Assignment 2*

PSTAT 231

Villaseñor-Derbez J.C. | 8749749

1 Set up

1.1 Defin chunk options

```
# Default options
knitr::opts_chunk$set(echo = TRUE,
                      message = F,
                      warning = F)

# Load packages
suppressPackageStartupMessages({
  library(here)
  library(ggridges)
  library(furrr)
  library(tree)
  library(class)
  library(rpart)
  library(maptree)
  library(ROCR)
  library(MASS)
  library(magrittr)
  library(tidyverse)
})
```

1.2 Define aesthetics for graphs

1.3 Load data and standardize it

```
spam <- read_table2(file = here("raw_data", "h2", "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) # scale others
```

1.4 Functions and structures we'll need

```
# Define function for error rates
calc_error_rate <- function(predicted.value, true.value) {
  mean(true.value!=predicted.value)
}
```

*Code available on GitHUb at: https://github.com/jcvdav/PSTAT231/tree/master/docs/assig2

```r
# Create matrix that will store information about all our approaches
records = matrix(NA, nrow = 3, ncol = 2)
colnames(records) <- c("train.error","test.error")
rownames(records) <- c("knn","tree","logistic")
```

1.5 Create testing and training sets

```r
# Create random indices
set.seed(1)
#Sample 1000 random rows to use as testing set
test.indices <- sample(1:nrow(spam), 1000)

# Split the data
spam.train <- spam[-test.indices, ]
spam.test <- spam[test.indices, ]
```

1.6 Set up 10-fold CV

```r
# Number of folds
nfold <- 10
set.seed(1)

folds <- seq.int(nrow(spam.train)) %>% ## sequential obs ids
  cut(breaks = nfold, labels = F) %>% ## sequential fold ids
  sample() ## random fold ids
```

2 Training

2.1 K-nearest neighbor

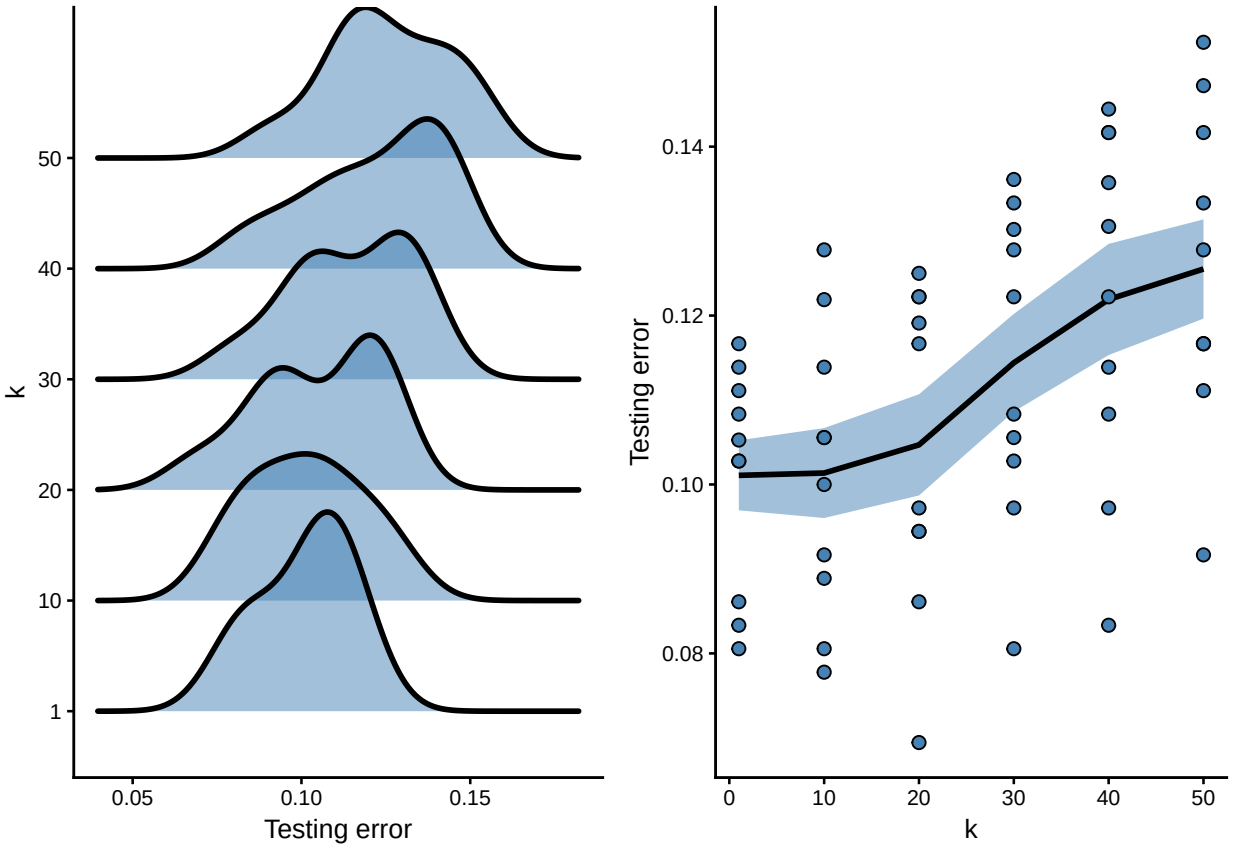
```r
do.chunk <- function(chunkid, folddef, Xdat, Ydat, k){
  # Training
  # identify rows to be used in this fold
  train <- (folddef!=chunkid)
  # Extract predictors for this fold
  Xtr <- Xdat[train, ]
  # Extract outcome variable for this fold
  Ytr <- Ydat[train]

  # Testing
  # Get predictors for testing fold
  Xvl <- Xdat[!train, ]
  # Get outcomes for testing fold
  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))
}
```

2.2 (Selecting number of neighbors) Use 10-fold cross validation to select the best number of neighbors `best.kfold` out of six values of `k` in `kvec = c(1, seq(10, 50, length.out=5))`. Use the folds defined above and use the following `do.chunk` definition in your code. Again put `set.seed(1)` before your code. What value of `k` leads to the smallest estimated test error?

```
# Set up tunning parameter options
kvec <- c(1, 10, 20, 30, 40, 50)

# Set up covariates
Xdat <- spam.train %>%
  select(-y) %>%
  as.matrix()
# Outcome variable
Ydat <- spam.train$y

# Plan multiprocess for parallel fitting
plan(multiprocess)
# Fit all
set.seed(1)
out <- expand.grid(k = kvec, chunkid = 1:nfold) %>%
  as_tibble() %>%
  mutate(fit = future_map2(chunkid, k, do.chunk, folddef = folds, Xdat = Xdat, Ydat = Ydat)) %>%
  unnest()
```

```
density_plot <- ggplot(data = out, mapping = aes(x = val.error, y = as.character(k))) +
  geom_density_ridges() +
  labs(x = "Testing error", y = "k")

scatter_plot <- ggplot(data = out, mapping = aes(x = k, y = val.error)) +
  stat_summary(geom = "ribbon", fun.data = "mean_se") +
  stat_summary(geom = "line", fun.y = "mean") +
  geom_point() +
  labs(x = "k", y = "Testing error")

cowplot::plot_grid(density_plot, scatter_plot, ncol = 2)
```




```
best.kfold <- 10
```

2.3 (Training and Test Errors) Now that the best number of neighbors has been determined, compute the training error using `spam.train` and test error using `spam.test` or the `k = best.kfold`. Use the function `calc_error_rate()` to get the errors from the predicted class labels. Fill in the first row of `records` with the train and test error from the knn fit.

```
# Matrix of training data covariates
Xtest <- spam.test %>%
  select(-y) %>%
  as.matrix()

# Traning data outcome
Ytest <- spam.test$y

set.seed(1)
## get classifications for training set
predYtr <- knn(train = Xdat, test = Xdat, cl = Ydat, k = best.kfold)

## get classifications for testing set
predYvl <- knn(train = Xdat, test = Xtest, cl = Ydat, k = best.kfold)

records[1, 1] <- calc_error_rate(predYtr, Ydat)
```

```
records[1, 2] <- calc_error_rate(predYvl, Ytest)
```

3 Decision tree

3.1 (Controlling Decision Tree Construction) Function `tree.control` specifies options for tree construction: set `minsize` equal to 5 (the minimum number of observations in each leaf) and `mindev` equal to 1e-5. See the help for `tree.control` for more information. The output of `tree.control` should be passed into `tree` function in the `control` argument. Construct a decision tree using training set `spam.train`, call the resulting tree `spamtree`. `summary(spamtree)` gives some basic information about the tree. How many leaf nodes are there? How many of the training observations are misclassified?

```
# Set up control
ctrl <- tree.control(nobs = length(Ydat), minsize = 5, mindev = 1e-5)

# Grow tree
spamtree <- tree(formula = y ~ ., data = spam.train, control = ctrl)
```

```
summary(spamtree)
```

```
##
## Classification tree:
## tree(formula = y ~ ., data = spam.train, control = ctrl)
## 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
```

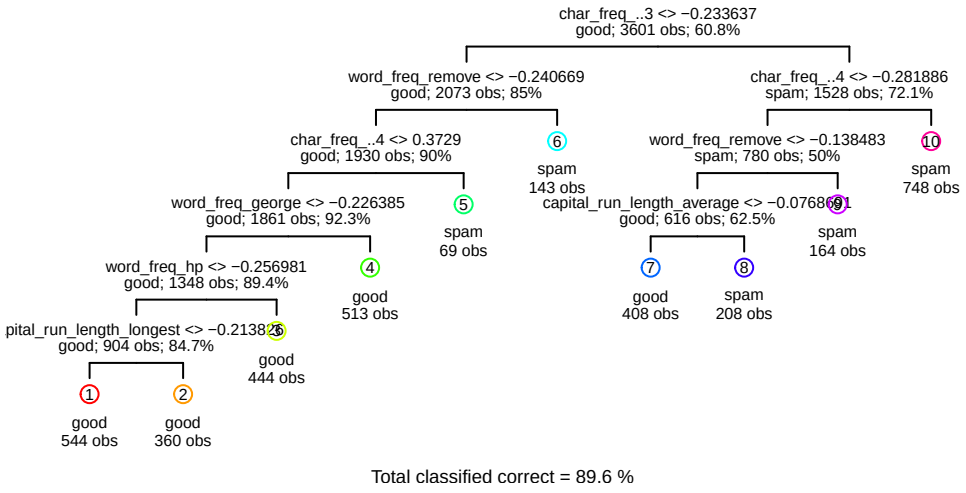
```
sum(predict(spamtree, type = "class") != Ydat)
```

```
## [1] 47
```

The tree has 184 leafs. A total of 47 observations within the training set were misclassified.

3.2 (Decision Tree Pruning) We can prune a tree using the `prune.tree` function. Pruning iteratively removes the leaves that have the least effect on the overall misclassification. Prune the tree until there are only 10 leaf nodes so that we can easily visualize the tree. Use `draw.tree` function from the `maptree` package to visualize the pruned tree. Set `nodeinfo = TRUE`.

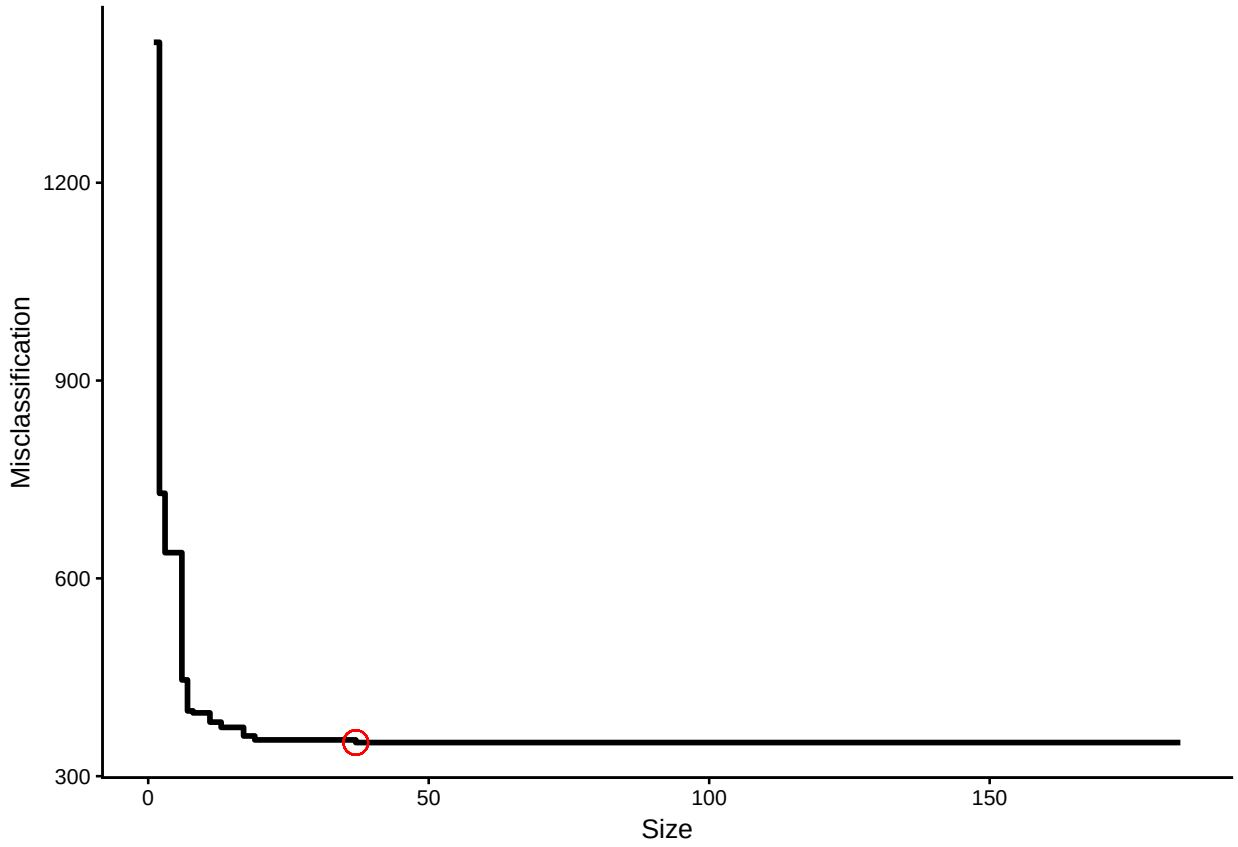
```
# Prune the tree
spamtree_pruned <- prune.tree(tree = spamtree, best = 10)

# Plot the tree
draw.tree(spamtree_pruned, nodeinfo = T, cex = 0.5)
```



3.3 **In this problem we will use cross validation to prune the tree. Fortunately, the tree package provides and easy to use function to do the cross validation for us with the `cv.tree` function. Use the same `fold` partitioning you used in the KNN problem (refer to `cv.tree` help page for detail about `rand` argument). Also be sure to set `method = misclass`. Plot the misclassification as function of tree size. Determine the optimal tree size that minimizes misclassification. Important: if there are multiple tree sizes that have the same minimum estimated misclassification, you should choose the smallest tree. This reflects the idea that we want to choose the simplest model that explains the data well ("Occam's razor"). Show the optimal tree size `best.size.cv` in the plot.**

The figure below shows misclassification error for different sizes. The red dot shows the smallest tree size for which misclassification error is minimum. The second chunck gets this value.

```
spamtree_cv <- cv.tree(object = spamtree,
                       rand = folds,
                       method = "misclass")

# default plot is ugly
spamtree_cv_tidy <- tibble(size = spamtree_cv$size,
                           misclass = spamtree_cv$dev)

ggplot(spamtree_cv_tidy, aes(x = size, y = misclass)) +
  geom_step(size = 1) +
  geom_point(x = 37, y = 351, size = 4, fill = "transparent", color = "red") +
  labs(x = "Size", y = "Misclassification")
```




```
# Obtain size that minimizes misclassification
best.size.cv <- spamtree_cv_tidy %>%
  filter(misclass <= min(misclass)) %>%
  filter(size == min(size)) %$%
  misclass
```

3.4 **(Training and Test Errors) We previously pruned the tree to a small tree so that it could be easily visualized. Now, prune the original tree to size `best.size.cv` and call the new tree `spamtree.pruned`. Calculate the training error and test error when `spamtree.pruned` is used for prediction. Use function `calc_error_rate()` to compute misclassification error. Also, fill in the second row of the matrix `records` with the training error rate and test error rate.**

```
# Prune the tree
spamtree_pruned <- prune.tree(tree = spamtree, best = 37)

predYtr <- predict(spamtree_pruned, type = "class")
predYts <- predict(spamtree_pruned, type = "class", newdata = spam.test)

records[2, 1] <- calc_error_rate(predYtr, Ydat)
records[2, 2] <- calc_error_rate(predYts, Ytest)
```

4 Logistic regression

4.1 In a binary classification problem, let p represent the probability of class label "1", which implies $1-p$ represents probability of class label "0". The logistic function (also called the "inverse logit") is the cumulative distribution function of logistic distribution, which maps a real number z to the open interval $(0,1)$:

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

4.1.1 a) Show that indeed the inverse of a logistic function is the *logit* function:

$$z(p) = ln\left(\frac{p}{1-p}\right)$$

$$\begin{aligned}
p &= \frac{e^z}{1+e^z} \\
(1+e^z)p &= \frac{e^z}{1+e^z}(1+e^z) \\
p + pe^z &= e^z \\
p + pe^z - e^z &= 0 \\
p + (p-1)e^z &= 0 \\
(p-1)e^z &= -p \\
e^z &= \frac{-p}{(p-1)} \\
ln(e^z) &= ln\left(\frac{-p}{p-)}\right) \\
z &= ln\left(\frac{p}{1-p}\right)
\end{aligned}$$

4.1.2 b) The logit function is a commonly used *link* function for a generalized linear model of binary data. One reason for this is that implies interpretable coefficients. Assume that $z = \beta_0 + \beta_1 x_1$, and $p = logistic(z)$. How does the odds of the outcome change if you increase x_1 by two? Assume β_1 is negative: what value does p approach as $x_1 \to \infty$? What value does p approach as $x_1 \to -\infty$?

If x_1 increases by two, then the odds would be $odds(p) = e^{\beta_0+\beta_1(x_1+2)}$. In other words, the odds would be multiplied by a factor of $e^2 = 7.38$.

With a negative $beta_1$, as $x_1 \to \infty$ $p \to 0$ and $x_1 \to -\infty$ $p \to 1$.

4.2 Use logistic regression to perform classification. Logistic regression specifically estimates the probability that an observation as a particular class label. We can define a probability threshold for assigning class labels based on the probabilities returned by the `glm` fit.

In this problem, we will simply use the "majority rule". If the probability is larger than 50% class as spam. Fit a logistic regression to predict spam given all other features in the dataset using the `glm` function.

Estimate the class labels using the majority rule and calculate the training and test errors. Add the training and test errors to the third row of records. Print the full `records` matrix. Which method had the lowest misclassification error on the test set?

```
logistic <- glm(formula = y ~ ., data = spam.test, family = "binomial")

predYtr <- ifelse(predict(logistic, type = "response") <= "good", "spam")
predYts <- ifelse(predict(logistic, type = "response", newdata = spam.train) <= 0.5, "good", "train")

records[3, 1] <- calc_error_rate(predYtr, Ydat)
records[3, 2] <- calc_error_rate(predYts, Ytest)
```

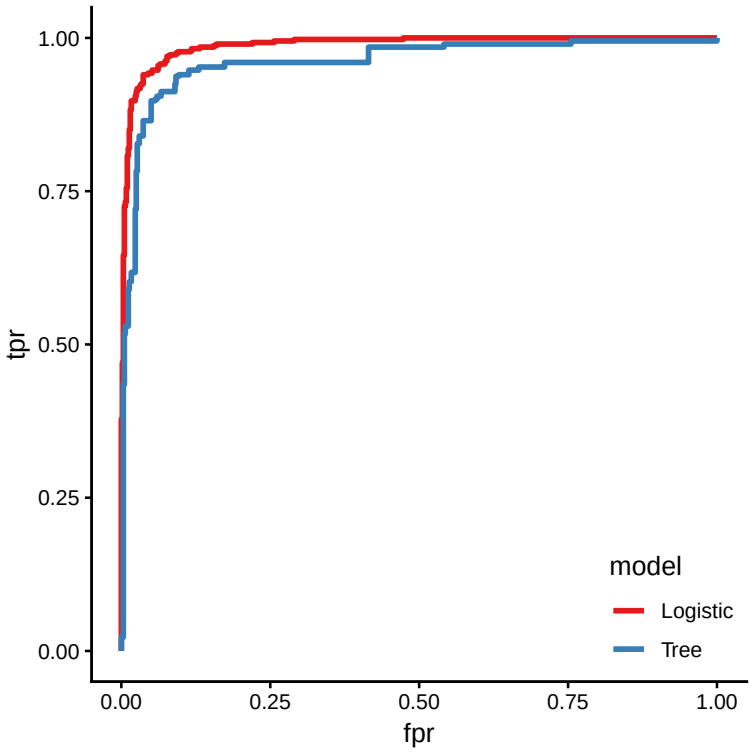
```
records
```

```
##          train.error test.error
## knn       0.08192169  0.0900000
## tree      0.05554013  0.0760000
## logistic  0.60760900  0.6423216
```

4.3 Receiver Operating Characteristic curve

4.3.1 (ROC curve) We will construct ROC curves based on the predictions of the *test* data from the model defined in `spamtree.pruned` and the logistic regression model above. Plot the ROC for the test data for both the decision tree and the logistic regression on the same plot. Compute the area under the curve for both models (AUC). Which classification method seems to perform the best by this metric?

```
tree_pred <- prediction(predict(spamtree_pruned, spam.test, type = "vector")[,2], Ytest)
tree_perf <- performance(tree_pred, "tpr", "fpr")

tree_tibble <- tibble(fpr = tree_perf@x.values[[1]],
                      tpr = tree_perf@y.values[[1]],
                      model = "Tree")

log_pred <- prediction(predict(logistic, spam.test, type = "response"), Ytest)
log_perf <- performance(log_pred, "tpr", "fpr")

log_tibble <- tibble(fpr = log_perf@x.values[[1]],
                     tpr = log_perf@y.values[[1]],
                     model = "Logistic")

rbind(tree_tibble, log_tibble) %>%
  ggplot(aes(x = fpr, y = tpr, color = model)) +
  geom_step(size = 1) +
  theme(legend.justification = c(1, 0),
        legend.position = c(1, 0)) +
  scale_color_brewer(palette = "Set1")
```




```
tree_auc <- performance(tree_pred, "auc")@y.values[[1]]
log_auc <- performance(log_pred, "auc")@y.values[[1]]
```

The tree produces an AUC = 0.9881, while the logistic fit only AUC = 0.9673. The tree has better predictive power.

4.4 10. In the SPAM example, take "positive" to mean "spam". If you are the designer of a spam filter, are you more concerned about the potential for false positive rates that are too large or true positive rates that are too small? Argue your case.

In this case I would be more worried about large false positive rates. That would imply that good emails might be flagged as spam, and I might be missing important information that I actually care about.

5 231 Problems

5.1 What is the decision threshold, M(1/2), corresponding to a probability threshold of 1/2?

5.2 (Variable Standardization and Discretization) Improve the normality of the the numerical attributes by taking the log of all chemical variables. After log transformation, impute missing values using the median method from homework 1. Note: do not take the log transform of a1 since it is not a chemical variable. Transform the variable a1 into a categorical variable with two levels: high if a1 is greater than 0.5, and low if a1 is smaller than or equal to 0.5.

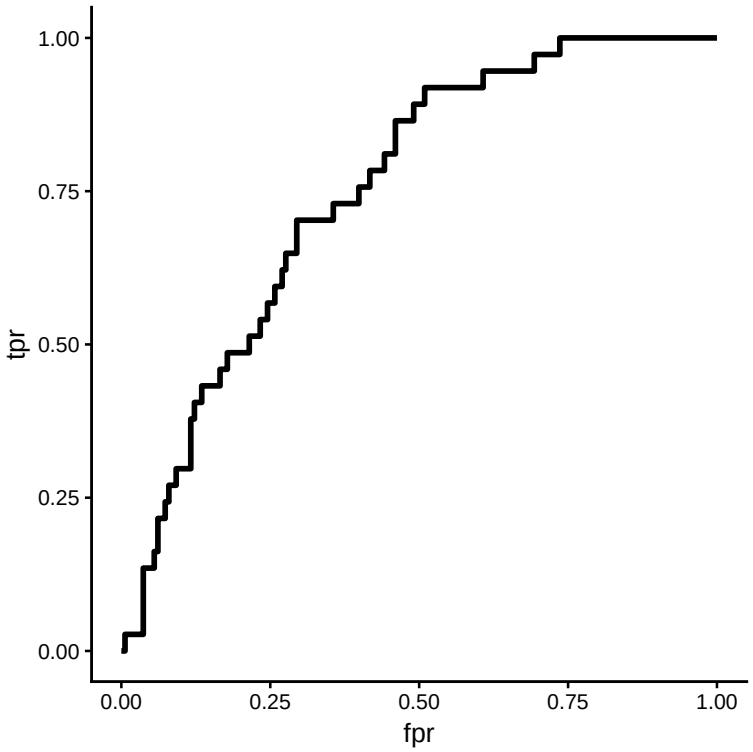
```
algae <- read_table2(here("raw_data", "h1", "algaeBloom.txt"),
                     col_names = c('season','size','speed','mxPH','mnO2','Cl','NO3','NH4',
                                   'oPO4','PO4','Chla','a1','a2','a3','a4','a5','a6','a7'),
                     na = "XXXXXXX", col_types = cols())
```

```
algae <- algae %>%
  dplyr::select(-c(a2, a3, a4, a5, a6, a7)) %>%
  mutate_at(.vars = vars(mxPH, mnO2, Cl, NO3, NH4, oPO4, PO4, Chla),
            .funs = log) %>%
  mutate_at(.vars = vars(mxPH, mnO2, Cl, NO3, NH4, oPO4, PO4, Chla),
            function(x){ifelse(is.na(x), median(x, na.rm = T), x)}) %>%
  mutate(a1 = ifelse(a1 > 0.5, "high", "low"),
         a1 = fct_relevel(a1, "low"))
```

5.3 Linear and Quadratic Discriminant Analysis

5.3.1 In LDA we assume that $\Sigma_1 = \Sigma_2$. Use LDA to predict whether a1 is high or low using the MASS::lda() function. The CV argument in the MASS::lda function uses Leave-one-out cross validation (LOOCV) when estimating the fitted values to avoid overfitting. Set the CV argument to true. Plot an ROC curve for the fitted values.

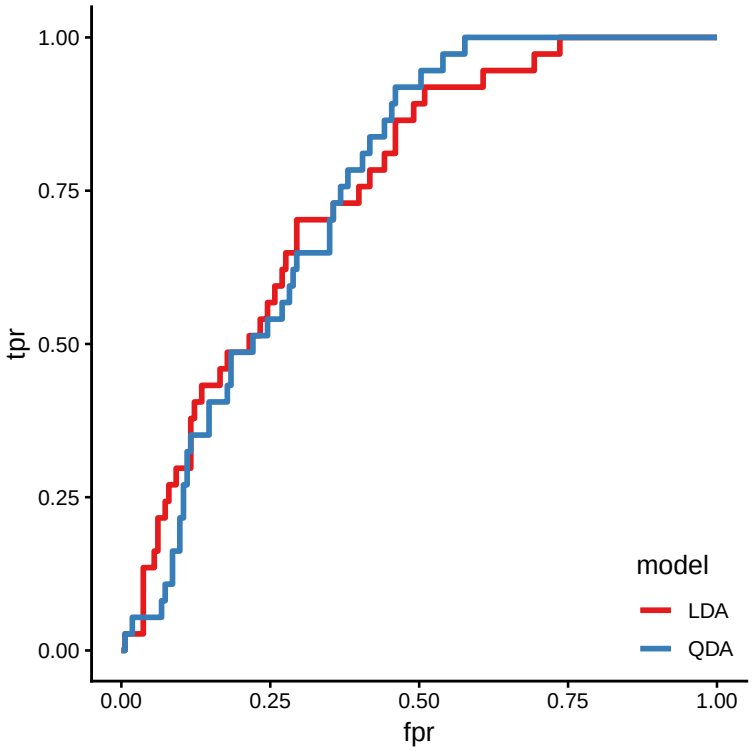
```
lda_fit <- lda(formula = a1 ~ ., data = algae, CV = T)

lda_pred <- prediction(lda_fit$posterior[,1], algae$a1)
lda_perf <- performance(lda_pred, "tpr", "fpr")

tibble(fpr = lda_perf@x.values[[1]],
       tpr = lda_perf@y.values[[1]]) %>%
  ggplot(aes(x = fpr, y = tpr)) +
  geom_step(size = 1)
```



5.4 **Quadratic discriminant analysis is strictly more flexible than LDA because it is not required that $\Sigma_1 = \Sigma_2$. In this sense, LDA can be considered a special case of QDA with the covariances constrained to be the same. Use a quadratic discriminant model to predict the a1 using the function`MASS::qda`. Again setting `CV=TRUE` and plot the ROC on the same plot as the LDA ROC. Compute the area under the ROC (AUC) for each model. To get the predicted class probabilities look at the value of posterior in the lda and qda objects. Which model has better performance? Briefly explain, in terms of the bias-variance tradeoff, why you believe the better model outperforms the worse model?**

```
qda_fit <- qda(formula = a1 ~ ., data = algae, CV = T)

qda_pred <- prediction(qda_fit$posterior[,1], algae$a1)
qda_perf <- performance(qda_pred, "tpr", "fpr")

tibble(fpr = lda_perf@x.values[[1]],
       tpr = lda_perf@y.values[[1]],
       model = "LDA") %>%
  rbind(tibble(fpr = qda_perf@x.values[[1]],
               tpr = qda_perf@y.values[[1]],
               model = "QDA")) %>%
  ggplot(aes(x = fpr, y = tpr, color = model)) +
  geom_step(size = 1)+
  theme(legend.justification = c(1, 0),
        legend.position = c(1, 0)) +
```

```
  scale_color_brewer(palette = "Set1")
```




```
lda_auc <- performance(lda_pred, "auc")@y.values[[1]]
qda_auc <- performance(qda_pred, "auc")@y.values[[1]]
```

The LDA has $AUC = 0.751$, while QDA has an AUC slightly larger, at $AUC = 0.753$. This implies that QDA is a better algorigthm at predicting. We have not split the code into testing and training sets, and perhaps this difference might be even greater when testing against a new dataset as opposed as the LOOCV done so far. Since GDA does not assume equal covariances, it might be more flexible when new data is presented.