Homework 8

Anton Yang

Instructions: Please list your name and student number clearly. In order to receive credit for a problem, your solution must show sufficient detail so that the grader can determine how you obtained your answer.

Submit a single pdf generated using R Markdown. All R code should be included, as well as all output produced. Upload your work to the Canvas course site.

Problem 1

Recall the dataset `tumor.csv` used in previous homework assignments. As a reminder, you may need to change certain variables to factors.

a) Create a training set containing a random sample of 90% observations, and a test set containing the remaining 10% of the observations. Remember to set the seed to 1 for consistent results.

```
library(tree)
set.seed(1)
data<-read.csv("tumor.csv")
split<-sample(nrow(data), 0.9*nrow(data))
training_set<-data[split,]
test_set<-data[-split,]
```

b) Fit a tree to the training data, with `Diagnosis` as the response and the other variables as predictors. Use the `summary()` function to produce summary statistics for the tree, and describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?

```
training_set$Diagnosis<-as.factor(training_set$Diagnosis)
model<-tree(Diagnosis~.,data=training_set)
summary(model)

##
## Classification tree:
## tree(formula = Diagnosis ~ ., data = training_set)
## Variables actually used in tree construction:
## [1] "Concave.Points" "Area"           "Texture"        "Perimeter"
## Number of terminal nodes:  9
## Residual mean deviance:  0.1964 = 98.81 / 503
## Misclassification error rate: 0.03906 = 20 / 512
```

According to the summary, there's 9 terminal nodes and with 0.03906 training error rate. The summary also shows that there 0.1964 residual mean deviance.

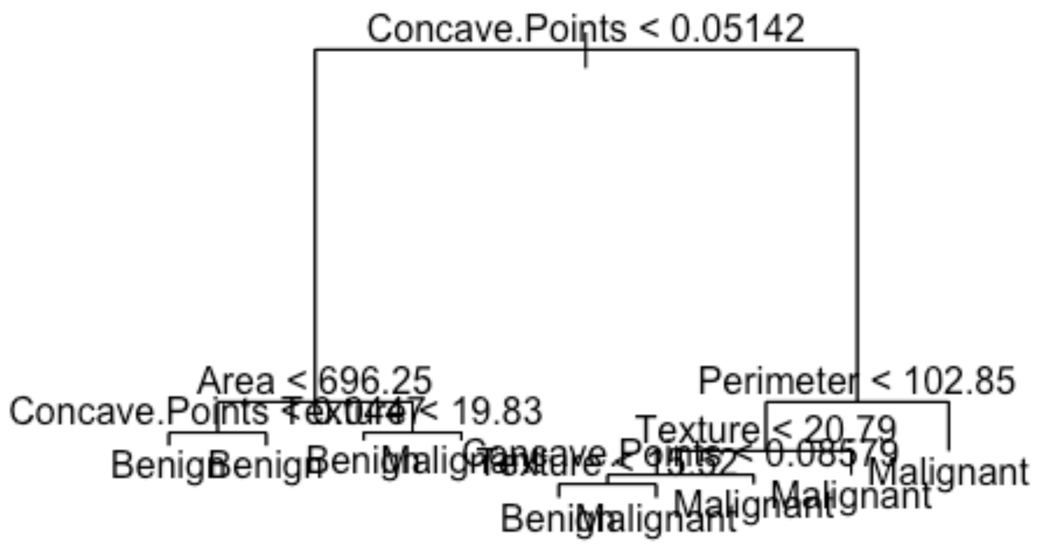
c) Type in the name of the tree object in order to get a detailed text output. Pick one of the terminal nodes, and interpret the information displayed.

```
print(model)

## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 512 674.300 Benign ( 0.63086 0.36914 )
##    2) Concave.Points < 0.05142 316 126.600 Benign ( 0.94937 0.05063 )
##      4) Area < 696.25 303  73.940 Benign ( 0.97360 0.02640 )
##        8) Concave.Points < 0.0447 289  42.190 Benign ( 0.98616 0.01384 ) *
##        9) Concave.Points > 0.0447 14  16.750 Benign ( 0.71429 0.28571 ) *
##      5) Area > 696.25 13  17.320 Malignant ( 0.38462 0.61538 )
##       10) Texture < 19.83 7   8.376 Benign ( 0.71429 0.28571 ) *
##       11) Texture > 19.83 6   0.000 Malignant ( 0.00000 1.00000 ) *
##    3) Concave.Points > 0.05142 196 141.700 Malignant ( 0.11735 0.88265 )
##      6) Perimeter < 102.85 66  85.340 Malignant ( 0.34848 0.65152 )
##       12) Texture < 20.79 42  57.840 Benign ( 0.54762 0.45238 )
##         24) Concave.Points < 0.08579 36  47.090 Benign ( 0.63889 0.36111 )
##           48) Texture < 15.52 13   0.000 Benign ( 1.00000 0.00000 ) *
##           49) Texture > 15.52 23  31.490 Malignant ( 0.43478 0.56522 ) *
##         25) Concave.Points > 0.08579 6   0.000 Malignant ( 0.00000 1.00000
) *
##       13) Texture > 20.79 24   0.000 Malignant ( 0.00000 1.00000 ) *
##      7) Perimeter > 102.85 130   0.000 Malignant ( 0.00000 1.00000 ) *
```

According to the model, the node number 48, which is texture, shows that if texture is less than 15.52, the predicted class for this node is Benign. The probabilities for the classes Benign and Malignant are (1.000, 0.000) respectively. This means that all 13 observations reaching this node are predicted to Benign with a probability of 100%.

d) Create a plot of the tree, and interpret the results.

```
plot(model)
text(model,pretty=0)
```



Based on the tree, the root node is Concave Points. If the Concave Points is less than 0.05142, it will go to the left side, and if greater, then it will go to the right side. We can see that there's more splits it Concave Point is greater than 0.05142, and there's a lot of requirement in order for the predicted class to be Benign. If the Concave Point is greater, then there's only one way in order for the predicted class to be Malignant.

e) Predict the response on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?

```
prediction <- predict(model, newdata = test_set, type="class")


conf_matrix <- table(test_set$Diagnosis, prediction)

print(conf_matrix)

##            prediction
##             Benign Malignant
##   Benign        30         4
##   Malignant      2        21

misclassification_rate <- mean(test_set$Diagnosis != prediction)

misclassification<-1-sum(diag(conf_matrix))/sum(conf_matrix)
```

```
print(misclassification)

## [1] 0.1052632
```

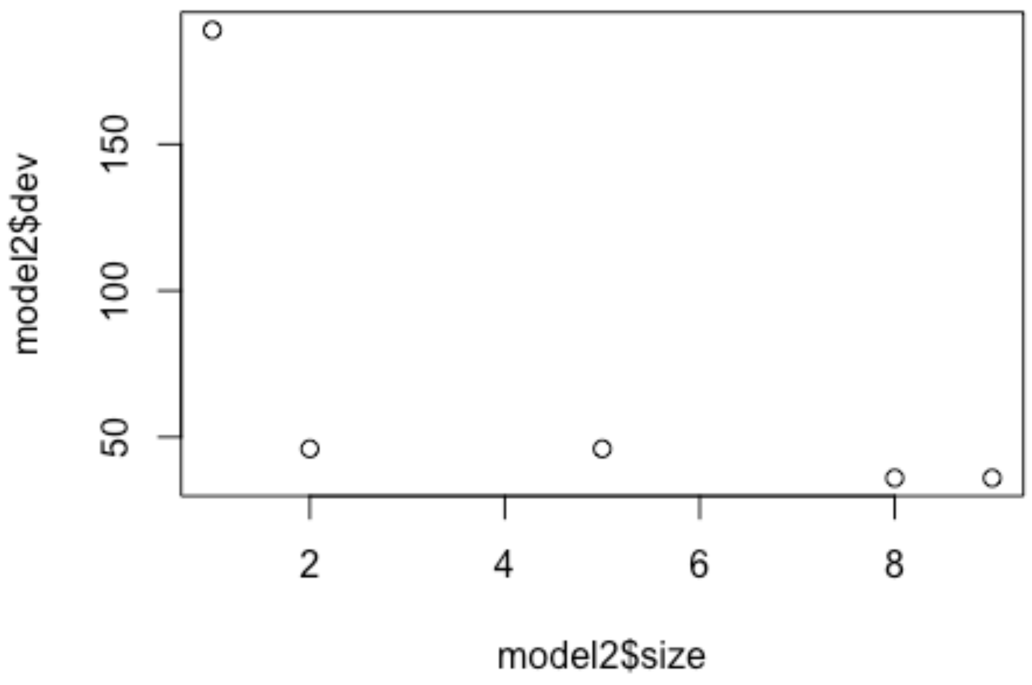
We can see that the test error rate is 0.1052632.

f) Apply the `cv.tree()` function to the training set in order to determine the optimal tree size.

```
set.seed(1)
model2<-cv.tree(model, FUN=prune.misclass)
print(model2)

## $size
## [1] 9 8 5 2 1
##
## $dev
## [1]  36  36  46  46 189
##
## $k
## [1]       -Inf   0.000000   3.000000   3.333333 150.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

g) Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.

```
plot(model2$size,model2$dev)
```



h) Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to selection of a pruned tree, then create a pruned tree with five terminal nodes.

```
prune_model<-prune.misclass(model, best=5)
```

We can see that the cross validation shows the lowest error happen at 9 terminal nodes, which means there's no lead to selection of a pruned tree, so we'll create a prune model with 5 terminal nodes.

i) Compare the *training* error rates between the pruned and unpruned trees. Which is higher?

```
prediction1<-predict(model, new_data=training_set,type="class")
conf_matrix1<-table(training_set$Diagnosis, prediction1)
print(conf_matrix1)

##            prediction1
##             Benign Malignant
##   Benign       313        10
##   Malignant     10       179

misclassification1<-1-sum(diag(conf_matrix1))/sum(conf_matrix1)
print(misclassification1)
```

```
## [1] 0.0390625

prediction2<-predict(prune_model, newdata=training_set, type="class")
conf_matrix2<-table(training_set$Diagnosis,prediction2)
print(conf_matrix2)

##            prediction2
##             Benign Malignant
##   Benign       323         0
##   Malignant     29       160

misclassification2<-1-sum(diag(conf_matrix2))/sum(conf_matrix2)

print(misclassification2)

## [1] 0.05664062
```

We can see that the pruned model has a higher misclassification rate than the unpruned tree. This means that the unpruned tree is better at classifying for the training set than the pruned tree.

j) Compare the *test* error rates between the pruned and unpruned trees. Which is higher?

```
prediction3<-predict(model, newdata=test_set,type="class")
conf_matrix3<-table(test_set$Diagnosis, prediction3)
print(conf_matrix3)

##            prediction3
##             Benign Malignant
##   Benign        30         4
##   Malignant      2        21

misclassification3<-1-sum(diag(conf_matrix3))/sum(conf_matrix3)
print(misclassification3)

## [1] 0.1052632

prediction4<-predict(prune_model, newdata=test_set, type="class")
conf_matrix4<-table(test_set$Diagnosis,prediction4)
print(conf_matrix4)

##            prediction4
##             Benign Malignant
##   Benign        33         1
##   Malignant      7        16

misclassification4<-1-sum(diag(conf_matrix4))/sum(conf_matrix4)

print(misclassification4)

## [1] 0.1403509
```

We can see that the unpruned model also win for the test set. This means that it's best to use unpruned model for both training prediction and test prediction.

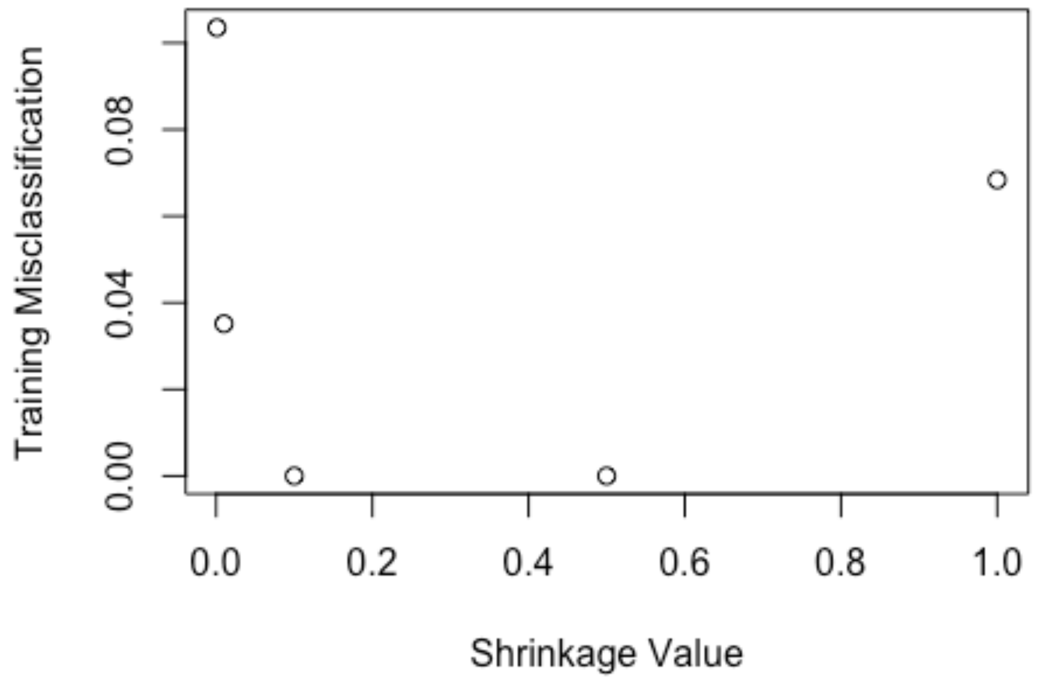
k) Now apply bagging to the training set. What is the test misclassification rate for this result?

```
set.seed(1)
library(randomForest)

## randomForest 4.7-1.1

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

bagging_model<-randomForest(Diagnosis~., data=training_set)
prediction5<-predict(bagging_model, newdata=test_set)
conf_matrix5<-table(test_set$Diagnosis, prediction5)
print(conf_matrix5)

##             prediction5
##              Benign Malignant
##   Benign        31         3
##   Malignant      1        22

misclassification5<-1-sum(diag(conf_matrix5))/sum(conf_matrix5)

print(misclassification5)

## [1] 0.07017544
```

We can see that the misclassification for the bagging method is significantly lower than the nonbagging model.

l) Perform boosting on the training set with 1,000 trees for a range of values of the shrinkage parameter λ. Produce a plot with different shrinkage values on the x-axis and the corresponding *training* misclassification rate on the y-axis. Use 0.5 as the cut point for classification of Benign and Malignant.

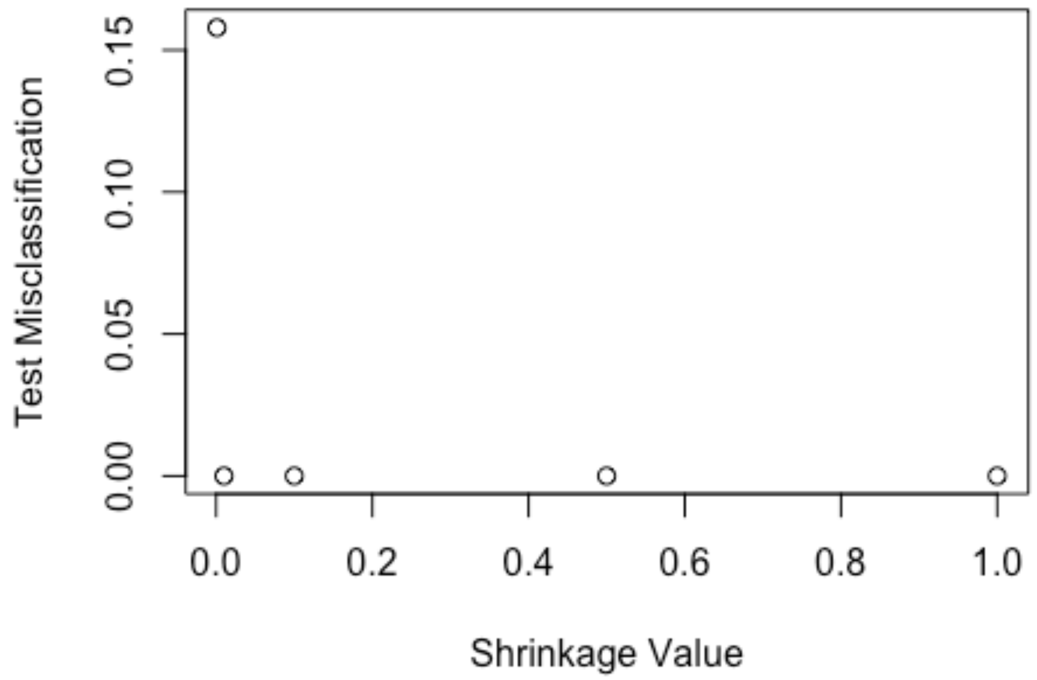
```
set.seed(1)
library(gbm)

## Loaded gbm 2.1.9

## This version of gbm is no longer under development. Consider transitioning
to gbm3, https://github.com/gbm-developers/gbm3

lambda<-c(0.001,0.01,0.1,0.5,1)

training_error<-rep(NA, length(lambda))
training_set2<-training_set
training_set2$Diagnosis<-ifelse(training_set$Diagnosis == "Malignant", 1,0)

for (i in 1:length(lambda)) {
  boosting_model <- gbm(Diagnosis ~ .,
                              distribution = "bernoulli",
                              data = training_set2,
                              n.trees = 1000,
                              shrinkage = lambda[i])

  prediction6 <- predict(boosting_model, training_set2, n.trees = 1000)

  predicted_classes<-ifelse(prediction6 > 0.5, 1, 0)

  misclassification6 <- mean(predicted_classes != training_set2$Diagnosis)

  training_error[i] <- misclassification6
}

plot(lambda, training_error, xlab="Shrinkage Value", ylab="Training
Misclassification")
```



m) Produce a plot with different shrinkage values on the x-axis and the corresponding *test* misclassification rate on the y-axis.

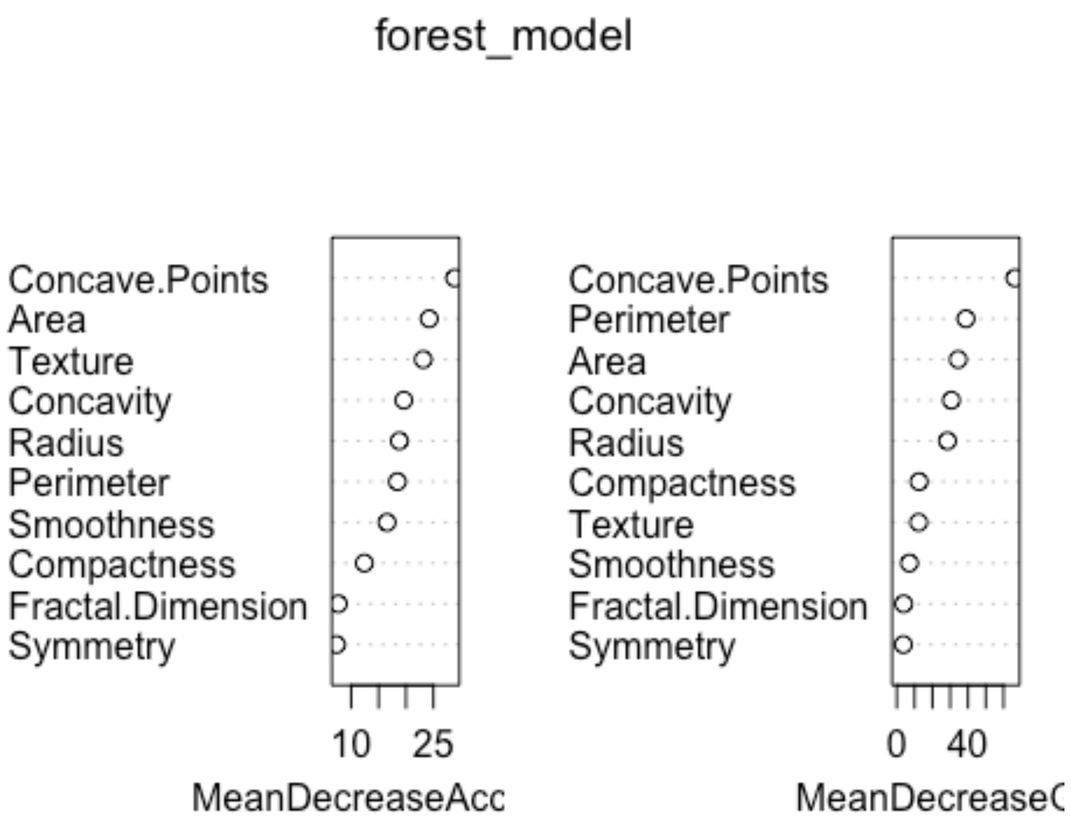
```
lambda<-c(0.001,0.01,0.1,0.5,1)

test_error<-rep(NA, length(lambda))
```

```r
test_set2<-test_set
test_set2$Diagnosis<-ifelse(test_set2$Diagnosis == "Malignant", 1,0)

for (i in 1:length(lambda)) {
  boosting_model2 <- gbm(Diagnosis ~ .,
                         distribution = "bernoulli",
                         data = test_set2,
                         n.trees = 1000,
                         shrinkage = lambda[i])

  prediction7 <- predict(boosting_model2, test_set2, n.trees = 1000)

  predicted_classes2<-ifelse(prediction7 > 0.5, 1, 0)

  misclassification7 <- mean(predicted_classes2 != test_set2$Diagnosis)

  test_error[i] <- misclassification7
}

plot(lambda, test_error, xlab="Shrinkage Value", ylab="Test
Misclassification")
```

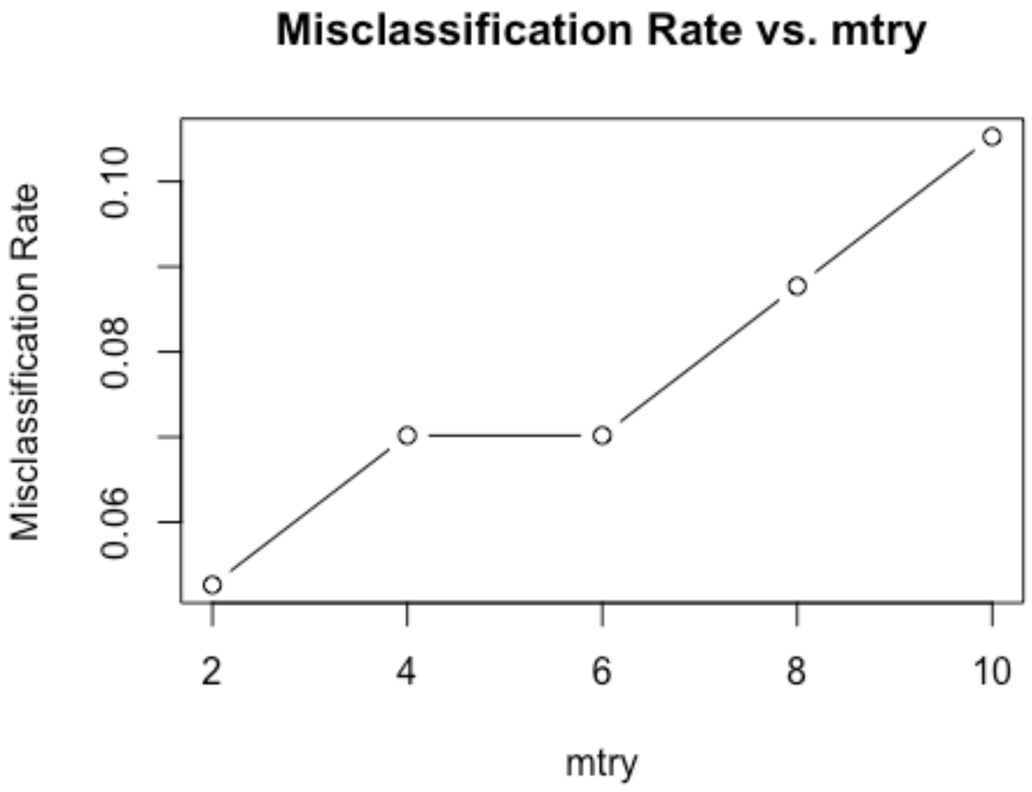


n) Use random forests to analyze this data. What test misclassification rate do you obtain (again, use 0.5 to classify either Malignant or Benign)? Use the `importance()` function to determine which variables are most important. Describe the effect of m, the number of variables considered at each split, on the error rate obtained.

```
forest_model<-randomForest(Diagnosis~., data=training_set, importance = TRUE)

prediction8<-predict(forest_model, test_set)

conf_matrix5<-table(test_set$Diagnosis,prediction8)
print(conf_matrix5)

##            prediction8
##             Benign Malignant
##   Benign        31         3
##   Malignant      1        22

misclassification9<-1-sum(diag(conf_matrix5))/sum(conf_matrix5)

print(misclassification9)

## [1] 0.07017544

var_importance<-importance(forest_model)
print(var_importance)

##                         Benign Malignant MeanDecreaseAccuracy
MeanDecreaseGini
## Radius            14.9026033 12.554806             18.797294
28.499044
## Texture           14.5636509 20.031592             23.147144
12.303404
## Perimeter         13.1807814 13.770106             18.436406
38.878980
## Area              19.6525651 15.653924             24.241173
34.496431
## Smoothness         5.6536740 15.661321             16.549769
7.100927
## Compactness        8.9443525  7.983897             12.405370
12.535633
## Concavity          9.9778071 16.616085             19.604953
30.682759
## Concave.Points    18.3935638 22.252202             28.876952
66.373499
## Symmetry          -0.6045406  9.107559              7.332785
3.701963
## Fractal.Dimension  6.6633526  3.046377              7.664803
3.885588

varImpPlot(forest_model)
```



```r
mtry_values <- c(2, 4, 6, 8, 10)

misclassification_rates <- numeric(length(mtry_values))


for (i in seq_along(mtry_values)) {
  forest_model <- randomForest(Diagnosis ~ .,
                               data = training_set,
                               ntree = 500,
                               mtry = mtry_values[i],
                               importance = TRUE)

  predictions <- predict(forest_model, test_set)

  misclassification_rate <- mean(predictions != test_set$Diagnosis)

  misclassification_rates[i] <- misclassification_rate
}

plot(mtry_values, misclassification_rates, type = "b",
     xlab = "mtry", ylab = "Misclassification Rate",
     main = "Misclassification Rate vs. mtry")
```



As we can see from the plot, as the m increases, the error rate becomes higher. So it's the best to leave mtry as 2.