Kevin Gong
kg2445
HW04
Stat W4240
Section 2

Homework 5

Question #1

Part a)

Best subset selection will likely have the smallest (best) training RSS since it searches all possible models while forward and backward stepwise selection will only choose from a subset of the space.

Part b)

While all three models have the possibility of overfitting, **backwards stepwise selection** will have a better chance of giving the smallest test RSS since it is the most inflexible model of the three.

Part c)

i) **True.** The best additional feature will be included in the k+1 variable model along with the k features selected by the k variable model.
ii) **True.** The k variable model contains one less feature than the k+1 variable model and also lacks the feature that would result in the smallest RSS gain.
iii) **False.** It's possible for the two models to choose resulting sets that don't overlap.
iv) **False.** It's possible for the two models to choose resulting sets that don't overlap.
v) **False.** It's possible for the two models to choose resulting sets that don't overlap.

Question #2

Part a)

Choice **iii.** The training RSS will steadily increase as s increases since the larger s value will cause more beta coefficients to become 0.

Part b)

Choice **ii**. The test RSS will initially improve due to less overfitting, but as important coefficients are removed, the test RSS goes back up.

Part c)

Choice **iv**. The variance will steadily decrease as s increases and more beta coefficients become 0.

Part d)

Choice **iii**. The squared bias will steadily increase as s increases and more beta coefficients become 0.

Part e)

None. The Bayes error rate is independent of the model.

Question #3

Part a)

Part b)

We know from the derivative of the objective function that beta1 + beta2 = 0, so beta1 must equal beta2.

Part c)

Part d)

We know that there are multiple solutions to the optimization problem since (1,2) and (1,(1) 2) are equivalent solutions.

Questions #4

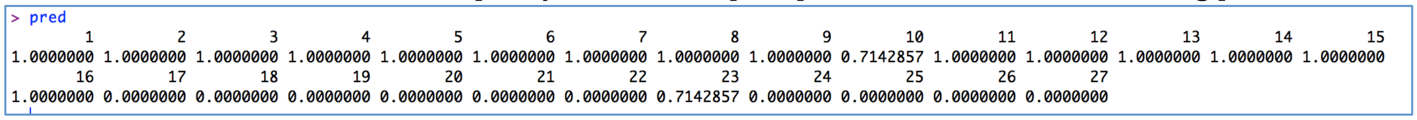
The majority vote classification will provide us with:
mean(x>0.5)>0.5

The average probability classification will provide us with:
0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75 with mean(x)>0.5

Question #5

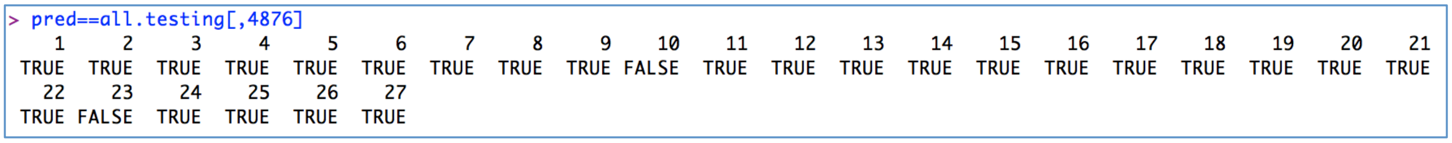
Part a)

The tree classification with Gini impurity coefficient splits provides us with the following predictions:



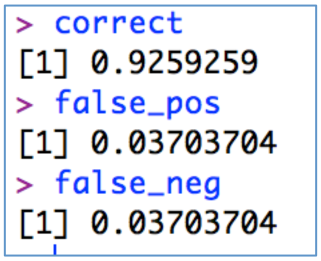
```
> pred
        1         2         3         4         5         6         7         8         9        10        11        12        13        14        15 
1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 0.7142857 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 
       16        17        18        19        20        21        22        23        24        25        26        27 
1.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.7142857 0.0000000 0.0000000 0.0000000 0.0000000 
```

We can quickly see that most of these are classified correctly:



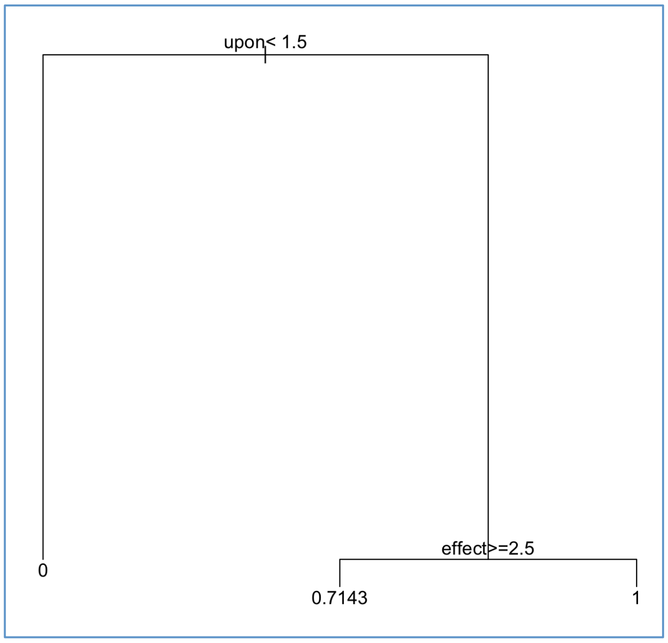
```
> pred==all.testing[,4876]
    1     2     3     4     5     6     7     8     9    10    11    12    13    14    15    16    17    18    19    20    21 
 TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE 
   22    23    24    25    26    27 
 TRUE FALSE  TRUE  TRUE  TRUE  TRUE 
```

We find our proportion classified correctly, proportion of false positives, and proportion of false negatives to be:



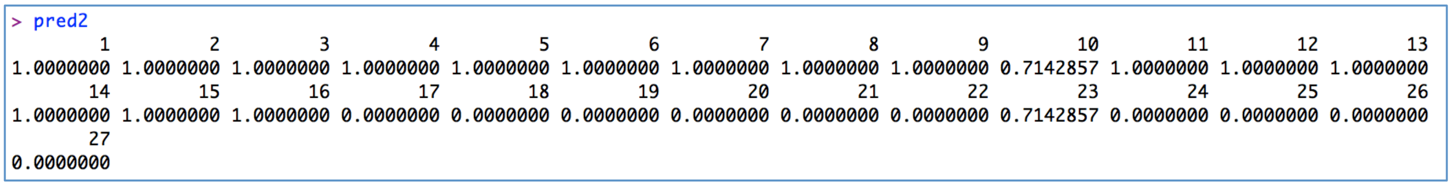
```
> correct
[1] 0.9259259
> false_pos
[1] 0.03703704
> false_neg
[1] 0.03703704
```

We also produce our classification tree with labeled splits:



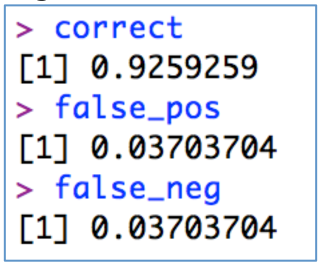
Part b)

The tree classification with information gain splits provides us with the following predictions:



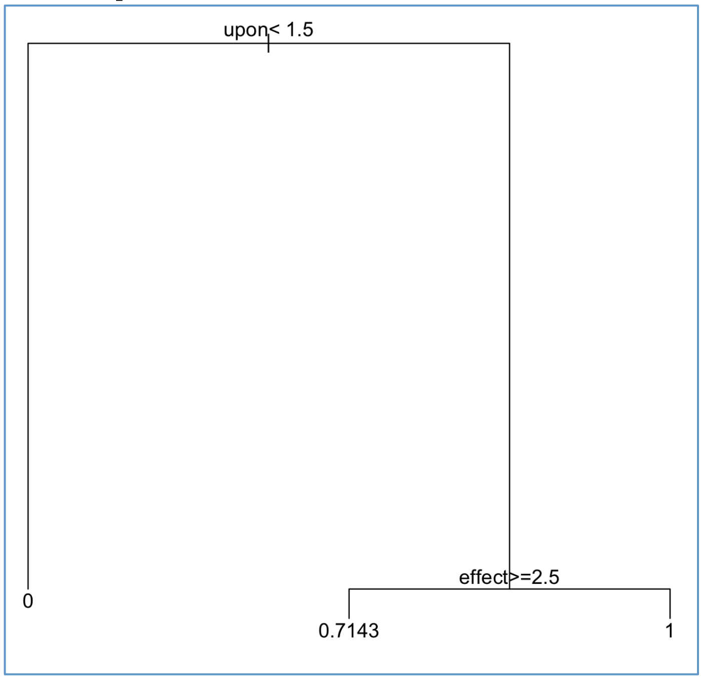
```
> pred2
        1         2         3         4         5         6         7         8         9        10        11        12        13 
1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 0.7142857 1.0000000 1.0000000 1.0000000 
       14        15        16        17        18        19        20        21        22        23        24        25        26 
1.0000000 1.0000000 1.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.7142857 0.0000000 0.0000000 0.0000000 
       27 
0.0000000 
```

We find our proportion classified correctly, proportion of false positives, and proportion of false negatives to be:



```
> correct
[1] 0.9259259
> false_pos
[1] 0.03703704
> false_neg
[1] 0.03703704
```

We also produce our classification tree with labeled splits:



As we can see, there are no differences between the two plots. This is likely because we are working with fairly simple data sets, so differences in classification are unlikely to arise despite using different methods.

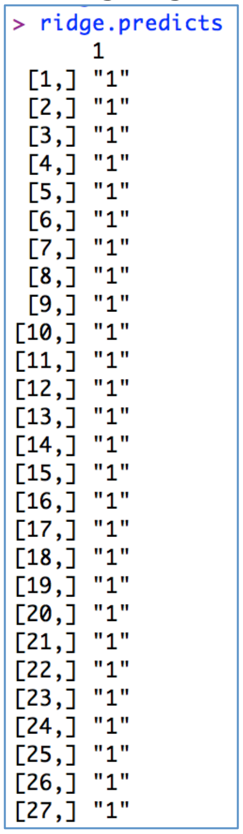
Question #6

Part a)

We cannot use an unregularized logistic regression model with this data set since the unregularized logistic regression model (lambda=0) would worsen instead of improve the classification performance.

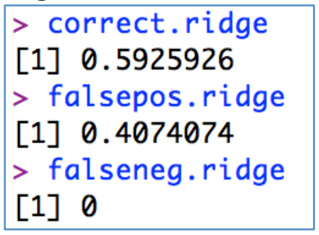
Part b)

The ridge regression model gives us the following predictions:



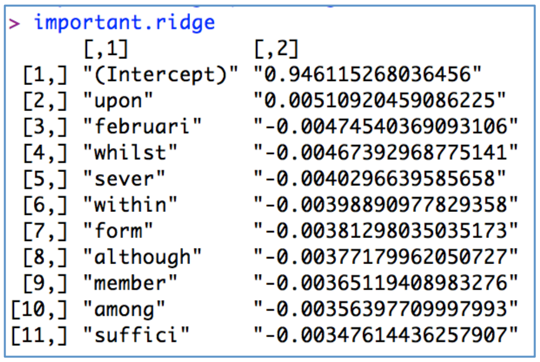
```
> ridge.predicts
      1
 [1,] "1"
 [2,] "1"
 [3,] "1"
 [4,] "1"
 [5,] "1"
 [6,] "1"
 [7,] "1"
 [8,] "1"
 [9,] "1"
[10,] "1"
[11,] "1"
[12,] "1"
[13,] "1"
[14,] "1"
[15,] "1"
[16,] "1"
[17,] "1"
[18,] "1"
[19,] "1"
[20,] "1"
[21,] "1"
[22,] "1"
[23,] "1"
[24,] "1"
[25,] "1"
[26,] "1"
[27,] "1"
```

We find our proportion classified correctly, proportion of false positives, and proportion of false negatives to be:



```
> correct.ridge
[1] 0.5925926
> falsepos.ridge
[1] 0.4074074
> falseneg.ridge
[1] 0
```

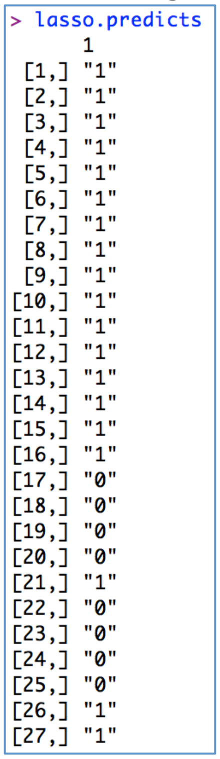
The ten most important words (ignore "Intercept" in the first row) and their corresponding coefficients are below. This was found by finding the 10 words with the largest magnitude (abs. value) of coefficients.



```
> important.ridge
      [,1]          [,2]
 [1,] "(Intercept)" "0.946115268036456"
 [2,] "upon"        "0.00510920459086225"
 [3,] "februari"    "-0.00474540369093106"
 [4,] "whilst"      "-0.00467392968775141"
 [5,] "sever"       "-0.0040296639585658"
 [6,] "within"      "-0.00398890977829358"
 [7,] "form"        "-0.00381298035035173"
 [8,] "although"    "-0.00377179962050727"
 [9,] "member"      "-0.00365119408983276"
[10,] "among"       "-0.00356397709997993"
[11,] "suffici"     "-0.00347614436257907"
```

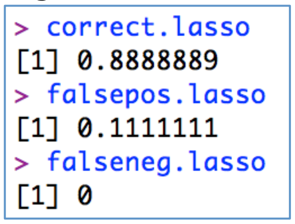
Part c)

The lasso regression model gives us the following predictions:



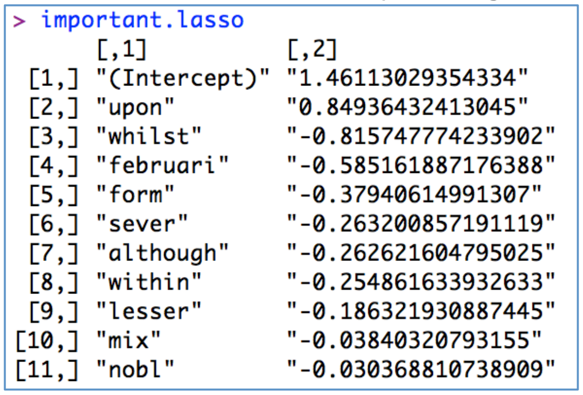
```
> lasso.predicts
      1
 [1,] "1"
 [2,] "1"
 [3,] "1"
 [4,] "1"
 [5,] "1"
 [6,] "1"
 [7,] "1"
 [8,] "1"
 [9,] "1"
[10,] "1"
[11,] "1"
[12,] "1"
[13,] "1"
[14,] "1"
[15,] "1"
[16,] "1"
[17,] "0"
[18,] "0"
[19,] "0"
[20,] "0"
[21,] "1"
[22,] "0"
[23,] "0"
[24,] "0"
[25,] "0"
[26,] "1"
[27,] "1"
```

We find our proportion classified correctly, proportion of false positives, and proportion of false negatives to be:



```
> correct.lasso
[1] 0.8888889
> falsepos.lasso
[1] 0.1111111
> falseneg.lasso
[1] 0
```

The ten most important words (ignore "Intercept" in the first row) and their corresponding coefficients are below. This was found by finding the 10 words with the largest magnitude (abs. value) of coefficients.



```
> important.lasso
      [,1]          [,2]
 [1,] "(Intercept)" "1.46113029354334"
 [2,] "upon"        "0.84936432413045"
 [3,] "whilst"      "-0.815747774233902"
 [4,] "februari"    "-0.585161887176388"
 [5,] "form"        "-0.37940614991307"
 [6,] "sever"       "-0.263200857191119"
 [7,] "although"    "-0.262621604795025"
 [8,] "within"      "-0.254861633932633"
 [9,] "lesser"      "-0.186321930887445"
[10,] "mix"         "-0.03840320793155"
[11,] "nobl"        "-0.030368810738909"
```

We see that with a few exceptions, the top 10 most popular words are fairly similar for both ridge and lasso methods, though the ordering of some words are switched around. However, we notice that the magnitude of the coefficients for the lasso method are significantly larger than those of the coefficients for the ridge method.

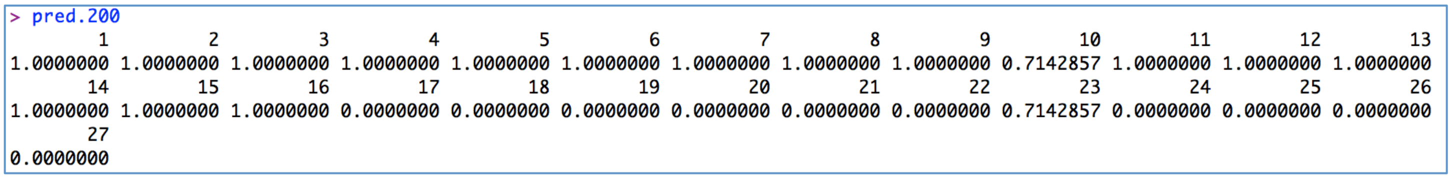
Question 7

Part a)

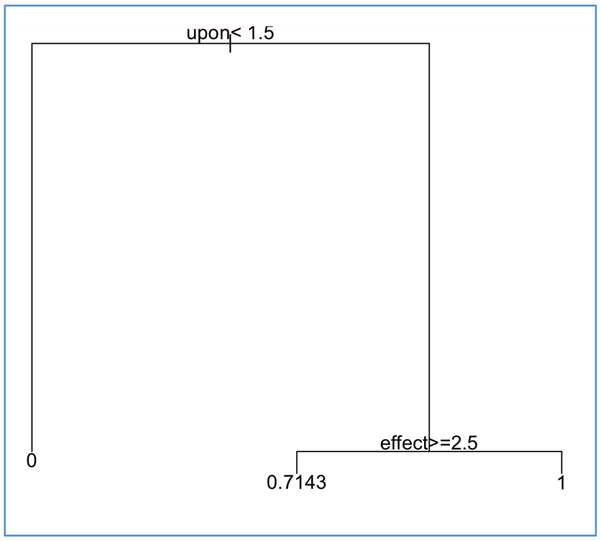
Part b)

For all 4 values of n (200, 500, 1000, and 2500), we got the following results:

Same predictions for both gini impurity and information gain splits:

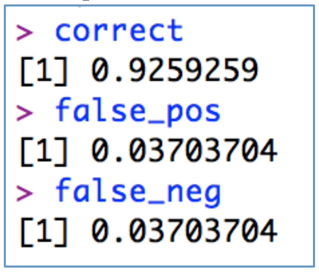


```
> pred.200
        1         2         3         4         5         6         7         8         9        10        11        12        13 
1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 0.7142857 1.0000000 1.0000000 1.0000000 
       14        15        16        17        18        19        20        21        22        23        24        25        26 
1.0000000 1.0000000 1.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.7142857 0.0000000 0.0000000 0.0000000 
       27 
0.0000000 
```



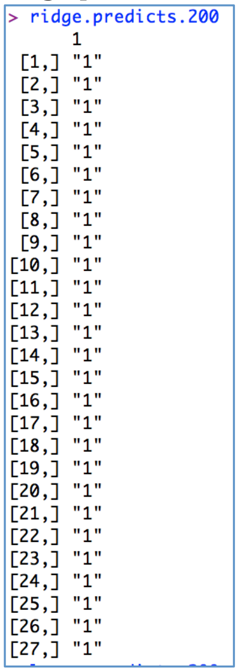
Same tree for both gini impurity and information gain splits.

Same prediction rates for both gini impurity and information gain splits:

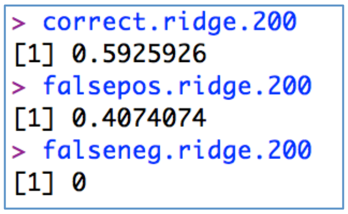


```
> correct
[1] 0.9259259
> false_pos
[1] 0.03703704
> false_neg
[1] 0.03703704
```

ridge predictions:

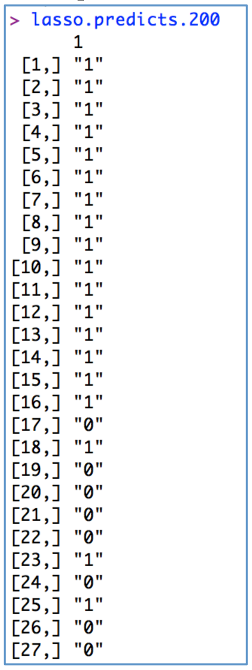


```
> ridge.predicts.200
      1
 [1,] "1"
 [2,] "1"
 [3,] "1"
 [4,] "1"
 [5,] "1"
 [6,] "1"
 [7,] "1"
 [8,] "1"
 [9,] "1"
[10,] "1"
[11,] "1"
[12,] "1"
[13,] "1"
[14,] "1"
[15,] "1"
[16,] "1"
[17,] "1"
[18,] "1"
[19,] "1"
[20,] "1"
[21,] "1"
[22,] "1"
[23,] "1"
[24,] "1"
[25,] "1"
[26,] "1"
[27,] "1"
```

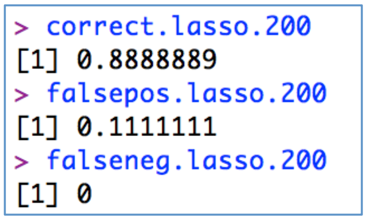


```
> correct.ridge.200
[1] 0.5925926
> falsepos.ridge.200
[1] 0.4074074
> falseneg.ridge.200
[1] 0
```

lasso predictions:



```
> lasso.predicts.200
      1
 [1,] "1"
 [2,] "1"
 [3,] "1"
 [4,] "1"
 [5,] "1"
 [6,] "1"
 [7,] "1"
 [8,] "1"
 [9,] "1"
[10,] "1"
[11,] "1"
[12,] "1"
[13,] "1"
[14,] "1"
[15,] "1"
[16,] "1"
[17,] "0"
[18,] "1"
[19,] "0"
[20,] "0"
[21,] "0"
[22,] "0"
[23,] "1"
[24,] "0"
[25,] "1"
[26,] "0"
[27,] "0"
```



```
> correct.lasso.200
[1] 0.8888889
> falsepos.lasso.200
[1] 0.1111111
> falseneg.lasso.200
[1] 0
```

Again, these results were the same for all 4 values of n. This likely happens because we're working with fairly simple data sets, so differences in classification are unlikely to arise despite using different subsets of the dictionary.

Code

```
#############################
# Kevin Gong
# STAT W4240
# Homework 045
# 4/16/14
#

#############################

#################
# Setup
#################

# make sure R is in the proper working directory
# note that this will be a different path for every machine
setwd("~/Dropbox/SIPA/Data Mining/hw05")


# first include the relevant libraries
# note that a loading error might mean that you have to
# install the package into your R distribution.
# Use the package installer and be sure to install all dependencies
library(tm)
library(SnowballC)
library(rpart)
library(glmnet)

#################
# Problem 5a
#################

##########################################
# Preprocess the data
# You should have made directiories in
# hw04 for cleaned data; make sure these
# are in your path
##########################################

##########################################
# To read in data from the directories:
# Partially based on code from C. Shalizi
read.directory <- function(dirname) {
```

```r
    # Store the infiles in a list
    infiles = list();
    # Get a list of filenames in the directory
    filenames = dir(dirname,full.names=TRUE);
    for (i in 1:length(filenames)){
        infiles[[i]] = scan(filenames[i],what="",quiet=TRUE);
         }
    return(infiles)
}
hamilton.train <- read.directory('fp_hamilton_train_clean')
hamilton.test <- read.directory('fp_hamilton_test_clean')
madison.train <- read.directory('fp_madison_train_clean')
madison.test <- read.directory('fp_madison_test_clean')
##########################################

##########################################
# Make dictionary sorted by number of times a word appears in corpus
# (useful for using commonly appearing words as factors)
# NOTE: Use the *entire* corpus: training, testing, spam and ham
make.sorted.dictionary.df <- function(infiles){
    # This returns a dataframe that is sorted by the number of times
    # a word appears

    # List of vectors to one big vetor
    dictionary.full <- unlist(infiles)
    # Tabulates the full dictionary
    tabulate.dic <- tabulate(factor(dictionary.full))
    # Find unique values
    dictionary <- unique(dictionary.full)
    # Sort them alphabetically
    dictionary <- sort(dictionary)
    dictionary.df <- data.frame(word = dictionary, count = tabulate.dic)
    sort.dictionary.df <-
 dictionary.df[order(dictionary.df$count,decreasing=TRUE),];
    return(sort.dictionary.df)
}
dictionary <-
 make.sorted.dictionary.df(c(hamilton.train,hamilton.test,madison.train,madison.te
 st))

head(dictionary)

##########################################
```

```
##########################################
# Make a document-term matrix, which counts the number of times each
# dictionary element is used in a document
make.document.term.matrix <- function(infiles,dictionary){
    # This takes the text and dictionary objects from above and outputs a
    # document term matrix
    num.infiles <- length(infiles);
    num.words <- nrow(dictionary);
    # Instantiate a matrix where rows are documents and columns are words
    dtm <- mat.or.vec(num.infiles,num.words); # A matrix filled with zeros
    for (i in 1:num.infiles){
        num.words.infile <- length(infiles[[i]]);
        infile.temp <- infiles[[i]];
        for (j in 1:num.words.infile){
            ind <- which(dictionary == infile.temp[j])[[1]];
            # print(sprintf('%s,%s', i , ind))
            dtm[i,ind] <- dtm[i,ind] + 1;
            #print(c(i,j))
        }
    }
return(dtm);
}

dtm.hamilton.train <- make.document.term.matrix(hamilton.train,dictionary)
dtm.hamilton.test <- make.document.term.matrix(hamilton.test,dictionary)
dtm.madison.train <- make.document.term.matrix(madison.train,dictionary)
dtm.madison.test <- make.document.term.matrix(madison.test,dictionary)
##########################################

##########################################

head(dtm.hamilton.train)
dim(dtm.hamilton.test)
dim(dtm.madison.train)
dim(dtm.madison.test)

vector(1,35)
sapply(1:35,character(1))

class_htrain=matrix(nrow=35, ncol=1)
class_htrain[,1]=1
class_htrain
htrain <-cbind(dtm.hamilton.train,class_htrain)
```

```
class_htest=matrix(nrow=16, ncol=1)
class_htest[,1]=1
class_htest
htest <-cbind(dtm.hamilton.test,class_htest)

class_mtrain=matrix(nrow=15, ncol=1)
class_mtrain[,1]=0
class_mtrain
mtrain <-cbind(dtm.madison.train,class_mtrain)

class_mtest=matrix(nrow=11, ncol=1)
class_mtest[,1]=0
class_mtest
mtest <-cbind(dtm.madison.test,class_mtest)

all.training = rbind(htrain,mtrain)
all.testing = rbind(htest,mtest)

all.training[,4876]

colnames(all.training) = c(as.vector(dictionary$word),"y")
colnames(all.testing) = c(as.vector(dictionary$word),"y")

colnames(all.training)

all.training.frame = as.data.frame(all.training)
all.testing.frame = as.data.frame(all.testing)



tree.gini = rpart(all.training.frame$y~.,data=all.training.frame)
pred = predict(tree.gini,all.testing.frame)
pred
pred==all.testing[,4876]

correct = sum(pred==all.testing[,4876])/length(pred)
correct
false_pos = sum(pred>all.testing[,4876] & !pred==all.testing[,4876])/length(pred)
false_pos
false_neg = sum(pred<all.testing[,4876] & !pred==all.testing[,4876])/length(pred)
false_neg


correct
```

```
false_pos
false_neg

par(xpd=TRUE)
plot(tree.gini)
text(tree.gini)


##########################################

#################
# Problem 5b
#################

##########################################

tree.info =
 rpart(all.training.frame$y~.,data=all.training.frame,parms=list(split="informatio
 n"))
pred2 = predict(tree.info,all.testing.frame)
!pred2==all.testing[,4876]
pred2

correct = sum(pred2==all.testing[,4876])/length(pred2)
correct
false_pos = sum(pred2>all.testing[,4876] &
 !pred2==all.testing[,4876])/length(pred2)
false_pos
false_neg = sum(pred2<all.testing[,4876] &
 !pred2==all.testing[,4876])/length(pred2)
false_neg

correct
false_pos
false_neg

par(xpd=TRUE)
plot(tree.info)
text(tree.info)



##########################################
```

```r
#################
# Problem 6b
#################

##########################################
dtm.hamilton.train.n <- scale(dtm.hamilton.train,center=TRUE, scale=TRUE)
dtm.hamilton.test.n <- scale(dtm.hamilton.test,center=TRUE, scale=TRUE)
dtm.madison.train.n <- scale(dtm.madison.train,center=TRUE, scale=TRUE)
dtm.madison.test.n <- scale(dtm.madison.test,center=TRUE, scale=TRUE)

dim(dtm.hamilton.train.n)
head(dtm.hamilton.train.n)

dtm.hamilton.train.n[is.nan(dtm.hamilton.train.n)]=0
dtm.hamilton.test.n[is.nan(dtm.hamilton.test.n)]=0
dtm.madison.train.n[is.nan(dtm.madison.train.n)]=0
dtm.madison.test.n[is.nan(dtm.madison.test.n)]=0

all.training.s <- scale(all.training[,1:4875],center=TRUE, scale=TRUE)
all.testing.s <- scale(all.testing[,1:4875],center=TRUE, scale=TRUE)
head(all.testing.s)


all.training.s[is.nan(all.training.s)]=0
all.testing.s[is.nan(all.testing.s)]=0



class_htrain.n=matrix(nrow=35, ncol=1)
class_htrain.n[,1]=1
class_htrain.n
htrain.n <-cbind(dtm.hamilton.train.n,class_htrain.n)

class_htest.n=matrix(nrow=16, ncol=1)
class_htest.n[,1]=1
class_htest.n
htest.n <-cbind(dtm.hamilton.test.n,class_htest.n)

class_mtrain.n=matrix(nrow=15, ncol=1)
class_mtrain.n[,1]=0
class_mtrain.n
mtrain.n <-cbind(dtm.madison.train.n,class_mtrain.n)

class_mtest.n=matrix(nrow=11, ncol=1)
class_mtest.n[,1]=0
class_mtest.n
mtest.n <-cbind(dtm.madison.test.n,class_mtest.n)

all.training.n = rbind(htrain.n,mtrain.n)
all.testing.n = rbind(htest.n,mtest.n)

all.training.n[,3]

colnames(all.training.n) = c(as.vector(dictionary$word),"y")
colnames(all.testing.n) = c(as.vector(dictionary$word),"y")

colnames(all.training.n)

#all.training.frame.n = as.data.frame(all.training.n)
#all.testing.frame.n = as.data.frame(all.testing.n)

help(glmnet)
cv.glmnet()

ridge.train <-
 cv.glmnet(all.training.s[,1:4875],all.training.n[,4876],family="binomial",alpha=0
 )

ridge.train.betas <-
 glmnet(all.training.n[,1:4875],all.training.n[,4876],alpha=1,lambda=ridge.train$l
 ambda.min,family="binomial")


head(coef(ridge.train))
#s = "lambda.min"

ridge.train
ridge.train$lambda
ridge.train.betas

ridge.train[is.nan(ridge.train)]=0


ridge.predicts <- predict(ridge.train,all.testing.s[,1:4875],s =
 "lambda.min",type="class")
ridge.predicts
```

```
as.vector(ridge.predicts)==all.testing.n[,4876]

correct.ridge = sum(as.vector(ridge.predicts)==all.testing.n[,4876])/27
correct.ridge
falsepos.ridge = sum(as.vector(ridge.predicts)>all.testing.n[,4876] &
 !as.vector(ridge.predicts)==all.testing[,4876])/27
falsepos.ridge
falseneg.ridge = sum(as.vector(ridge.predicts)<all.testing.n[,4876] &
 !as.vector(ridge.predicts)==all.testing[,4876])/27
falseneg.ridge


correct.ridge
falsepos.ridge
falseneg.ridge





ridge.coefs <- coef(ridge.train)
ridge.coefs.v <- as.vector(ridge.coefs)
which.max[ridge.coefs.v[2:4875]
ridge.coefs.v[order(ridge.coefs.v)[1:10]]

order(ridge.coefs.v)

which(tail(sort(ridge.coefs),11))

rownames(ridge.coefs)[order(abs(ridge.coefs), decreasing=TRUE)][1:11]
ridge.coefs["upon",]


important.ridge = matrix(nrow=11,ncol=2)
important.ridge[,1]= rownames(ridge.coefs)[order(abs(ridge.coefs),
 decreasing=TRUE)][1:11]
important.ridge[,2]=
 ridge.coefs.v[order(abs(ridge.coefs.v),decreasing=TRUE)[1:11]]
important.ridge



##########################################

#################
```

```
# Problem 6c
#################

##########################################
lasso.train <-
 cv.glmnet(all.training.s[,1:4875],all.training.n[,4876],family="binomial",alpha=1
 )

#ridge.train.betas <-
 glmnet(all.training.n[,1:4875],all.training.n[,4876],alpha=1,lambda=ridge.train$l
 ambda.min,family="binomial")




sort(ridge.coefs)

ridge.coefs[5,]

ridge.coefs[432]

#s = "lambda.min"

lasso.train
ridge.train$lambda
ridge.train.betas

ridge.train[is.nan(ridge.train)]=0


lasso.predicts <- predict(lasso.train,all.testing.s[,1:4875],s =
 "lambda.min",type="class")
lasso.predicts

as.vector(ridge.predicts)==all.testing.n[,4876]

correct.lasso = sum(as.vector(lasso.predicts)==all.testing.n[,4876])/27
correct.lasso


correct.lasso = sum(as.vector(lasso.predicts)==all.testing.n[,4876])/27
correct.lasso
falsepos.lasso = sum(as.vector(lasso.predicts)>all.testing.n[,4876] &
 !as.vector(lasso.predicts)==all.testing[,4876])/27
falsepos.lasso
falseneg.lasso = sum(as.vector(lasso.predicts)<all.testing.n[,4876] &
 !as.vector(lasso.predicts)==all.testing[,4876])/27
falseneg.lasso


correct.lasso
falsepos.lasso
falseneg.lasso


coef(lasso.train)
lasso.coefs <- coef(lasso.train)
lasso.coefs.v <- as.vector(lasso.coefs)
which.max[lasso.coefs.v[2:4875]]
lasso.coefs.v[order(lasso.coefs.v)[1:10]]
lasso.coefs.v

lasso.coefs.v[order(abs(lasso.coefs.v),decreasing=TRUE)[1:11]]

tail(sort(abs(lasso.coefs),decreasing=TRUE),11)


rownames(lasso.coefs)[order(abs(lasso.coefs), decreasing=TRUE)][1:11]

order(lasso.coefs, decreasing=TRUE)

important.lasso = matrix(nrow=11,ncol=2)
important.lasso[,1]= rownames(lasso.coefs)[order(abs(lasso.coefs),
 decreasing=TRUE)][1:11]
important.lasso[,2]=
 lasso.coefs.v[order(abs(lasso.coefs.v),decreasing=TRUE)[1:11]]
important.lasso


############################################

#################
# Problem 7b
#################

############################################
make.log.pvec <- function(dtm,mu){
    # Sum up the number of instances per word
    pvec.no.mu <- colSums(dtm)
    # Sum up number of words
    n.words <- sum(pvec.no.mu)
    # Get dictionary size
    dic.len <- length(pvec.no.mu)
    # Incorporate mu and normalize
    log.pvec <- log(pvec.no.mu + mu) - log(mu*dic.len + n.words)
    return(log.pvec)
}

#set our mu value
mu=1/4875

#generate the log probabilities for the Hamilton and Madison training and testing
 sets
logp.all.training <- make.log.pvec(all.training[,1:4875],mu)
logp.all.testing <- make.log.pvec(all.testing[,1:4875],mu)

#double check ranges to make sure they roughly between -2 to -25
range(logp.all.training)
range(logp.all.testing)

logp.all.training.m <- as.matrix(logp.all.training)
logp.all.testing.m <- as.matrix(logp.all.testing)

ranked.words <- rownames(logp.all.training.m)[order(logp.all.training.m,
 decreasing=TRUE)]
ranked.words[1:]
ranked.words
all.training[,ranked.words[1:200]]

all.training.s[,ranked.words[1]]




ranked.words[1:20]

all.training[,ranked.words[1:20]]


all.training.frame
```

```
tree.gini.200 =
 rpart(all.training.frame$y~.,data=all.training.frame[,ranked.words[1:200]])
pred.200 = predict(tree.gini.200,all.testing.frame[,ranked.words[1:200]])
pred.200
!pred.200==all.testing[,4876]

correct = sum(pred.200==all.testing[,4876])/length(pred)
correct
false_pos = sum(pred.200>all.testing[,4876] &
 !pred.200==all.testing[,4876])/length(pred)
false_pos
false_neg = sum(pred.200<all.testing[,4876] &
 !pred.200==all.testing[,4876])/length(pred)
false_neg

correct
false_pos
false_neg

plot(tree.gini.200)
text(tree.gini.200)




tree.info.200 =
 rpart(all.training.frame$y~.,data=all.training.frame[,ranked.words[1:200]],parms=
 list(split="information"))
pred.200 = predict(tree.info.200,all.testing.frame[,ranked.words[1:200]])
pred.200
!pred.200==all.testing[,4876]


correct = sum(pred.200==all.testing[,4876])/length(pred2)
correct
false_pos = sum(pred.200>all.testing[,4876] &
 !pred.200==all.testing[,4876])/length(pred)
false_pos
false_neg = sum(pred.200<all.testing[,4876] &
 !pred.200==all.testing[,4876])/length(pred)
false_neg
```

```r
plot(tree.info.200)
text(tree.info.200)




ridge.train.200 <-
 cv.glmnet(all.training.s[,ranked.words[1:200]],all.training.n[,4876],family="bino
 mial",alpha=0)

ridge.predicts.200 <-
 predict(ridge.train.200,all.testing.s[,ranked.words[1:200]],s =
 "lambda.min",type="class")
ridge.predicts.200


correct.ridge.200 = sum(as.vector(ridge.predicts.200)==all.testing.n[,4876])/27
correct.ridge.200
falsepos.ridge.200 = sum(as.vector(ridge.predicts.200)>all.testing.n[,4876] &
 !as.vector(ridge.predicts.200)==all.testing[,4876])/27
falsepos.ridge.200
falseneg.ridge.200 = sum(as.vector(ridge.predicts.200)<all.testing.n[,4876] &
 !as.vector(ridge.predicts.200)==all.testing[,4876])/27
falseneg.ridge.200


correct.ridge.200
falsepos.ridge.200
falseneg.ridge.200




lasso.train.200 <-
 cv.glmnet(all.training.s[,ranked.words[1:200]],all.training.n[,4876],family="bino
 mial",alpha=1)
lasso.predicts.200 <-
 predict(lasso.train.200,all.testing.s[,ranked.words[1:200]],s =
 "lambda.min",type="class")
lasso.predicts.200

correct.lasso.200 = sum(as.vector(lasso.predicts.200)==all.testing.n[,4876])/27
correct.lasso.200
falsepos.lasso.200 = sum(as.vector(lasso.predicts.200)>all.testing.n[,4876] &
 !as.vector(lasso.predicts.200)==all.testing[,4876])/27
falsepos.lasso.200
falseneg.lasso.200 = sum(as.vector(lasso.predicts.200)<all.testing.n[,4876] &
 !as.vector(lasso.predicts.200)==all.testing[,4876])/27
falseneg.lasso.200


correct.lasso.200
falsepos.lasso.200
falseneg.lasso.200


##########################################

#################
# End of Script
#################
```