PSTAT 131 Homework 2

Luis Aragon and Ronak Parikh

5/1/2019

Set Up

```
library(tidyverse)
library(tree)
library(plyr)
library(dplyr)
library(class)
library(rpart)
library(maptree)
library(ROCR)
library(reshape2)
```

```
# Read in Data
spam <- read_table2("spambase.tab", guess_max=2000)
spam <- spam %>%
  mutate(y = factor(y, levels=c(0,1), labels=c("good", "spam"))) %>% # label as factors
  mutate_at(.vars=vars(-y), .funs=scale)

dim(spam)
```

```
## [1] 4601   58
```

```
head(spam)
```

```
## # A tibble: 6 x 58
##   word_freq_make word_freq_addre~ word_freq_all word_freq_3d word_freq_our
##            <dbl>            <dbl>         <dbl>        <dbl>         <dbl>
## 1         -0.342           0.331          0.713      -0.0469        0.0116
## 2          0.345           0.0519         0.435      -0.0469       -0.256
## 3         -0.146          -0.165          0.852      -0.0469        1.36
## 4         -0.342          -0.165         -0.557      -0.0469        0.473
## 5         -0.342          -0.165         -0.557      -0.0469        0.473
## 6         -0.342          -0.165         -0.557      -0.0469        2.29
## # ... with 53 more variables: word_freq_over <dbl>,
## #   word_freq_remove <dbl>, word_freq_internet <dbl>,
## #   word_freq_order <dbl>, word_freq_mail <dbl>, word_freq_receive <dbl>,
## #   word_freq_will <dbl>, word_freq_people <dbl>, word_freq_report <dbl>,
## #   word_freq_addresses <dbl>, word_freq_free <dbl>,
## #   word_freq_business <dbl>, word_freq_email <dbl>, word_freq_you <dbl>,
## #   word_freq_credit <dbl>, word_freq_your <dbl>, word_freq_font <dbl>,
## #   word_freq_000 <dbl>, word_freq_money <dbl>, word_freq_hp <dbl>,
## #   word_freq_hpl <dbl>, word_freq_george <dbl>, word_freq_650 <dbl>,
## #   word_freq_lab <dbl>, word_freq_labs <dbl>, word_freq_telnet <dbl>,
## #   word_freq_857 <dbl>, word_freq_data <dbl>, word_freq_415 <dbl>,
## #   word_freq_85 <dbl>, word_freq_technology <dbl>, word_freq_1999 <dbl>,
## #   word_freq_parts <dbl>, word_freq_pm <dbl>, word_freq_direct <dbl>,
## #   word_freq_cs <dbl>, word_freq_meeting <dbl>, word_freq_original <dbl>,
```

```
## #   word_freq_project <dbl>, word_freq_re <dbl>, word_freq_edu <dbl>,
## #   word_freq_table <dbl>, word_freq_conference <dbl>, char_freq_. <dbl>,
## #   char_freq_..1 <dbl>, char_freq_..2 <dbl>, char_freq_..3 <dbl>,
## #   char_freq_..4 <dbl>, char_freq_..5 <dbl>,
## #   capital_run_length_average <dbl>, capital_run_length_longest <dbl>,
## #   capital_run_length_total <dbl>, y <fct>
```

```r
# Error Rate Function
calc_error_rate <- function(predicted.value, true.value){
  return(mean(true.value!=predicted.value))
}

# Keep Track of our model performance
records = matrix(NA, nrow=3, ncol=2)
colnames(records) <- c("train.error","test.error")
rownames(records) <- c("knn","tree","logistic")

# Train/Test split
set.seed(1)
test.indices = sample(1:nrow(spam), 1000)
spam.train=spam[-test.indices,]
spam.test=spam[test.indices,]

dim(spam.train)
```

```
## [1] 3601   58
```

```r
dim(spam.test)
```

```
## [1] 1000   58
```

```r
# Folds for CV
nfold = 10
set.seed(1)
folds = seq.int(nrow(spam.train)) %>% ## sequential obs ids
  cut(breaks = nfold, labels=FALSE) %>% ## sequential fold ids
  sample ## random fold ids
```

K-Nearest Neighbor Method

PART 1

```r
# CV
do.chunk <- function(chunkid, folddef, Xdat, Ydat, k){

  train = (folddef!=chunkid)

  Xtr = Xdat[train,]
  Ytr = Ydat[train]
  Xvl = Xdat[!train,]
  Yvl = Ydat[!train]

  ## get classifications for current training chunks
  predYtr = knn(train = Xtr, test = Xtr, cl = Ytr, k = k)
```

```
  ## get classifications for current test chunk
  predYvl = knn(train = Xtr, test = Xvl, cl = Ytr, k = k)

  data.frame(fold = chunkid, # k folds
             train.error = calc_error_rate(predYtr, Ytr),
             val.error = calc_error_rate(predYvl, Yvl))
}
```

```
# Check for missing values
sum(is.na(spam.train))
```

```
## [1] 0
```

```
sum(is.na(spam.test))
```

```
## [1] 0
```

```
# Clean Up our train and test sets
Xtrain <- spam.train %>% select(-y)
Ytrain <- spam.train$y

Xtest <- spam.test %>% select(-y)
Ytest <- spam.test$y
```

```
kvec = c(1, seq(10, 50, length.out=5))
kvec
```

```
## [1]  1 10 20 30 40 50
```

```
error.folds = NULL

set.seed(1)

for (j in kvec){
  tmp = ldply(1:nfold, do.chunk, # Apply do.chunk() function to each fold
              folddef=folds, Xdat=Xtrain, Ydat=Ytrain, k=j)# Necessary arguments to be passed into do.c
  tmp$neighbors = j # Keep track of each value of neighbors
  error.folds = rbind(error.folds, tmp) # combine results
}
```

```
# Transform the format of error.folds for further convenience
errors = melt(error.folds, id.vars=c('fold', 'neighbors'), value.name='error')

# Choose the number of neighbors which minimizes validation error
val.error.means = errors %>%
  # Select all rows of validation errors
  filter(variable=='val.error') %>%
  # Group the selected data frame by neighbors
  group_by(neighbors, variable) %>%
  # Calculate CV error rate for each k
  summarise_each(funs(mean), error) %>%
  # Remove existing group
  ungroup() %>%
    filter(error==min(error))

val.error.means
```

```
## # A tibble: 1 x 3
```

```
##   neighbors variable  error
##       <dbl> <fct>     <dbl>
## 1        10 val.error 0.101
```

```
# Best # of neighbors
best.kfold = max(val.error.means$neighbors)
best.kfold
```

```
## [1] 10
```

When k = 10, we get the smallest estimated test error.

PART 2

```
set.seed(1)

# Test
pred.YTest = knn(train=Xtrain, test=Xtest, cl=Ytrain, k=best.kfold)

# Confusion matrix
conf.matrix = table(predicted=pred.YTest, true=Ytest)
conf.matrix
```

```
##          true
## predicted good spam
##      good  569   62
##      spam   31  338
```

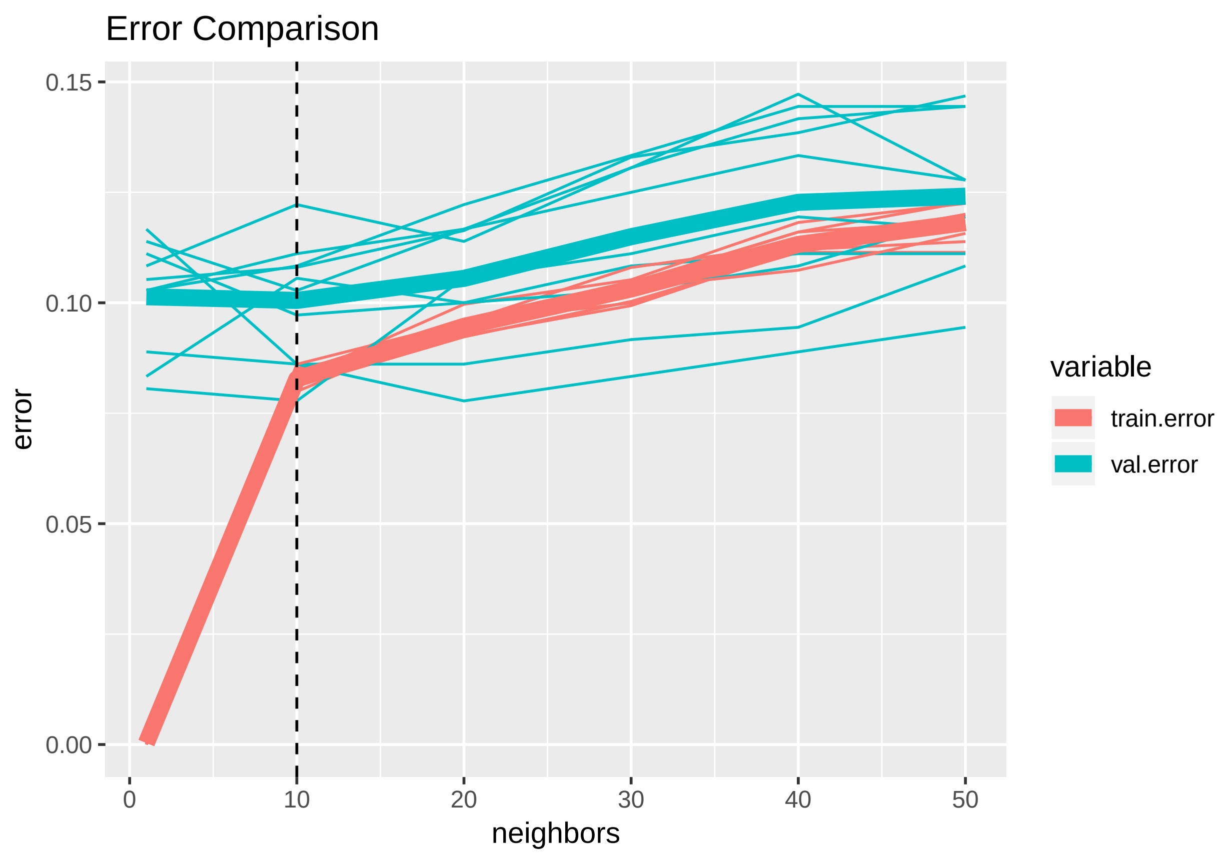
```
# Test accuracy rate
sum(diag(conf.matrix)/sum(conf.matrix))
```

```
## [1] 0.907
```

```
# Test error rate
1 - sum(diag(conf.matrix)/sum(conf.matrix))
```

```
## [1] 0.093
```

```
# Plot errors
ggplot(errors, aes(x=neighbors, y=error, color=variable))+
    geom_line(aes(group=interaction(variable,fold))) +
    stat_summary(aes(group=variable), fun.y="mean", geom='line', size=3) +
    geom_vline(aes(xintercept=best.kfold), linetype='dashed')+
  ggtitle('Error Comparison')
```




```
# Train Error
val.error.means[1,3]
```

```
## # A tibble: 1 x 1
##   error
##   <dbl>
## 1 0.101
```

```
# Test Error
calc_error_rate(pred.YTest, Ytest)
```

```
## [1] 0.093
```

```
# Store Error in Records
records[1,1] <- as.numeric(unlist(val.error.means[1,3]))
records[1,2] <- calc_error_rate(pred.YTest, Ytest)
records
```

```
##            train.error test.error
## knn          0.1005255      0.093
## tree                NA         NA
## logistic            NA         NA
```

Decision Trees

PART 3

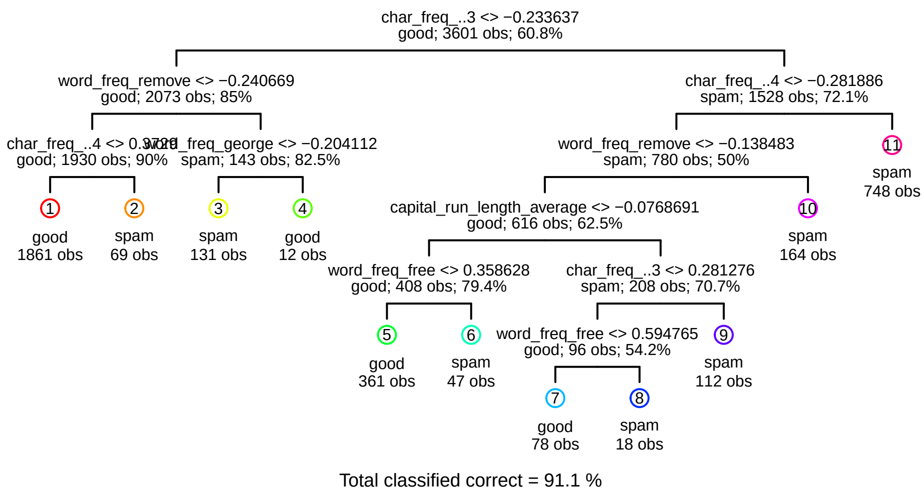
```
set.seed(1)

# Set control
control <- tree.control(nrow(spam.train), minsize = 5, mindev = 1e-5)

# Create full tree
spamtree = tree(y ~., control = control, data = spam.train)
summary(spamtree)
```

```
##
## Classification tree:
## tree(formula = y ~ ., data = spam.train, control = control)
## Variables actually used in tree construction:
##  [1] "char_freq_..3"               "word_freq_remove"
##  [3] "char_freq_..4"               "word_freq_george"
##  [5] "word_freq_hp"                "capital_run_length_longest"
##  [7] "word_freq_receive"           "word_freq_free"
##  [9] "word_freq_direct"            "capital_run_length_average"
## [11] "word_freq_re"                "word_freq_you"
## [13] "capital_run_length_total"    "word_freq_credit"
## [15] "word_freq_our"               "word_freq_your"
## [17] "word_freq_will"              "char_freq_..1"
## [19] "word_freq_meeting"           "word_freq_1999"
## [21] "word_freq_make"              "word_freq_hpl"
## [23] "char_freq_."                 "word_freq_over"
## [25] "word_freq_font"              "word_freq_report"
## [27] "word_freq_money"             "word_freq_address"
## [29] "word_freq_all"               "word_freq_000"
## [31] "word_freq_data"              "word_freq_project"
## [33] "word_freq_people"            "word_freq_email"
## [35] "word_freq_415"               "word_freq_edu"
## [37] "word_freq_technology"        "word_freq_mail"
## [39] "word_freq_business"          "char_freq_..2"
## [41] "word_freq_order"             "char_freq_..5"
## Number of terminal nodes:  184
## Residual mean deviance:  0.04748 = 162.2 / 3417
## Misclassification error rate: 0.01333 = 48 / 3601
```

We have 184 leaf nodes and there are 48 training observations out of 3601 that were misclassified.

PART 4

```
# Prune and draw tree
prune = prune.tree(spamtree, best = 10, method = 'misclass')
draw.tree(prune, nodeinfo = TRUE, cex = 0.5)
```



PART 5

```r
set.seed(1)

# K-Fold cross validation
cv = cv.tree(spamtree, FUN=prune.misclass, K=nfold, rand = folds)

# Print out cv
cv
```

```
## $size
##  [1] 184 153 148 134 116 108  96  76  70  65  62  46  37  30  28  20  19
## [18]  17  13  11   8   7   6   3   2   1
##
## $dev
##  [1]  351  351  351  351  351  351  351  351  351  351  351  351  351  355
## [15]  355  355  355  361  374  382  396  399  446  639  729 1413
##
## $k
##  [1]        -Inf   0.0000000   0.4000000   0.5000000   0.6666667
##  [6]   0.7500000   0.9166667   1.0000000   1.3333333   1.6000000
## [11]   1.6666667   2.0000000   3.0000000   4.0000000   4.5000000
## [16]   4.8750000   5.0000000   5.5000000   6.7500000   7.5000000
## [21]  12.0000000  17.0000000  31.0000000  80.0000000  93.0000000
## [26] 676.0000000
##
```

```
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

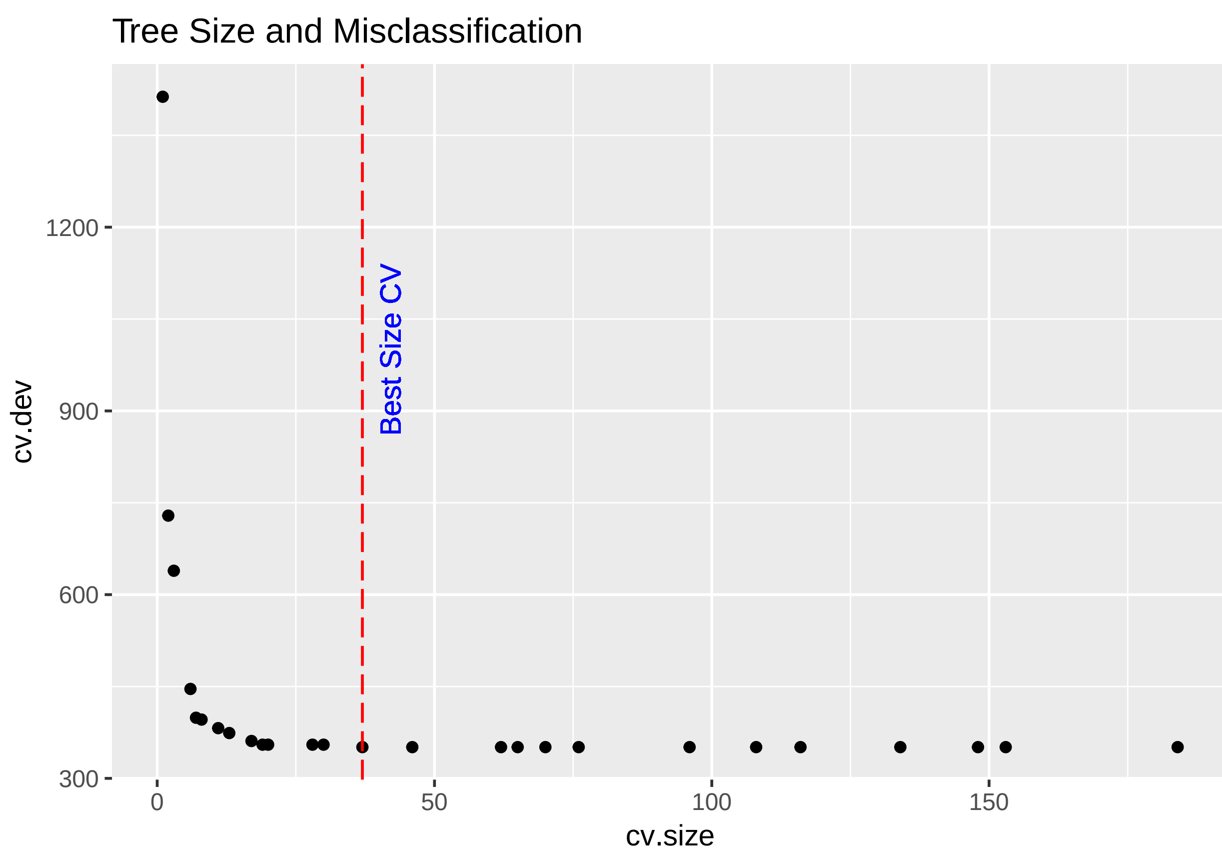
```r
# Best size
best.size.cv = min(cv$size[cv$dev == min(cv$dev)])
best.size.cv
```

```
## [1] 37
```

The optimal tree is size is 37.

```r
# Plot Missclassification vs Tree Size
dtrees_df <- data.frame(cv$size, cv$dev)
ggplot(dtrees_df, aes(x=cv.size, y=cv.dev)) +
  geom_point() +
  ggtitle('Tree Size and Misclassification') +
  geom_vline(xintercept = best.size.cv, linetype='longdash', color = 'red') +
  geom_text(aes(x=best.size.cv+5, label="Best Size CV", y=1000), colour="blue", angle=90, text=element_t
```

```
## Warning: Ignoring unknown parameters: text
```



PART 6

```r
spamtree.pruned = prune.tree(spamtree, best = best.size.cv, method = 'misclass')
```

```r
# Predict on test set
pred.pt.prune = predict(spamtree.pruned, Xtest, type="class")

# Obtain confusion matrix
err.pt.prune = table(pred.pt.prune, Ytest)
err.pt.prune
```

```
##              Ytest
## pred.pt.prune good spam
##          good  575   47
##          spam   25  353
```

```r
# Test error rate (Classification Error)
1-sum(diag(err.pt.prune))/sum(err.pt.prune)
```

```
## [1] 0.072
```

```r
# Get test prediction
pred.pt.pruneTrain = predict(spamtree.pruned, Xtrain, type="class")

# Calculate Train and Test Error
dtree_test_error <- calc_error_rate(pred.pt.prune, Ytest)
dtree_train_error <- calc_error_rate(pred.pt.pruneTrain, Ytrain)

# Put in records
records[2,1] <- dtree_train_error
records[2,2] <- dtree_test_error
records
```

```
##          train.error test.error
## knn       0.10052555      0.093
## tree      0.05165232      0.072
## logistic          NA         NA
```

Logistics Regression

PART 7

7a

Given,

$$p(z) = \frac{e^z}{1+e^z}$$
$$p(1+e^z) = e^z$$
$$p + pe^z = e^z$$

$$pe^z = e^z - p$$
$$p = 1 - \frac{p}{e^z}$$
$$1 - p = \frac{p}{e^z}$$
$$e^z = \frac{p}{1-p}$$
$$z = ln(\frac{p}{1-p})$$

7b

$$p = \frac{e^{\beta_0+\beta_1 x_1}}{1+e^{\beta_0+\beta_1 x_1}}$$

When $x_1 = x_1 + 2$:

$$p = \frac{e^{\beta_0+\beta_1(x_1+2)}}{1+e^{\beta_0+\beta_1(x_1+2)}} = \frac{e^{\beta_0+\beta_1 x_1+2\beta_1}}{1+e^{\beta_0+\beta_1 x_1+2\beta_1}} = \frac{e^{\beta_0+\beta_1 x_1}e^{2\beta_1}}{1+e^{\beta_0+\beta_1 x_1}e^{2\beta_1}}$$

As x increases by 2, the odds is multiplied by $e^{2\beta_1}$.

$$\lim_{x\to\infty} p = 0$$

Also, as x goes to infinity, the numerator becomes smaller and the denominator becomes bigger. Therefore, the probability gets closer to 0.

$$\lim_{x\to-\infty} p = 1$$

As x goes to negative infinity, the probability goes to 1.

PART 8

```r
set.seed(1)

# Fit logistic regression
glm.fit = glm(y ~ ., data=spam.train, family=binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
# Summary
summary(glm.fit)
```

```
##
## Call:
## glm(formula = y ~ ., family = binomial, data = spam.train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.1553  -0.2110   0.0000   0.1198   5.3014
##
## Coefficients:
##                            Estimate Std. Error z value Pr(>|z|)
## (Intercept)               -12.84435    2.10718  -6.096 1.09e-09 ***
```

```
## word_freq_make            -0.13463    0.07713  -1.745 0.080919 .
## word_freq_address         -0.20268    0.10824  -1.872 0.061152 .
## word_freq_all              0.05956    0.06013   0.991 0.321862
## word_freq_3d               2.83556    2.25293   1.259 0.208171
## word_freq_our              0.37610    0.07847   4.793 1.64e-06 ***
## word_freq_over             0.23702    0.07526   3.149 0.001636 **
## word_freq_remove           0.92476    0.14524   6.367 1.92e-10 ***
## word_freq_internet         0.21442    0.07757   2.764 0.005703 **
## word_freq_order            0.12879    0.08586   1.500 0.133615
## word_freq_mail             0.03812    0.04744   0.804 0.421577
## word_freq_receive         -0.03848    0.06417  -0.600 0.548687
## word_freq_will            -0.10603    0.07190  -1.475 0.140272
## word_freq_people          -0.05848    0.07699  -0.760 0.447484
## word_freq_report           0.04631    0.04676   0.990 0.322001
## word_freq_addresses        0.41023    0.23462   1.748 0.080381 .
## word_freq_free             0.78080    0.12784   6.108 1.01e-09 ***
## word_freq_business         0.38772    0.11003   3.524 0.000425 ***
## word_freq_email            0.06350    0.07139   0.890 0.373724
## word_freq_you              0.16018    0.07222   2.218 0.026563 *
## word_freq_credit           0.43676    0.26061   1.676 0.093758 .
## word_freq_your             0.25711    0.06955   3.697 0.000219 ***
## word_freq_font             0.25301    0.20332   1.244 0.213341
## word_freq_000              0.71311    0.16363   4.358 1.31e-05 ***
## word_freq_money            0.29526    0.11831   2.496 0.012570 *
## word_freq_hp              -3.05776    0.56651  -5.397 6.76e-08 ***
## word_freq_hpl             -0.97608    0.44395  -2.199 0.027903 *
## word_freq_george         -37.66212    7.73935  -4.866 1.14e-06 ***
## word_freq_650              0.40553    0.16152   2.511 0.012049 *
## word_freq_lab             -1.50081    1.04987  -1.430 0.152854
## word_freq_labs            -0.13027    0.14894  -0.875 0.381765
## word_freq_telnet          -0.06221    0.17332  -0.359 0.719664
## word_freq_857              0.43220    1.31101   0.330 0.741647
## word_freq_data            -0.51143    0.20884  -2.449 0.014328 *
## word_freq_415             -4.05702    1.34626  -3.014 0.002582 **
## word_freq_85              -1.08129    0.44277  -2.442 0.014602 *
## word_freq_technology       0.30600    0.14361   2.131 0.033100 *
## word_freq_1999            -0.03759    0.08720  -0.431 0.666359
## word_freq_parts           -0.13351    0.10771  -1.240 0.215156
## word_freq_pm              -0.26365    0.19283  -1.367 0.171540
## word_freq_direct          -0.11977    0.13906  -0.861 0.389090
## word_freq_cs             -17.91992    8.96274  -1.999 0.045567 *
## word_freq_meeting         -2.87094    1.09516  -2.621 0.008755 **
## word_freq_original        -0.18975    0.16951  -1.119 0.262971
## word_freq_project         -0.84156    0.33192  -2.535 0.011231 *
## word_freq_re              -0.88509    0.17902  -4.944 7.65e-07 ***
## word_freq_edu             -1.16372    0.27209  -4.277 1.89e-05 ***
## word_freq_table           -0.14180    0.11737  -1.208 0.226970
## word_freq_conference      -1.73004    0.77730  -2.226 0.026034 *
## char_freq_.               -0.37561    0.13418  -2.799 0.005120 **
## char_freq_..1             -0.14717    0.10153  -1.449 0.147217
## char_freq_..2             -0.03451    0.07874  -0.438 0.661189
## char_freq_..3              0.21780    0.05543   3.929 8.52e-05 ***
## char_freq_..4              1.38495    0.19770   7.005 2.47e-12 ***
## char_freq_..5              1.18928    0.50712   2.345 0.019019 *
```

```
## capital_run_length_average   0.57032    0.64766   0.881 0.378538
## capital_run_length_longest   1.29365    0.52356   2.471 0.013479 *
## capital_run_length_total     0.82240    0.17105   4.808 1.52e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4823.9  on 3600  degrees of freedom
## Residual deviance: 1435.1  on 3543  degrees of freedom
## AIC: 1551.1
##
## Number of Fisher Scoring iterations: 13
```

```
# Test
trainP <- predict(glm.fit, type = "response")
testP <- predict(glm.fit, newdata = Xtest, type = "response")

# Save the predicted labels using 0.5 as a threshold
spam.train <- spam.train %>%
  mutate(predspam=as.factor(ifelse(trainP > 0.5, "spam", "good")))

spam.test <- spam.test %>%
  mutate(predspam=as.factor(ifelse(testP > 0.5, "spam", "good")))
```

```
# Store errors in records
records[3,1] <- calc_error_rate(spam.train$predspam, Ytrain)
records[3,2] <- calc_error_rate(spam.test$predspam, Ytest)

# Compare errors
records
```

```
##          train.error test.error
## knn       0.10052555      0.093
## tree      0.05165232      0.072
## logistic  0.07081366      0.081
```

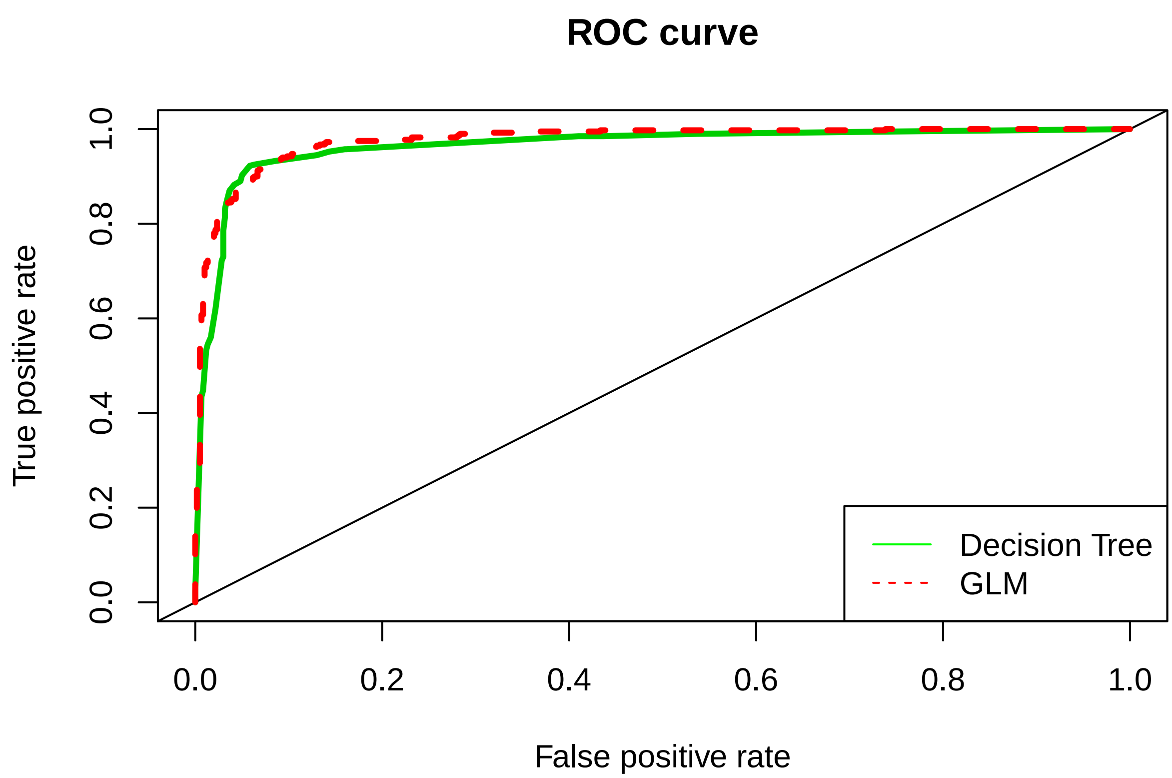
The method with the lowest classification error is decision trees, with a test error of 0.072.

ROC

PART 9

```
# First arument is the prob.training, second is true labels
prob.tree.Test = predict(spamtree.pruned, Xtest, type="vector")
prob.glm.Test = predict(glm.fit, Xtest, type="response")

# ROC Set Up
pred.tree.ROC = prediction(prob.tree.Test[,2], Ytest)
pred.glm.ROC = prediction(prob.glm.Test, Ytest)

# We want TPR on the y axis and FPR on the x axis
perf.tree = performance(pred.tree.ROC, measure="tpr", x.measure="fpr")

# We want TPR on the y axis and FPR on the x axis
```

```r
perf.glm = performance(pred.glm.ROC, measure="tpr", x.measure="fpr")

plot(perf.tree, col=3, lwd=3, main="ROC curve")
abline(0,1)
plot(perf.glm, add = TRUE, col = 2, lwd = 3, lty = 2)

legend("bottomright", legend=c("Decision Tree", "GLM"),
       col=c("green", "red"), lty=1:2)
```




```r
# Calculate AUC
auc.tree = performance(pred.tree.ROC, "auc")@y.values
auc.tree
```

```
## [[1]]
## [1] 0.9647083
```

```r
# Calculate AUC
auc.glm = performance(pred.glm.ROC, "auc")@y.values
auc.glm
```

```
## [[1]]
## [1] 0.9758875
```

By using the Area Under the Curve metric, we can say that the Logistic Model is better because there is more area under the curve.

PART 10

We are more concerned about false positive rates that are too large, because that means we are placing good emails into the spam folder (these could be important emails). It is important that good emails are not thrown in the spam folder. We are not worried about true positive rates being small because for most people, it is not extremely necessary to put every single spam in the spam folder. The user can always delete the spam in their regular inbox themselves, but they cannot easily recovery or notice if a good email is thrown into spam.