\_rate=sum(pre\_test!=test.label)/nrow(test.data)

> run\_time

user system elapsed

1.61 0.08 1.78

> error\_rate

[1] 0.0655