```
#########################################################################
#           Functions .R
#########################################################################
#
#
#
#     ###     Description         ###
#
# Functions contains various generally aplicable methods/functions
# that I desire to have scripted and available in a tested
# and reliable manner.  Anything that can be refactored should
# eventually find it's way here.
#
#     ###     By              ###
#
# Written by Christopher Carlson
#
#     ###     For             ###
#
# Written initially for Western Michigan University's Summer 1 2015
#     Semester course, CS 5950 - Machine Learning.
#
#     ###     Date            ###
#
# Monday June 29, 2015
#
#     INDEX
#
# 0. Preload Necessary Libraries
#
# 1. "Round-Mean" - round_mean(X, digits)
#
# 2. "Ceiling-Mean" - ceil_mean(X)
#
# 3. "Floor-Mean" - floor(X)
#
# 4. "Print-Summary" - print_summary(X)
#
# 5. "Subset-Folds" - subset_flds(folds, i_test)
#
# 6. "Grow Forest" - grow_forest(data_frame, test_data)
#
# 7. "Grow Tree" - grow_tree(data_frame)
#
# 8. "Test Tree" - test_tree(model, test_data)
#
# 9. "Generate Tree-Fit Confusion Matrix" - gen_tree_conf_mat(pred, test_data)
```

```
#
#  10. "Plot Tree" - plot_tree(tree_model, main_message)
#
#  11. "Get Random Forest Confusion Matrix" - get_rf_conf_mat(rf_object)
#
#  12. "Apply K-Folds to Tree" - k_folds(tree)
#
#  13. "Write Confusion Matrix" - write_confusion_matrix(c_matrix, r_file)
#
#  14. "Write Message" - write_message(msg, r_file)
#
#  15. "Load Data" - load_data(data_file)
#
#########                              ##########

# 0

## Load Libraries
library('randomForest')
library(rpart)
library(plyr)
##

# 1

##  "Round-Mean"
## Find the mean of some collection X and then round the result to
## digits places.  Used with apply functions.
round_mean <- function(X, digits)
{
  round(data.frame(mean(X)), digits)
}

# 2

## "Ceiling-Mean"
## Find the mean of collection X and then round the result up to the
## nearest integer value.
ceil_mean <- function(X)
{
  ceiling(mean(X))
}

#3

## "Floor-Mean"
## Find the mean of collection X and then round the result down to the
```

```r
## nearest integer value.
floor_mean <- function(X)
{
  floor(mean(X))
}

# 4

## "Print Summary"
## Print the summary of item X.
print_summary <- function(X)
{
  print(summary(X))
}

# 5

## "Subset Folds"
## Subsets K folds into Training and Testing Data where the i'th fold
## becomes test and the other four folds become training folds.
subset_flds <- function(folds, i_test)
{
  env <- parent.frame()
  env$test  <- folds[[i_test]]
  env$train <- folds[c(seq(1:(length(folds))))[-i_test]]
}

# 6

## "Grow Forest
## Function to generate an RF model given a data frame object from the
## mushrooms data set, and some test data.
grow_forest <- function(data_frame, test_data) {

# Random Forest Object is generated by the 'randomForest' command.
# mtry is the number of variables tested for each split, ntree is the
# number of trees grown, xtest and ytest specify respectively the
# set of predictors to test the data on, and the set of correct responces
# corresponding with the predictors.
  randomForest(edibility~., data=data_frame, mtry=3, ntree=150,
      xtest=test_data[-1], ytest=test_data$edibility,
      importance=TRUE, proximity=TRUE)
}

# 7

# "Grow Tree"
```

```
# Function to generate a model given a data frame object from the
# mushrooms data set.
grow_tree <- function(data_frame) {
  rpart(edibility~., data=data_frame, method="class")
}

# 8

# "Test Tree"
# A function to classify some data using a tree model produced by
# the grow_tree, (rpart), function.   It does not return typical
# results, instead it simply returns an array of P's and E's
# indicating which of the two values the probilities indicated.
test_tree <- function(model, test_data) {
  pred <- data.frame(predict(model, test_data, type='prob'))

  results <- array("p", nrow(test_data))

  for(i in 1:nrow(results))
  {
    if(pred[i, ]$e > pred[i, ]$p)
    {
      results[i] = "e"
    }
  }
  results
}

# 9

# "Generate Tree-Fit Confusion Matrix"
# A function to compare the predictions with the data labeles
# to construct a confusion matrix based of the tree clasifiers
# results.
gen_tree_conf_mat <- function(pred, test_data) {

  pp = 0 # counter for poisonous mushrooms predicted as poisonous
  pe = 0 # counter for poisonous mushrooms predicted as edible
  ep = 0 # counter for edible mushrooms predicted as poisonous
  ee = 0 # counter for edible mushrooms predicted as edible

  for(i in 1:nrow(pred))
  {
    if(pred[i] == test_data$edibility[i])
    {
      if(pred[i] == 'e')
      {
```

```
      ee = ee+1
    }
    else
    {
      pp = pp+1
    }
  }
  else
  {
    if(pred[i] == 'e')
    {
      pe = pe+1
    }
    else
    {
      ep = ep+1
    }
  }
}
conf_matrix <- matrix(c(ee, ep, pe, pp), nrow=2, ncol=2)
rownames(conf_matrix)<-c("e", "p")
colnames(conf_matrix)<-c("e", "p")
conf_matrix
}

# 10

## "Plot Tree"
## Given a tree model, plot the tree model.
plot_tree <- function(tree_model, main_label="Classification Tree for Mushroom Ed
{
par(mfrow=c(1,2), xpd=NA)
plot(tree_model, uniform=TRUE, main=main_label)
text(tree_model, use.n=TRUE, all=TRUE, cex=.8)
post(tree_model, file=paste0("results_temp/fit_", i), title="Classification Tre
}

# 11

## "Apply K-Folds to Tree"
## Applies K-Folds using classification trees.  Returns a list
## of confusion matrices.
kfolds_tree <- function(train, test)
{
  # First get the parent environment to ease command line use.
  env <- parent.frame()
  # Now generate the models for each of the training data lists.
```

```
  env$fits  <- lapply(train, FUN=grow_tree)

  # Next generate predictions from each model using the test data.
  env$preds <- lapply(fits, FUN=test_tree, test_data=test)

  # Generate a list of confusion matrices.
  env$conf_mats <- lapply(preds, FUN=gen_tree_conf_mat, test_data=test)
}

# 12

## "Prune Tree"
## Given a tree model, prune the tree.
prune_tree <- function(fit)
{
  # The prune cp parameter or "Complexity Parameter" is the measure
  # to use to prune on.  Here we decide which to use based on which
  # has the smallest cross-validation error.
  prune(fit, cp=fit$cptable[which.min(fit$cptable[,"xerror"]), "CP"])
}

# 13

## "Get Random Forest Confusion Matrix"
## A function to extract the confusion matrices from the random
## forest object returned by randomforest.
get_rf_conf_mat <- function(rf_object)
{
  conf<-as.array(rf_object$confusion)
  p<-conf[,c(1,2)]
  conf_mat<-matrix(c(p[1],p[2], p[3], p[4]), nrow=2, ncol=2)
  conf_mat
}

# 14

## "Write Confusion matrix"
## A function to write a confusion matrix to the results file.
write_confusion_matrix <- function(c_matrix, r_file)
{
  write(c_matrix, file=r_file, ncolumns=2, append=TRUE)
  write("\n", file=r_file, ncolumns=1, append=(TRUE))
}

#15

## "Write Message"
```

```
## Write message to file.
write_message <- function(msg, r_file)
{
  write(msg, file=r_file, append=TRUE)
}

#16

## "Load Data"
## Runs data_setup.R
load_data <- function(data_file)
{
if(is.null(data_file))
{
  data_file<-'../Data/agaricus-lepiota.data'
}

## Get the parent environment
env <- parent.frame()
## Load Data
env$mushrooms=read.csv(data_file, header=TRUE, sep=",")

## Specify Some Variables
n_folds <- 5
index_folds <- list()
env$folds <- list()

n_entries_per_fold <- floor(nrow(env$mushrooms)/(n_folds))

## Generate the indices we will use to segment the data.
all_indices <- seq_len(nrow(mushrooms))
while( length(all_indices) > n_entries_per_fold)
{
  temp <- sample(all_indices, size = n_entries_per_fold)
  all_indices <- setdiff(all_indices, temp)
  index_folds <- c(index_folds, list(temp))
}

## At this point there are a few indices that were not used.  These
## indices are added to the index_folds vectors starting at vector 1.
i = 1
while( length(all_indices) > 0 )
{
  index_folds[[i]] <- append( index_folds[[i]], all_indices[1])
  all_indices <- setdiff(all_indices, all_indices[1])
  i = i+1
  if( i > n_folds ) i = 1
```

```
}

## Now we have a list of vectors such that all the vectors contain
## all of indices of the data set, all the vectors are
## mutually disjoint (no repeates among them), and all of them are
## randomly selected.  Now, using these sets of indices, we will
## subset the data into ten subsets.
  env$folds <- list(data.frame(mushrooms[index_folds[[1]], ]), data.frame(mushroo
                data.frame(mushrooms[index_folds[[3]], ]), data.frame(mushrooms[i
                data.frame(mushrooms[index_folds[[5]], ]))
                #, data.frame(mushrooms[index_folds[[6]], ]),
                #data.frame(mushrooms[index_folds[[7]], ]), data.frame(mushrooms[
                #data.frame(mushrooms[index_folds[[9]], ]), data.frame(mushrooms[


# Now the i'th fold can be accessed as a list item by: folds[[i]]
# categories can be accessed by: folds[[i]]$category_name
}



source('Functions.R')
#############################################################################
#           data_setup.R
#############################################################################
#
#
#
#     ###     Description         ###
#
#   Partitions the data into k-index_folds which can be used
# with any of the various models we might want to try out with the
#   data.  The goal of this is to be able to run this, and to initiate
#   the data into the R-workspace so models can be trained and tested
#   using the data.
#
#     ###     By              ###
#
# Written by Christopher Carlson
#
#     ###     For             ###
#
# Written initially for Western Michigan University's Summer 1 2015
#     Semester course, CS 5950 - Machine Learning.
#
#########                           ##########
```

```
# Load Data
mushrooms=read.csv("../Data/agaricus-lepiota.data", header=TRUE, sep=",")


# Specify Some Variables
n_folds <- 3
index_folds <- list()
folds <- list()

n_entries_per_fold <- floor(nrow(mushrooms)/(n_folds))

# Generate the indices we will use to segment the data.
all_indices <- seq_len(nrow(mushrooms))
while( length(all_indices) > n_entries_per_fold)
{
  temp <- sample(all_indices, size = n_entries_per_fold)
  all_indices <- setdiff(all_indices, temp)
  index_folds <- c(index_folds, list(temp))
}

  ## At this point there are a few indices that were not used.  These
  ## indices are added to the index_folds vectors starting at vector 1.
i = 1
while( length(all_indices) > 0 )
{
  index_folds[[i]] <- append( index_folds[[i]], all_indices[1])
  all_indices <- setdiff(all_indices, all_indices[1])
  i = i+1
  if( i > 10 ) i = 1
}

  ## Now we have a list of vectors such that all the vectors contain
  ## all of indices of the data set, all the vectors are
  ## mutually disjoint (no repeates among them), and all of them are
  ## randomly selected.  Now, using these sets of indices, we will
  ## subset the data into ten subsets.

  folds <- list(data.frame(mushrooms[index_folds[[1]], ]), data.frame(mushrooms[i
                data.frame(mushrooms[index_folds[[3]], ]))#
                # , data.frame(mushrooms[index_folds[[4]], ]),
                # data.frame(mushrooms[index_folds[[5]], ]))
                # , data.frame(mushrooms[index_folds[[6]], ]),
                # data.frame(mushrooms[index_folds[[7]], ]), data.frame(mushrooms
                # data.frame(mushrooms[index_folds[[9]], ]), data.frame(mushrooms
```

```r
# Now the i'th fold can be accessed as a list item by: folds[[i]]
# categories can be accessed by: folds[[i]]$category_name



source('Functions.R')
#########################################################################
#           classification_tree.R
#########################################################################
#
#
#
#     ###     Description         ###
# Runs the classification tree algorithm k-folds time and performs
# k-folds cross validation on the results.  Writes the results to
# the file "classification_tree_cv_results.txt" as confusion matrices.
#
#     ###     By              ###
#
# Written by Christopher Carlson
#
#     ###     For             ###
#
# Written initially for Western Michigan University's Summer 1 2015
#   Semester course, CS 5950 - Machine Learning.
#
#########                               ##########


# This matrix will hold the final results after running the complete
# cross validation.
confusion_matrix_averages <- vector("list", (length(folds)-1))

results_file <- "results/tree_classification.txt"

# In this section of the code, the goal is to cross-validate over each
# of the folds.  So for i in nFolds, it will make the i'th fold the
# testing data, and it will build a tree from each of the remaining
# folds.  Then it will test the fit of each tree on the test fold.
#
# The results of each test are added to a list, and finally at the
# end all the average test perfomance is calculated and reported.
for( i in 1:(length(folds)))
{
  # Name i'th fold 'test' and add the
```

```
  # remaining folds to a list called 'train'
  test  <- folds[[i]]
  train <- folds[c(seq(1:(length(folds))))[-i]]

  # First generate the models for each of the training data lists.
  fits  <- lapply(train, FUN=grow_tree)

  # Next generate predictions from each model using the test data.
  preds <- lapply(fits, FUN=test_tree, test_data=test)

  # Generate a list of confusion matrices.
  conf_mats <- lapply(preds, FUN=gen_tree_conf_mat, test_data=test)

  # Write results to file.
  write_message(paste("Iteration #", i, ":\n", sep=""), results_file)
  lapply(conf_mats, FUN=write_confusion_matrix, r_file=results_file)

  # Add the average confusion matrix for this iteration to the
  # confusion_matrix_averages list.
  confusion_matrix_averages[[i]]<- apply(simplify2array(conf_mats),
      c(1,2), mean)
}
# Write the results averages to the results file.
write_message("Averages of each CV iteration:\n", results_file)
lapply(confusion_matrix_averages, FUN=write_confusion_matrix, r_file=results_file


# Finally, generate a single confusion matrix from all the average
# confusion matrices and write it to results.
final_confusion_matrix <- apply(simplify2array(confusion_matrix_averages),
              c(1,2), mean)
write_message(paste0("Overall Test Error for ", length(folds),
          "-folds cross validation:\n"), results_file)
write_confusion_matrix(final_confusion_matrix, results_file)



source('Functions.R')
#############################################################################
#           random_forest.R
#############################################################################
#
#
#
#     ###     Description         ###
# Runs k-folds cross validation on random forest classifier for
# mushroom data k times and displayes the results in the file
```

```
#   'rand_forest_results.txt'.  (This is major overkill because
# the package 'random-forest' conducts it's own cross validation
# as a part of model generation, but I am doing it explicitly to
#   demonstrate the results over k-folds first hand.)
#
#     ###     By              ###
#
# Written by Christopher Carlson
#
#     ###     For             ###
#
# Written initially for Western Michigan University's Summer 1 2015
#     Semester course, CS 5950 - Machine Learning.
#
#########                              ##########

results_file='results/rf_results.txt'

# This matrix will hold the final results after running the complete
# cross validation.
confusion_matrix_averages <- vector("list", (length(folds)-1))



# Begin results file.
write("\t\tBEGIN RANDOM FOREST RESULTS\n", file=results_file,
    ncolumns = 1, append=FALSE)


# In this section of the code, the goal is to cross-validate over each
# of the folds.  So for i in nFolds, it will make the i'th fold the
# testing data, and it will build a tree from each of the remaining
# folds.  Then it will test the fit of each tree on the test fold.
#
# The results of each test are added to a list, and finally at the
# end all the average test perfomance is calculated and reported.
for( i in 1:(length(folds)))
{
  # Name i'th fold 'test' and add the
  # remaining folds to a list called 'train'
  test  <- folds[[i]]
  train <- folds[c(seq(1:(length(folds))))[-i]]



  # First generate the models for each of the training data lists.
  fits  <- lapply(train, FUN=grow_forest, test_data=test)
```

```
  # Generate a list of confusion matrices.
  conf_mats <- lapply(fits, FUN=get_rf_conf_mat)

  # Add the average confusion matrix for this iteration to the
  # confusion_matrix_averages list.
  confusion_matrix_averages[[i]]<- apply(simplify2array(conf_mats),
      c(1,2), mean)

  # Print the confusion matrices to the file.
  write_message(paste0("Iteration ", i, ":\n"), results_file)
  lapply(conf_mats, FUN=write_confusion_matrix, r_file=results_file)

 }

# Write the average confusion matrices to the results file.
write_message("Averages of each CV iteration:\n", results_file)
lapply(confusion_matrix_averages, FUN=write_confusion_matrix, r_file=results_file

# Finally, generate a single confusion matrix from all the average
# confusion matrices and write it to file.
final_confusion_matrix <- apply(simplify2array(confusion_matrix_averages),
              c(1,2), mean)
write_message(paste0("Average Confusion Matrix from ", length(folds),
          "-folds cross validation:\n"), results_file)
write_confusion_matrix(final_confusion_matrix, results_file)
```