PSTAT 131/231 HW #2

Lash Tan (231) and Jacobo Pereira-Pacheco (131)

4/27/2018

```
setwd('/home/ltan/HW/HW2')
library(tree)
library(randomForest)
library(class)
library(rpart)
library(maptree)
library(ROCR)
library(reshape2)
library(tidyverse)
library(dplyr)
```

```
spam <- read_table2("spambase.tab", guess_max=2000)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   capital_run_length_longest = col_integer(),
##   capital_run_length_total = col_integer(),
##   y = col_integer()
## )
```

```
## See spec(...) for full column specifications.
```

```
spam <- spam %>%
          mutate(y = factor(y, levels=c(0,1), labels=c("good", "spam"))) %>%
            # label as factors
          mutate_at(.vars=vars(-y), .funs=scale) # scale others

calc_error_rate <- function(predicted.value, true.value){
  return(mean(true.value!=predicted.value))
} ## calculate the misclassification rate

records = matrix(NA, nrow=3, ncol=2)
colnames(records) <- c("train.error","test.error")
rownames(records) <- c("knn","tree","logistic")
records ## tracking training and test error rate for classification methods
```

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

```
set.seed(1)
test.indices = sample(1:nrow(spam), 1000)
  ## test.indices is a vector of row indices that represent the training set
  ## chosen randomly
spam.train=spam[-test.indices,]
  ## spam.train is a data frame containing all the columns of the 3601 obs in training set
spam.test=spam[test.indices,]
```

```
  ## spam.test is a data frame containing all of the columns of the 1000 obs in test set
YTrain = spam.train$y  ## the response variable for the training set
XTrain = spam.train %>% select(-y) ## the design matrix for the training set

YTest = spam.test$y ## the response variable for the test set
XTest = spam.test %>% select(-y) ## the design matrix for the test set

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
## folds is a vector of 1-10 assigning fold IDs to each observation
## (360 each, 361 for chunk 1)
```

K-Nearest Neighbor Method

1) Selecting number of neighbors

```
set.seed(1)
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(train.error = calc_error_rate(predYtr, Ytr),
             val.error = calc_error_rate(predYvl, Yvl))
}

kvec = c(1, seq(10, 50, length.out=5)) ## (1, 10, 20, 30, 40, 50 for choices of neighbors)
error.folds = NULL ## will contain list of errors for choice of neighbors
```

```
set.seed(1)
for (j in kvec){
  tmp = plyr::ldply(1:nfold, do.chunk, folddef = folds,
                    Xdat = XTrain, Ydat = YTrain, k = j)
  tmp$neighbors = j
  error.folds <- rbind(error.folds, tmp)
}
head(error.folds)
```

```
##   train.error val.error neighbors
## 1   0.0006173   0.10526         1
## 2   0.0006171   0.11111         1
## 3   0.0006171   0.10278         1
## 4   0.0006171   0.11389         1
```

```
## 5   0.0000000   0.11667         1
## 6   0.0003085   0.08889         1
```

```
(error.folds %>%
  group_by(neighbors) %>%
  summarize_all(funs(mean)))
```

```
## # A tibble: 6 x 3
##   neighbors train.error val.error
##       <dbl>       <dbl>     <dbl>
## 1         1   0.0003394    0.1014
## 2        10   0.0827547    0.1005
## 3        20   0.0946650    0.1055
## 4        30   0.1031194    0.1150
## 5        40   0.1132092    0.1227
## 6        50   0.1181460    0.1241
```

```
(best.kfold <- (error.folds %>%
  group_by(neighbors) %>%
  summarize_all(funs(mean)) %>%
  filter(val.error == min(val.error)))
  $neighbors)
```

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

The value of K that leads to the smallest estimated test error is 10 neighbors.

2) Training and Test Errors

```
set.seed(1)
pred.YTrain <- knn(XTrain, XTrain, YTrain, best.kfold)
pred.YTest <- knn(XTrain, XTest, YTrain, best.kfold)
calc.YTrain <- calc_error_rate(pred.YTrain, YTrain)
calc.YTest <- calc_error_rate(pred.YTest, YTest)
records[1] <- calc.YTrain
records[4] <- calc.YTest
records
```

```
##          train.error test.error
## knn          0.08192       0.09
## tree              NA         NA
## logistic          NA         NA
```

Using the calc_error_rate() function, the 10-nearest neighbor error values in the records matrix are 0.08192169 for the training set and 0.09 for the test set.

Decision Tree Method

3) Controlling Decision Tree Construction

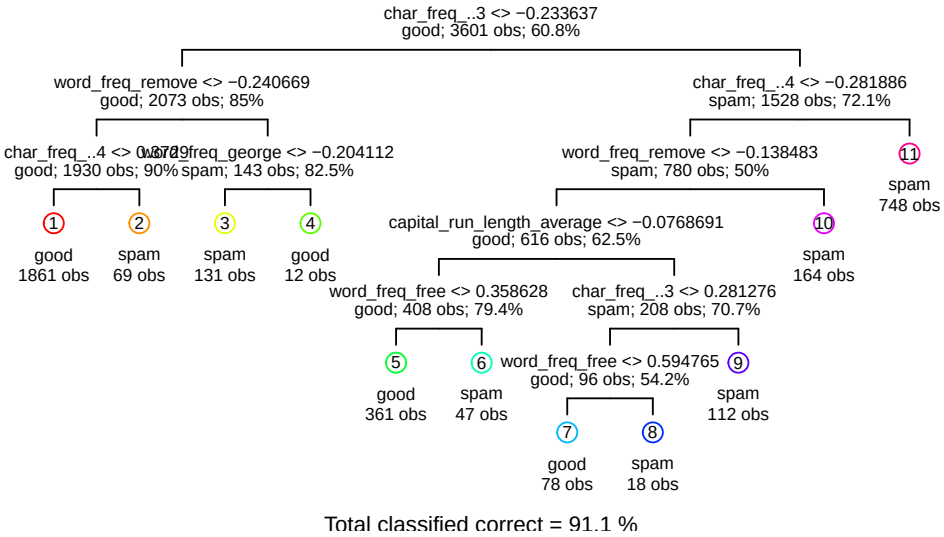
```
tc <- tree.control(nobs = nrow(spam.train), minsize = 5, mindev = 1e-5)
spamtree <- tree(y ~ ., data = spam.train,control = tc)
summary(spamtree)
```

```
##
## Classification tree:
## tree(formula = y ~ ., data = spam.train, control = tc)
## 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:  0.0475 = 162 / 3420
## Misclassification error rate: 0.0133 = 48 / 3601
```

There are 184 leaf nodes with 48 of the training set observations being misclassified.

4) Decision Tree Pruning

```
spam.prune <- prune.tree(spamtree, best = 10, method = "misclass")
draw.tree(spam.prune, nodeinfo = T, cex = .6)
```



The 10-node tree can be visualized as shown with 91.1% correct classification.

5)

```
(tree.cv <- cv.tree(spamtree, folds, FUN = prune.misclass, K = nfold))
```

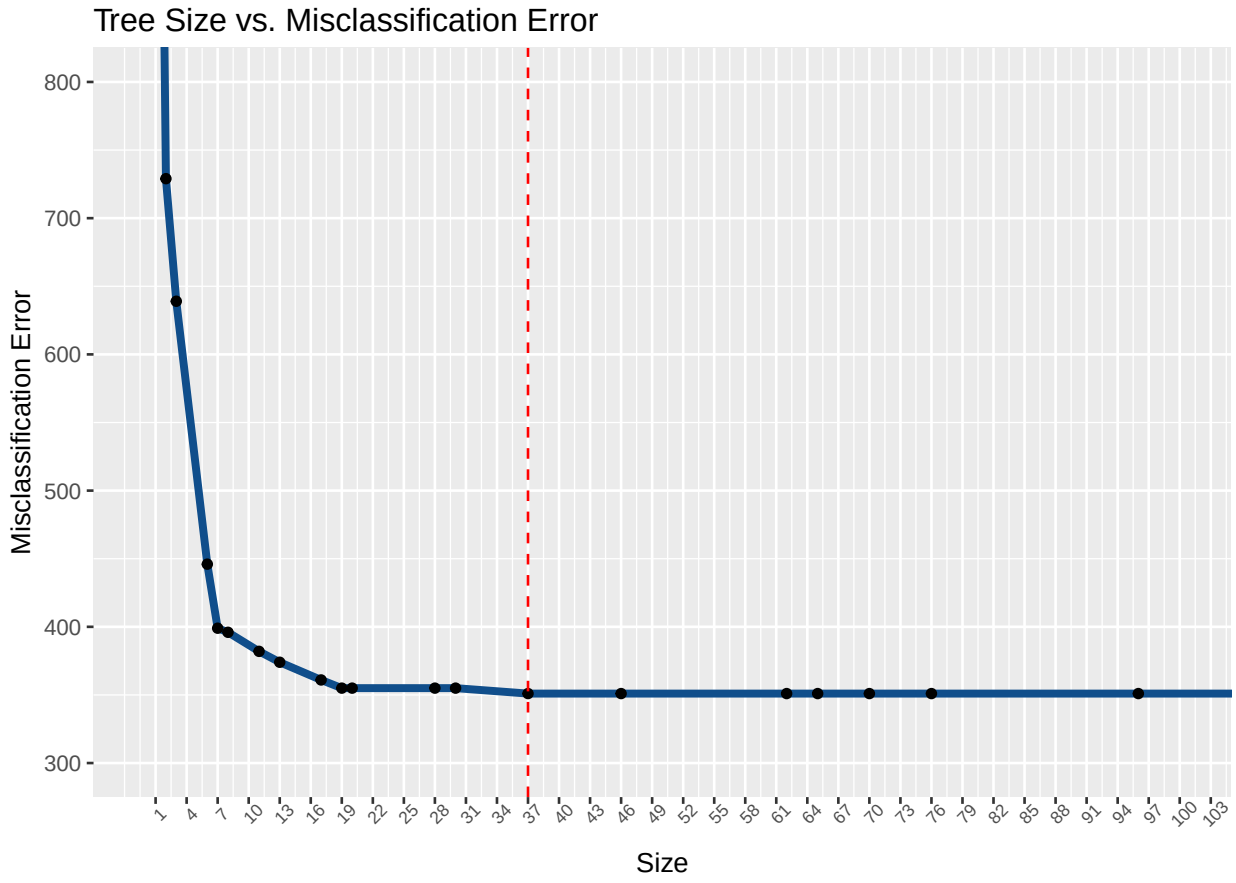
```
## $size
##  [1] 184 153 148 134 116 108  96  76  70  65  62  46  37  30  28  20  19
## [18]  17  13  11   8   7   6   3   2   1
##
## $dev
##  [1]  351  351  351  351  351  351  351  351  351  351  351  351  351  355
## [15]  355  355  355  361  374  382  396  399  446  639  729 1413
##
## $k
##  [1]     -Inf   0.0000   0.4000   0.5000   0.6667   0.7500   0.9167
##  [8]   1.0000   1.3333   1.6000   1.6667   2.0000   3.0000   4.0000
## [15]   4.5000   4.8750   5.0000   5.5000   6.7500   7.5000  12.0000
## [22]  17.0000  31.0000  80.0000  93.0000 676.0000
##
## $method
## [1] "misclass"
##
## attr(,"class")
```

```
## [1] "prune"         "tree.sequence"
```

```
tree.temp <- data.frame('size'= rev(tree.cv$size), 'misclass' = rev(tree.cv$dev))
(best.size.cv <- (tree.temp %>% filter(misclass == min(misclass)))$size[1])
```

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

```
tree.temp %>% ggplot(., mapping = aes(x = .$size, y = .$misclass)) +
  geom_line(col = 'dodgerblue4', lwd = 1.5) +
  geom_point() +
  geom_vline(xintercept = best.size.cv, col = 'red', linetype = 'dashed') +
  labs(title = 'Tree Size vs. Misclassification Error',
       x = 'Size', y = 'Misclassification Error') +
  coord_cartesian(xlim = c(0,100), ylim = c(300,800)) +
  scale_x_continuous(breaks = seq(1,184,3)) +
  theme(axis.text.x = element_text(angle=45, size = 7))
```



The optimal tree size is 37 as shown in the plot. This produces the smallest size that gives the absolute lowest misclassification error. If we wanted to shrink the tree further, we would need to sacrifice correct classification rates to do so.

6) Training and Test Errors

```
spamtree.pruned <- prune.misclass(spamtree, best = best.size.cv)
predict.train <- predict(spamtree.pruned, spam.train, type = 'class')
records[2] <- calc_error_rate(predict.train, YTrain)
```

```
predict.test <- predict(spamtree.pruned, spam.test, type = 'class')
records[5] <- calc_error_rate(predict.test, YTest)
records
```

```
##          train.error test.error
## knn          0.08192      0.090
## tree         0.05165      0.072
## logistic          NA         NA
```

From the calc_error_rate() function, the decision tree method produce misclassification errors of 0.05165232 for the training set and 0.072 for the test set.

---

7)

a)

Given $p(z) = \frac{e^z}{1+e^z},\ z \in \mathbb{R}$, we have

$$\frac{1}{p} = \frac{1+e^z}{e^z},\ p \neq 0$$

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

$$\frac{1}{p} - 1 = \frac{1}{e^z}$$

$$\frac{1-p}{p} = \frac{1}{e^z}$$

$$\frac{p}{1-p} = e^z,\ p \neq 1$$

$$ln(\frac{p}{1-p}) = z(p),\ p \in (0,1)$$

which shows that z maps from $\mathbb{R} \to (0,1)$.

b)

We define:

$$z = \beta_0 + \beta_1 x_1,\ p = logistic(z)$$

$$ln(odds) = z \quad \Rightarrow \quad odds = e^z = e^{\beta_0+\beta_1 x_1}$$

which we designate our "original odds." Now,

$$odds_{x_1+2} = e^{\beta_0+\beta_1(x_1+2)}$$

$$= e^{\beta_0+\beta_1 x_1+2\beta_1}$$

$$= e^{2\beta_1} \times e^{\beta_0+\beta_1 x_1}$$

$$= e^{2\beta_1} \times odds$$

Therefore, increasing x_1 by two gives an odds equal to $e^{2\beta_1}$ times the original odds. Now,

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

Since we assume β_1 is negative, as $x_1 \to \infty, \beta_1 x_1 \to -\infty$. Therefore,

$$\lim_{x\to\infty} \frac{e^{-\infty}}{1+e^{-\infty}} = \frac{0}{1} = 0$$

Also,

$$\lim_{x\to-\infty} \frac{e^{\beta_0+\beta_1 x_1}}{1+e^{\beta_0+\beta_1 x_1}} = \lim_{x\to\infty} \frac{e^{\beta_0+|\beta_1 x_1|}}{1+e^{\beta_0+|\beta_1 x_1|}}$$

and, by L'Hopital's rule, the probability converges to 1.

8)

```
spam.fit <- glm(y~., data = spam.train, family = "binomial")
summary(spam.fit)
```

```
##
## Call:
## glm(formula = y ~ ., family = "binomial", data = spam.train)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -4.155  -0.211   0.000   0.120   5.301
##
## Coefficients:
##                           Estimate Std. Error z value Pr(>|z|)
## (Intercept)               -12.8443     2.1072   -6.10  1.1e-09 ***
## word_freq_make             -0.1346     0.0771   -1.75  0.08092 .
## word_freq_address          -0.2027     0.1082   -1.87  0.06115 .
## word_freq_all               0.0596     0.0601    0.99  0.32186
## word_freq_3d                2.8356     2.2529    1.26  0.20817
## word_freq_our               0.3761     0.0785    4.79  1.6e-06 ***
## word_freq_over              0.2370     0.0753    3.15  0.00164 **
## word_freq_remove            0.9248     0.1452    6.37  1.9e-10 ***
## word_freq_internet          0.2144     0.0776    2.76  0.00570 **
## word_freq_order             0.1288     0.0859    1.50  0.13362
## word_freq_mail              0.0381     0.0474    0.80  0.42158
## word_freq_receive          -0.0385     0.0642   -0.60  0.54869
## word_freq_will             -0.1060     0.0719   -1.47  0.14027
## word_freq_people           -0.0585     0.0770   -0.76  0.44748
## word_freq_report            0.0463     0.0468    0.99  0.32200
## word_freq_addresses         0.4102     0.2346    1.75  0.08038 .
## word_freq_free              0.7808     0.1278    6.11  1.0e-09 ***
## word_freq_business          0.3877     0.1100    3.52  0.00043 ***
## word_freq_email             0.0635     0.0714    0.89  0.37372
## word_freq_you               0.1602     0.0722    2.22  0.02656 *
## word_freq_credit            0.4368     0.2606    1.68  0.09376 .
## word_freq_your              0.2571     0.0696    3.70  0.00022 ***
## word_freq_font              0.2530     0.2033    1.24  0.21334
## word_freq_000               0.7131     0.1636    4.36  1.3e-05 ***
## word_freq_money             0.2953     0.1183    2.50  0.01257 *
## word_freq_hp               -3.0578     0.5665   -5.40  6.8e-08 ***
## word_freq_hpl              -0.9761     0.4439   -2.20  0.02790 *
## word_freq_george          -37.6621     7.7393   -4.87  1.1e-06 ***
## word_freq_650               0.4055     0.1615    2.51  0.01205 *
```

```
## word_freq_lab               -1.5008     1.0499   -1.43  0.15285
## word_freq_labs              -0.1303     0.1489   -0.87  0.38177
## word_freq_telnet            -0.0622     0.1733   -0.36  0.71966
## word_freq_857                0.4322     1.3110    0.33  0.74165
## word_freq_data              -0.5114     0.2088   -2.45  0.01433 *
## word_freq_415               -4.0570     1.3463   -3.01  0.00258 **
## word_freq_85                -1.0813     0.4428   -2.44  0.01460 *
## word_freq_technology         0.3060     0.1436    2.13  0.03310 *
## word_freq_1999              -0.0376     0.0872   -0.43  0.66636
## word_freq_parts             -0.1335     0.1077   -1.24  0.21516
## word_freq_pm                -0.2636     0.1928   -1.37  0.17154
## word_freq_direct            -0.1198     0.1391   -0.86  0.38909
## word_freq_cs               -17.9199     8.9627   -2.00  0.04557 *
## word_freq_meeting           -2.8709     1.0952   -2.62  0.00876 **
## word_freq_original          -0.1897     0.1695   -1.12  0.26297
## word_freq_project           -0.8416     0.3319   -2.54  0.01123 *
## word_freq_re                -0.8851     0.1790   -4.94  7.7e-07 ***
## word_freq_edu               -1.1637     0.2721   -4.28  1.9e-05 ***
## word_freq_table             -0.1418     0.1174   -1.21  0.22697
## word_freq_conference        -1.7300     0.7773   -2.23  0.02603 *
## char_freq_.                 -0.3756     0.1342   -2.80  0.00512 **
## char_freq_..1               -0.1472     0.1015   -1.45  0.14722
## char_freq_..2               -0.0345     0.0787   -0.44  0.66119
## char_freq_..3                0.2178     0.0554    3.93  8.5e-05 ***
## char_freq_..4                1.3849     0.1977    7.01  2.5e-12 ***
## char_freq_..5                1.1893     0.5071    2.35  0.01902 *
## capital_run_length_average   0.5703     0.6477    0.88  0.37854
## capital_run_length_longest   1.2936     0.5236    2.47  0.01348 *
## capital_run_length_total     0.8224     0.1710    4.81  1.5e-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
##
## Number of Fisher Scoring iterations: 13
```

```
prob.training <- round(predict(spam.fit, type="response"), 8)
prob.outcome.train <- spam.train %>%
                      mutate(predSPAM=as.factor(ifelse(prob.training <=.5, "good", "spam")))
records[3] <- calc_error_rate(prob.outcome.train$predSPAM, YTrain)

prob.test <- round(predict(spam.fit, spam.test, type="response"), 8)
prob.outcome.test <- spam.test %>%
                      mutate(predSPAM=as.factor(ifelse(prob.test <=.5, "good", "spam")))
records[6] <- calc_error_rate(prob.outcome.test$predSPAM, YTest)
records
```

```
##          train.error test.error
## knn          0.08192      0.090
## tree         0.05165      0.072
## logistic     0.07081      0.081
```

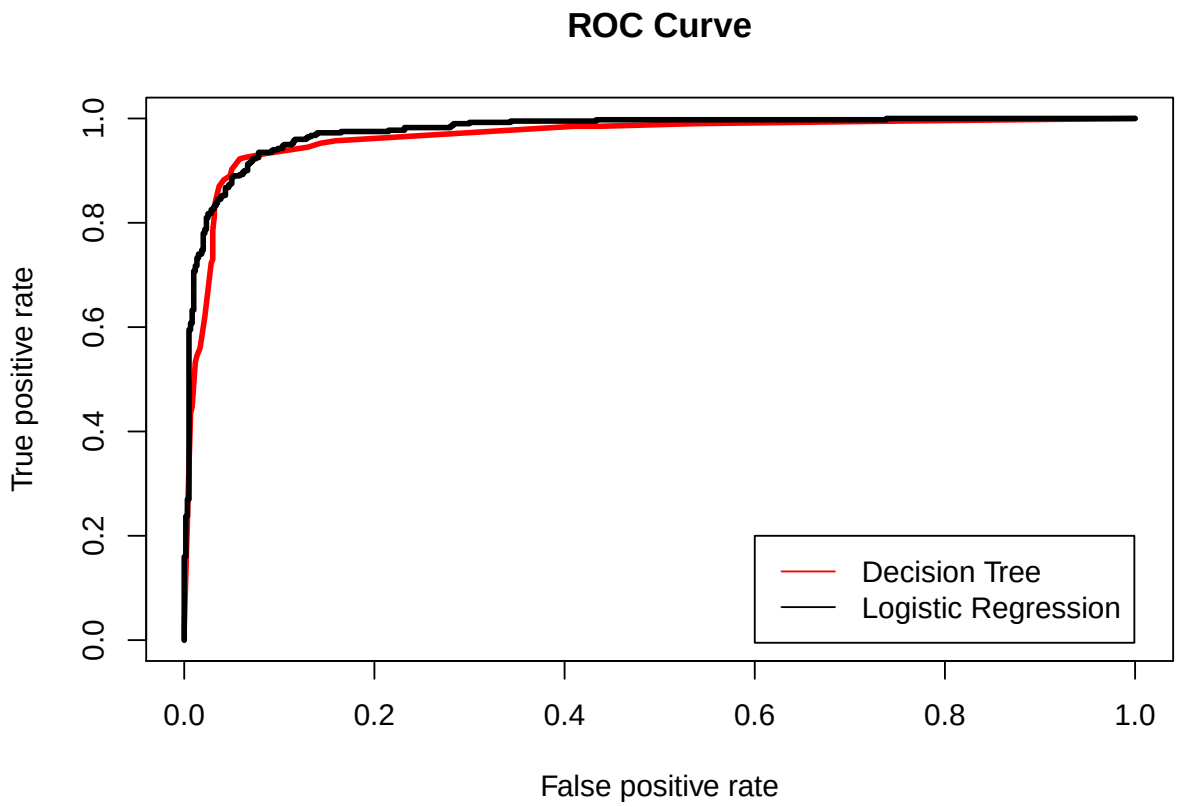
Viewing the full records matrix, the decision tree model has the lowest misclassification rate for this data, outperforming both KNN and logistic regression by this metric.

Receiver Operating Characteristic curve

9) ROC curve

```
tree.prob <- predict(spamtree.pruned, spam.test, type="vector")
tree.pred <- prediction(tree.prob[,2], spam.test$y)
tree.perf <- performance(tree.pred, measure = "tpr", x.measure = "fpr")
plot(tree.perf, col=2, lwd=3, main='ROC Curve')

glm.pred <- prediction(prob.test, prob.outcome.test$y)
glm.perf <- performance(glm.pred, measure = "tpr", x.measure = "fpr")
plot(glm.perf, col=1, lwd=3, main="ROC Curve", add="T")
legend(.6, .2, legend=c("Decision Tree", "Logistic Regression"),
       col=c("red", "black"), lty=1, cex=1)
```




```
performance(tree.pred,"auc")@y.values
```

```
## [[1]]
## [1] 0.9647
```

```
performance(glm.pred,"auc")@y.values
```

```
## [[1]]
## [1] 0.9759
```

The area under the curve (AUC) for our decision tree is 0.9647083 compared to 0.9758875 for logistic regression. Between these two methods, logistic regression performs better by this metric. This is interesting as we saw that the decision tree had a lower misclassification rate; however, AUC is calculated as a function of the true positive rate and the false positive rate.

10)

If we take "positive"" to mean "spam," the designer of a spam filter would be more concerned with false postive rates. For example, one would want to ensure the filter correctly classifies email as spam when appropriate because it would be more consequential if important/sensitive emails were not seen because they were incorrectly categorized as spam, whereas a few spam emails that are allowed through the filter can simply be deleted.

---

11)

As described in the problem, we classify $\hat{Y} = 1$ when $\mathbb{P}(Y = 1|X = x) > \tau$ for some probability threshold τ and that f_k is a multivariate normal density with covariance Σ_k and mean μ_k. As $\mathbb{P}(Y = 1|X = x) = \frac{f_1(x)\pi_1}{\pi_1 f_1(x)+\pi_2 f_2(x)}$, we have

$$\mathbb{P}(Y = 1|X = x) = \frac{f_1(x)\pi_1}{\pi_1 f_1(x) + \pi_2 f_2(x)} > \tau$$

$$\frac{\frac{\pi_1}{(2\pi)^{p/2}|\Sigma_1|^{1/2}} exp[-\frac{1}{2}(x-\mu_1)'\Sigma_1^{-1}(x-\mu_1)]}{\Sigma_{l=1}^{2}\frac{\pi_l}{(2\pi)^{p/2}|\Sigma_l|^{1/2}} exp[-\frac{1}{2}(x-\mu_l)'\Sigma_l^{-1}(x-\mu_l)]} > \tau$$

Taking the reciprocal:

$$\frac{\Sigma_{l=1}^{2}\frac{\pi_l}{(2\pi)^{p/2}|\Sigma_l|^{1/2}} exp[-\frac{1}{2}(x-\mu_l)'\Sigma_l^{-1}(x-\mu_l)]}{\frac{\pi_1}{(2\pi)^{p/2}|\Sigma_1|^{1/2}} exp[-\frac{1}{2}(x-\mu_1)'\Sigma_1^{-1}(x-\mu_1)]} < \frac{1}{\tau}$$

Separating the fraction:

$$1 + \frac{\frac{\pi_2}{(2\pi)^{p/2}|\Sigma_2|^{1/2}} exp[-\frac{1}{2}(x-\mu_2)'\Sigma_2^{-1}(x-\mu_2)]}{\frac{\pi_1}{(2\pi)^{p/2}|\Sigma_1|^{1/2}} exp[-\frac{1}{2}(x-\mu_1)'\Sigma_1^{-1}(x-\mu_1)]} < \frac{1}{\tau}$$

$$\frac{\frac{\pi_2}{(2\pi)^{p/2}|\Sigma_2|^{1/2}} exp[-\frac{1}{2}(x-\mu_2)'\Sigma_2^{-1}(x-\mu_2)]}{\frac{\pi_1}{(2\pi)^{p/2}|\Sigma_1|^{1/2}} exp[-\frac{1}{2}(x-\mu_1)'\Sigma_1^{-1}(x-\mu_1)]} < \frac{1}{\tau} - 1$$

Taking the logarithm:

$$ln(\frac{\frac{\pi_2}{(2\pi)^{p/2}|\Sigma_2|^{1/2}} exp[-\frac{1}{2}(x-\mu_2)'\Sigma_2^{-1}(x-\mu_2)]}{\frac{\pi_1}{(2\pi)^{p/2}|\Sigma_1|^{1/2}} exp[-\frac{1}{2}(x-\mu_1)'\Sigma_1^{-1}(x-\mu_1)]}) < ln(\frac{1-\tau}{\tau})$$

Now, it's a matter of simplifying:

$$ln(\frac{\pi_2}{(2\pi)^{p/2}|\Sigma_2|^{1/2}} exp[-\frac{1}{2}(x-\mu_2)'\Sigma_2^{-1}(x-\mu_2)]) - ln(\frac{\pi_1}{(2\pi)^{p/2}|\Sigma_1|^{1/2}} exp[-\frac{1}{2}(x-\mu_1)'\Sigma_1^{-1}(x-\mu_1)]) < ln(\frac{1-\tau}{\tau})$$

$$ln(\pi_2) - ln(\pi_1) - ln((2\pi)^{\frac{p}{2}}) + ln((2\pi)^{\frac{p}{2}}) - ln(|\Sigma_2^{\frac{1}{2}}|) + ln(|\Sigma_1^{\frac{1}{2}}|) - \frac{1}{2}(x-\mu_2)'\Sigma_2^{-1}(x-\mu_2) + \frac{1}{2}(x-\mu_1)'\Sigma_1^{-1}(x-\mu_1) < ln(\frac{1-\tau}{\tau})$$

As we define $\hat{\delta}_k(x) = -\frac{1}{2}(x-\mu_k)'\Sigma_k^{-1}(x-\mu_k) - \frac{1}{2}ln|\Sigma_k| + ln(\pi_k)$, we now have:

$$\delta_2(x) - \delta_1(x) < ln(\frac{1-\tau}{\tau})$$

or:

$$\delta_1(x) - \delta_2(x) > ln(\frac{\tau}{1-\tau})$$

where $ln(\frac{\tau}{1-\tau})$ is $M(\tau)$. Note that $(x-\mu_k)'\Sigma_k^{-1}(x-\mu_k)$ satisfies a vector quadratic form, meaning our decision boundary is quadratic. With the decision threshold, M(1/2), we have $ln(\frac{\frac{1}{2}}{1-\frac{1}{2}}) = ln(1) = 0$. This is expected as a probability threshold of 1/2 for two class levels simply means we take the class level with the higher likelihood.

12) Variable Standardization and Discretization

```
algae <- read_table2("algaeBloom.txt", col_names =
                       c('season','size','speed','mxPH','mnO2','Cl','NO3','NH4',
                         'oPO4','PO4','Chla','a1','a2','a3','a4','a5','a6','a7'),
                     na="XXXXXXX") %>%
          dplyr::select(-(season:mxPH),-(a2:a7)) %>%
          mutate_at(vars(mnO2:Chla),funs(log))
```

```
## Parsed with column specification:
## cols(
##   season = col_character(),
##   size = col_character(),
##   speed = col_character(),
##   mxPH = col_double(),
##   mnO2 = col_double(),
##   Cl = col_double(),
##   NO3 = col_double(),
##   NH4 = col_double(),
##   oPO4 = col_double(),
##   PO4 = col_double(),
##   Chla = col_double(),
##   a1 = col_double(),
##   a2 = col_double(),
##   a3 = col_double(),
##   a4 = col_double(),
##   a5 = col_double(),
##   a6 = col_double(),
##   a7 = col_double()
## )
```

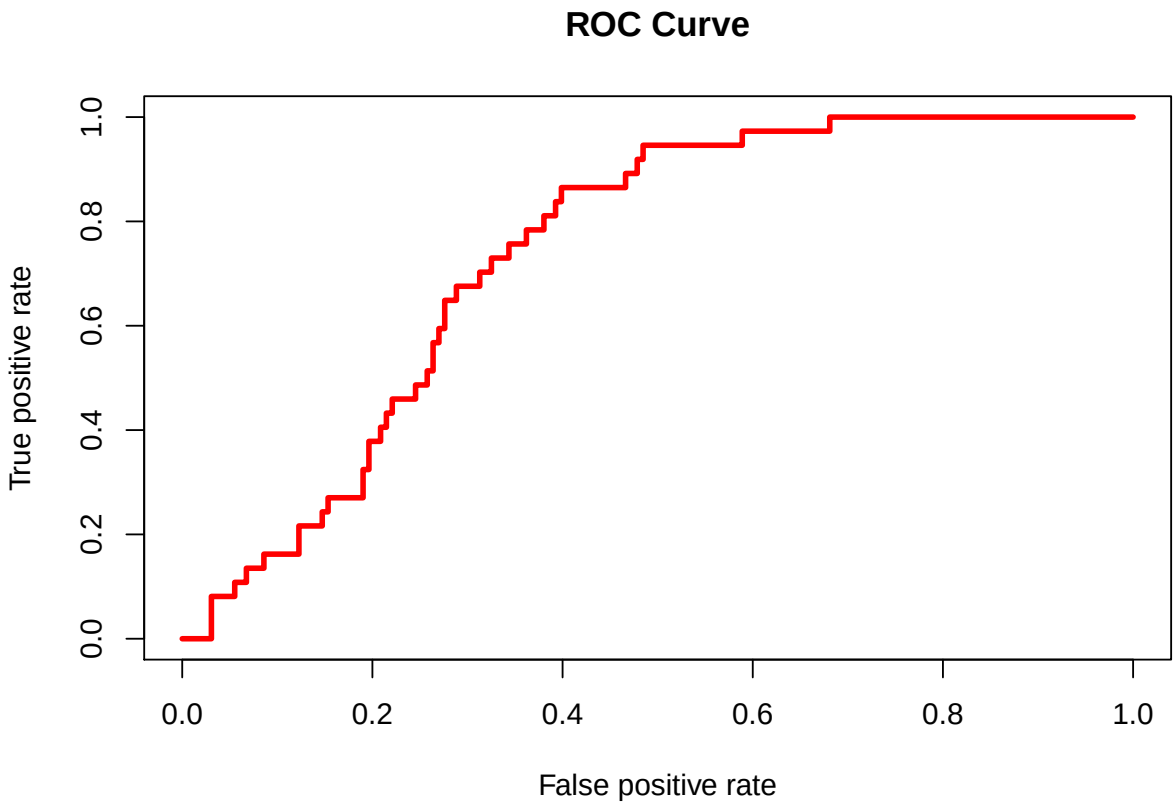
```
algae <- algae %>%
          mutate_at(vars(-a1), funs(ifelse(is.na(.), median(., na.rm=T), .))) %>%
          mutate_at(vars(a1), funs(as.factor(ifelse(a1>.5,'high','low'))))
```

The chemicals mnO2 through Chla have been log-transformed and missing values have been filled in with the respective medians. The algae count a1 has been factored into two levels: 'high' for levels above .5 and 'low' for levels between 0 to .5.

13) Linear and Quadratic Discriminant Analysis

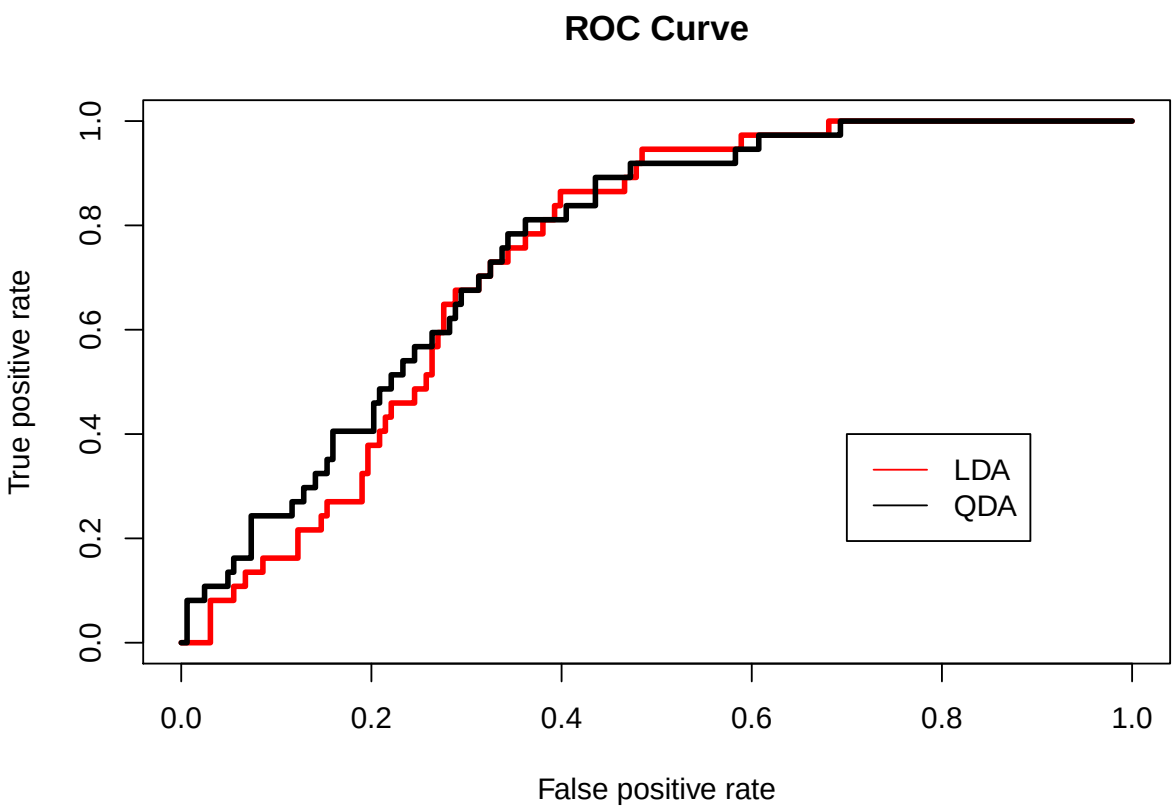
a)

```
algae.lda <- MASS::lda(a1 ~ ., algae, CV=T)
lda.pred <- prediction(algae.lda$posterior[,2], algae$a1)
lda.perf <- performance(lda.pred, measure = "tpr", x.measure = "fpr")
plot(lda.perf, col=2, lwd=3, main='ROC Curve')
```



The ROC for the LDA method can be seen as shown.

```
algae.qda <- MASS::qda(a1 ~ ., algae, CV=T)
qda.pred <- prediction(algae.qda$posterior[,2], algae$a1)
qda.perf <- performance(qda.pred, measure = "tpr", x.measure = "fpr")
plot(lda.perf, col=2, lwd=3, main='ROC Curve')
plot(qda.perf, col=1, lwd=3, main='ROC Curve', add=T)
legend(.7, .4, legend=c("LDA", "QDA"),
       col=c("red", "black"), lty=1, cex=1)
```




```
performance(lda.pred,"auc")@y.values
```

```
## [[1]]
## [1] 0.74
```

```
performance(qda.pred,"auc")@y.values
```

```
## [[1]]
## [1] 0.7573
```

We see that the ROC for the LDA and QDA methods have slight variation, especially toward the lower values of the false positive and true positive rates. The AUC for LDA is 0.7400099 whereas the AUC for QDA is 0.757254215, giving QDA a slight edge. This is to be expected as QDA has less assumptions/restraints than LDA in that the covariances matrix for each class are not equal. We may be led to believe that the actual decision boundary between these two classes is not quite linear, and therefore LDA will have more of a bias issue. Especially using LOOCV, we can expect that the decision boundary will fit particularly well for our specific data, and as QDA will have a more flexible boundary, it should steer toward a lower bias. However, if we took the same models and introduced new data, we may want to see if perhaps QDA was too flexible and LDA may actually be the better option.