Randon Forest with Grid Search Tuning

Chris Schmidt

11/18/2019

Random Forest with Grid Search Tuning of Hyperparameters

Random forests are a type of decision tree model process where multiple decorrelated trees are built on bootstrapped training samples.

Tree based methods can be used for regression and for classification modeling. They stratify or segment the predictor space into a number of simple regions.

With regression trees we take the response variable and make a split in the data for the predictor that has the most influence on the response based on some rule, perhaps the mean. Each region above and below the split point are then evaluated for the next most important predictor and a similar split occurs. This repeats a specified number of times per the rules that are assigned.

There are essentially two steps: 1. Divide the predictor space, the set of possible values for $X_1, X_2, ..., X_p$ into J distinct and non-overlapping regions, $R_1, R_2, ..., R_j$. 2. For every observation that falls in region R_j the same prediction is made using the mean of the response values for training observations in R_j.

The predictor space is divided into high-dimension rectangles, or *hyper-boxes*. The goal is to find boxes $R_1, R_2, ..., R_j$ that minimize the residual sum of squares, RSS, given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2,$$

where $\hat{y}_{R_j}$ is the mean response for the training observations in the jth *hyper-box*. Because this is very computationally expensive tp consider all possible partitions, a *top-down, greedy* approach called *bianry splitting* is used. The approach begins at the top of the tree where all observations are in the same region and makes the best split at each step which is the reason for the term *greedy*.

For recursive binary splitting, select predictor X_j and a specified cutpoint s such that splitting the regions of predictor space in which X_j takes on a value in relation to s represented $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ which leads to the largest reduction in RSS.

This defines a pair of half planes

$$R_1(j, s) = \{X|X_j < s\} \text{ and } R_2(j, s) = \{X|X_j \geq s\}$$

and we want to minimize the equation

$$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:x_i \in R_s(j,s)} (y_i - \hat{y}_{R_s})^2$$

for a value of j and s that we need to find. Thge process is repeated in each region formed until a stopping criterion is reached.

Random forests build a number of decision trees but when the trees are built each time a split is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors. The split can only use one of the m predictors. Typically $m \approx \sqrt{p}$ is the number of predictors considered at each split.

Use grid search to tune a random forest model for the Spam7 data in library(DAAG).

1. Separate the data into a training set (70%) and a testing set (30%). Tune the model by the grid search method and report the testing error.
2. Produce a plot to compare the different settings of the hyper-parameters as the graph on the last page of chapter-8-tuning-of-bagging-and-random-forest.pdf, or chapter-8-Gradient-Boosting-Machine-classification.pdf.

Load the Packages and Libraries needed. We need the ISLR, rpart, and rpart.plot. If you cannot install rpart.plot inside Rstudio, you can download the release file and install if from R console.

Create the testing and training sets for this project.

```
library(DAAG)
```

For the spam7 data set from the DAAG library, use set.seed() and build a randomized training set with 70% of the data and a test set with the remaining 30%.

```
## Loading required package: lattice
```

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
library(ggplot2)
set.seed(123)
data(spam7)
dataset <- spam7
inTraining <- createDataPartition(dataset$yesno, p = .7, list = FALSE)
training <- dataset[ inTraining,]
testing  <- dataset[-inTraining,]
```

III. Create the random forest model with tuning parameters

A. Run model on training set with set parameters

```
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```
##
## Attaching package: 'randomForest'
```

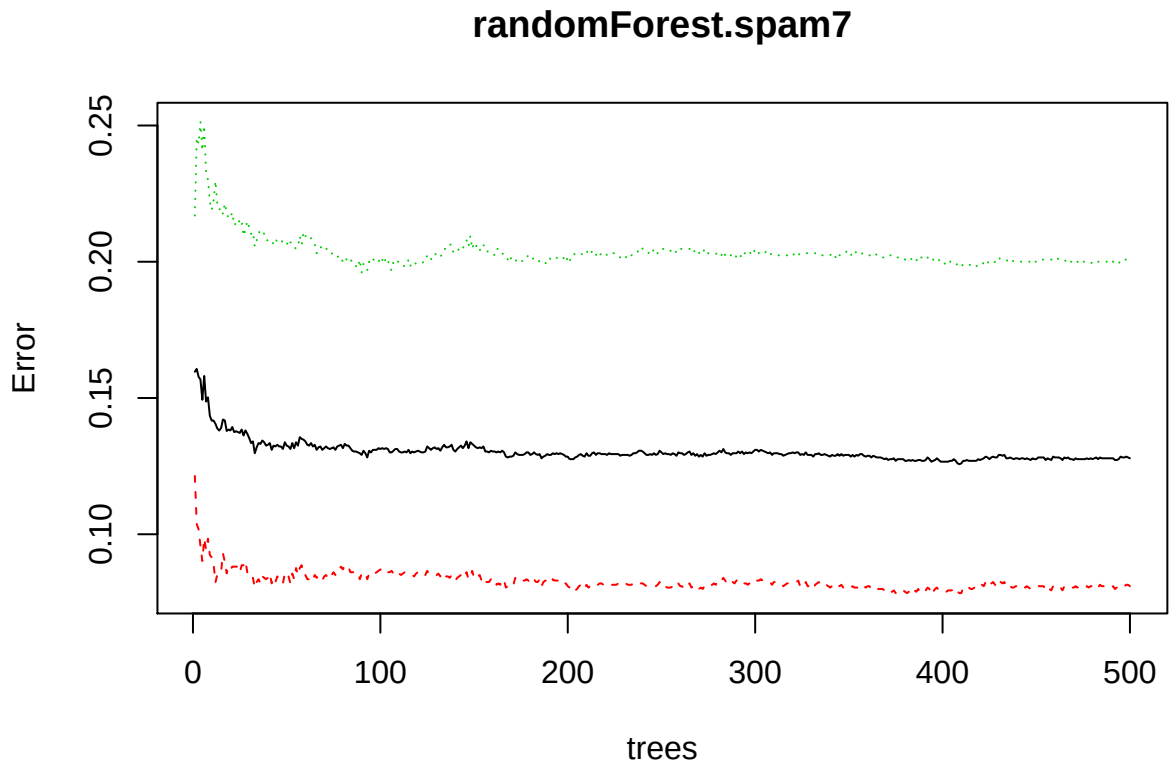
```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
randomForest.spam7 = randomForest(yesno ~.,
                            data = training,
                            mtry = 5,
                            ntree = 500)
print(randomForest.spam7)
```

```
##
## Call:
```

```
##  randomForest(formula = yesno ~ ., data = training, mtry = 5,      ntree = 500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##         OOB estimate of  error rate: 12.79%
## Confusion matrix:
##      n    y class.error
## n 1794  158  0.08094262
## y  254 1016  0.20000000
```

```
plot(randomForest.spam7)
```



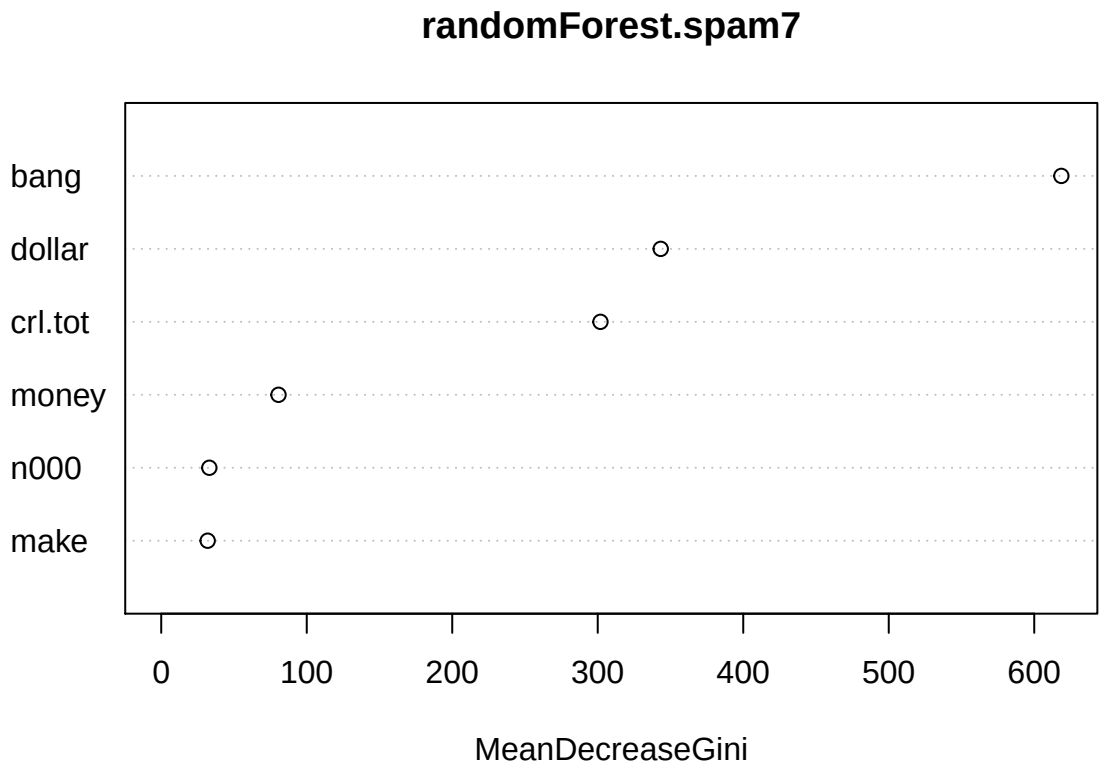
******Notice OOB estimate of error rate on training set in output above.**

B. Look at the Variable importance. Use the importance() and varImpPlot() functions to evaluate the MeanDecreaseGini values

```
importance(randomForest.spam7)
```

```
##         MeanDecreaseGini
## crl.tot        301.87254
## dollar         343.27673
## bang           618.63085
## money           80.59186
## n000            33.05850
## make            31.91448
```

```
varImpPlot(randomForest.spam7)
```



C. Make predictions with testing set and show the error.

```
Pred.randomForest.spam7 = predict( randomForest.spam7, testing)
mean((Pred.randomForest.spam7!=testing$yesno)^2)
```

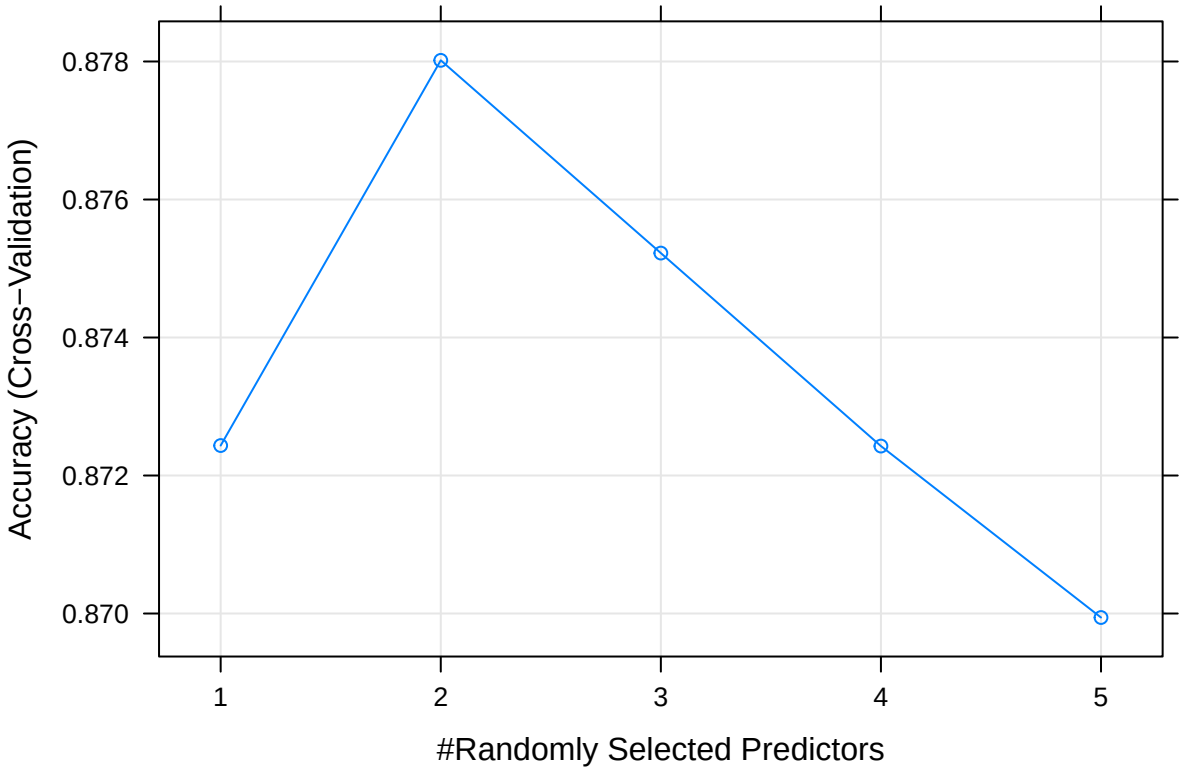
```
## [1] 0.1131255
```

******Note the error above of 11.24% for the prediction using the test set and the random forest model.**

IV. Use grid search to tune the model.

```
tune.grid <- expand.grid(.mtry = c(1:5))
rand.forestTrain <- train(yesno ~.,
                          data = training,
                          method = "rf",
                          metric = "Accuracy",
                          tuneGrid = tune.grid,
                          trControl = trainControl(method = "cv",
                                                   number = 10)
                          )

plot(rand.forestTrain)
```




```
print(rand.forestTrain)
```

```
## Random Forest
##
## 3222 samples
##    6 predictor
##    2 classes: 'n', 'y'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2900, 2900, 2900, 2900, 2900, 2899, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   1     0.8724343  0.7245855
##   2     0.8780167  0.7387083
##   3     0.8752235  0.7338703
##   4     0.8724266  0.7289880
##   5     0.8699421  0.7244573
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

****To find the error subtract the largest value in the Accuracy column above from one. I.e. (1 - 0.8764687) = 0.1235313 or 12.35% (rounded).**

A. Expand upon the hyperparameter grid search model.

```
optimal_trees = list()

hyperparameter.grid <- expand.grid(
maxnodes = c(5, 9, 50, nrow(training)),
ntree = c(100, 250, 500)
)

nr = nrow(hyperparameter.grid)
tuneGrid <- expand.grid(.mtry = c(2, 5)) # best value from last step is 5

for (i in 1:nrow(hyperparameter.grid)) {

    set.seed(999)
    rf_maxtrees <- train(yesno ~.,
                    data = training,
                    method = "rf",
                    metric = "Accuracy",
                    tuneGrid = tuneGrid,
                    trControl = trainControl(method = "cv",
                                             number = 10),
                    importance = TRUE,
                    nodesize = 5,
                    maxnodes = hyperparameter.grid[i,1],
                    ntree = hyperparameter.grid[i,2])
    key <- paste(hyperparameter.grid[i,1], hyperparameter.grid[i,2], sep = '-')
    optimal_trees[[key]] <- rf_maxtrees
}
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
```

```
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
```

```
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
```

```
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): maxnodes exceeds
## its max value.
```

```
results_mtry <- resamples(optimal_trees)
summary(results_mtry)
```

```
##
## Call:
```

```
## summary.resamples(object = results_mtry)
##
## Models: 5-100, 9-100, 50-100, 3222-100, 5-250, 9-250, 50-250, 3222-250, 5-500, 9-500, 50-500, 3222-5
## Number of resamples: 10
##
## Accuracy
##               Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 5-100    0.8260870 0.8423913 0.8571429 0.8519480 0.8626858 0.8668731    0
## 9-100    0.8447205 0.8493789 0.8635656 0.8612638 0.8688922 0.8788820    0
## 50-100   0.8602484 0.8672360 0.8728679 0.8739948 0.8812112 0.8881988    0
## 3222-100 0.8571429 0.8696662 0.8726708 0.8739900 0.8814852 0.8944099    0
## 5-250    0.8229814 0.8400621 0.8509317 0.8500875 0.8599071 0.8881988    0
## 9-250    0.8447205 0.8580274 0.8664596 0.8643703 0.8688922 0.8850932    0
## 50-250   0.8602484 0.8683225 0.8728679 0.8724410 0.8781056 0.8819876    0
## 3222-250 0.8571429 0.8675461 0.8757764 0.8743034 0.8782017 0.9006211    0
## 5-500    0.8260870 0.8400621 0.8524845 0.8519489 0.8630031 0.8819876    0
## 9-500    0.8416149 0.8580274 0.8651136 0.8649914 0.8726708 0.8819876    0
## 50-500   0.8602484 0.8706445 0.8744207 0.8743034 0.8804348 0.8881988    0
## 3222-500 0.8602484 0.8696662 0.8757764 0.8755437 0.8791632 0.8944099    0
##
## Kappa
##               Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## 5-100    0.6408684 0.6713427 0.6987742 0.6892048 0.7122253 0.7223581    0
## 9-100    0.6629119 0.6755575 0.7073704 0.7014300 0.7180110 0.7418493    0
## 50-100   0.6996144 0.7127144 0.7303703 0.7302627 0.7453050 0.7620397    0
## 3222-100 0.6907595 0.7223137 0.7304939 0.7306391 0.7474088 0.7727216    0
## 5-250    0.6319631 0.6678980 0.6892424 0.6861692 0.7047893 0.7633610    0
## 9-250    0.6600076 0.6954886 0.7117453 0.7082558 0.7176102 0.7557706    0
## 50-250   0.6979110 0.7195035 0.7294702 0.7268129 0.7408435 0.7481164    0
## 3222-250 0.6907595 0.7177305 0.7354678 0.7309277 0.7402931 0.7866932    0
## 5-500    0.6408684 0.6678980 0.6922757 0.6902470 0.7097891 0.7502143    0
## 9-500    0.6586005 0.6954886 0.7108689 0.7099001 0.7253470 0.7495189    0
## 50-500   0.6979110 0.7233135 0.7319992 0.7306235 0.7448022 0.7627021    0
## 3222-500 0.6979110 0.7191557 0.7341129 0.7335072 0.7444708 0.7727216    0
```

```
# modified code from the following website: https://www.guru99.com/r-random-forest-tutorial.html
```

******The best model in the output above for Accuracy shows an error of (1 - 0.9006211) = 0.0993789 or 9.94% (rounded).**

```
randomForest.spam7 <- train(yesno ~.,
                    data = training,
                    method = "rf",
                    metric = "Accuracy",
                    tuneGrid = tuneGrid,
                    trControl = trainControl(method = "cv",
                                             number = 10),
                    importance = TRUE
                    )
print(randomForest.spam7)
```

```
## Random Forest
##
## 3222 samples
```

```
##    6 predictor
##    2 classes: 'n', 'y'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2900, 2899, 2899, 2900, 2900, 2900, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.8736794  0.7295924
##   5     0.8671596  0.7176562
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

B. Store the Accuracy in a data.frame()

```
seq(from = 3, to = 25, length.out = 25)
```

```
##  [1]  3.000000  3.916667  4.833333  5.750000  6.666667  7.583333  8.500000
##  [8]  9.416667 10.333333 11.250000 12.166667 13.083333 14.000000 14.916667
## [15] 15.833333 16.750000 17.666667 18.583333 19.500000 20.416667 21.333333
## [22] 22.250000 23.166667 24.083333 25.000000
```

```
mtx_cv = results_mtry$values[, seq(from = 3, to = 25, length.out = 25)]

colnames(mtx_cv) = c(paste(hyperparameter.grid[,1], hyperparameter.grid[,2], sep = '-'), 'default')
```

V. Use a box-plot to display the results of the different tuning parameters.

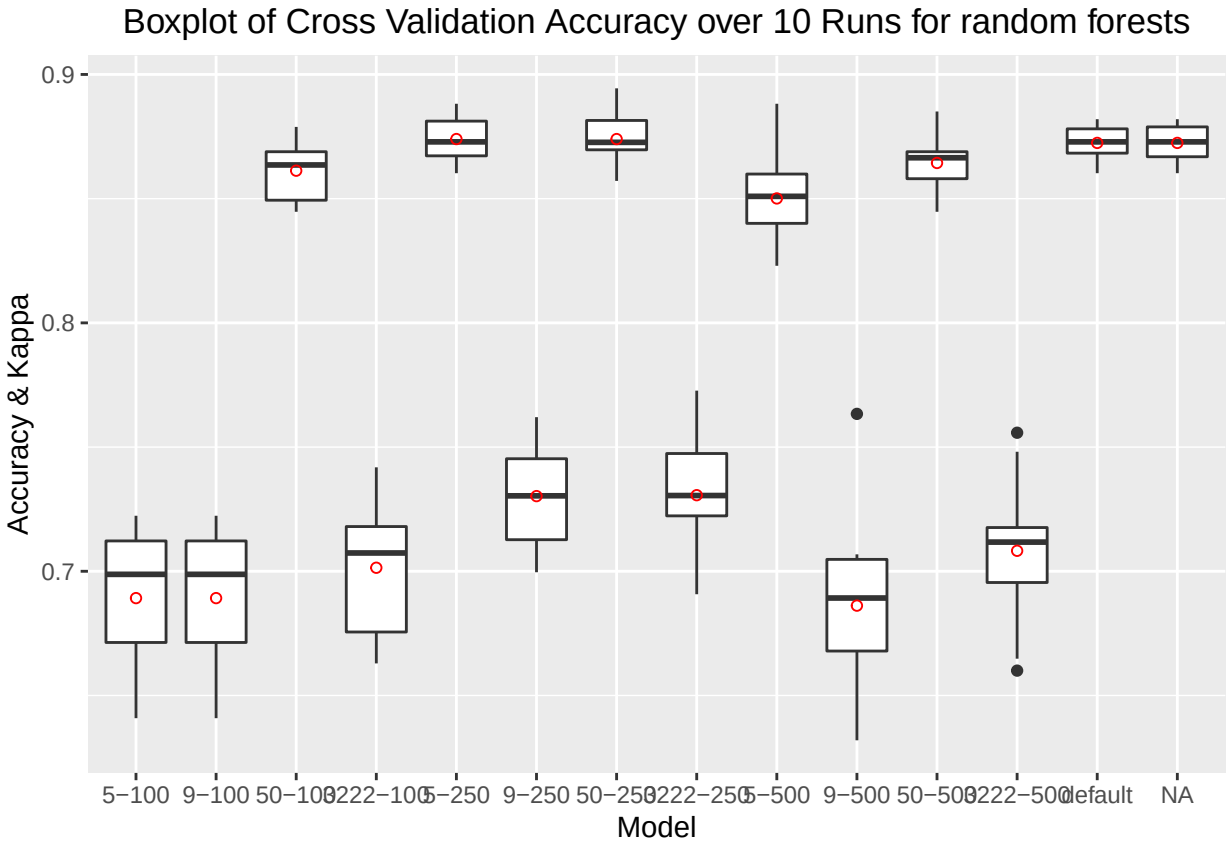
```
library(ggplot2)
library(reshape2)

M2 <- melt(mtx_cv, measure.vars = colnames(mtx_cv))

M2 <- melt(mtx_cv, measure.vars = colnames(mtx_cv))

ggplot(M2,aes(x = variable, y = value)) +
  geom_boxplot() + xlab("Model") +
   ylab("Accuracy & Kappa") +
     stat_summary(fun.y = mean,shape = 1,col = 'red',geom = 'point')+
       ggtitle("Boxplot of Cross Validation Accuracy over 10 Runs for random forests")+
         theme(plot.title = element_text(hjust = 0.5))
```

```
## Warning: `fun.y` is deprecated. Use `fun` instead.
```


Boxplot of Cross Validation Accuracy over 10 Runs for random forests
Accuracy & Kappa
0.9
0.8
0.7
5−100
9−100
50−100
3222−100
5−250
9−250
50−250
3222−250
5−500
9−500
50−500
3222−500
default
NA
Model