Kevin Gong
kg2445
HW04
Stat W4240
Section 2

Homework 4

Problem #1

Part a)

(see code)

We process the text and generate cleaned documents for 4 categories: Hamilton training, Hamilton testing, Madison training, and Madison testing.

Part b)

(see code)

We next read in all four cleaned infile directories into four variable names.

Part c)

(see code)

Our dictionary for the full dataset contains 4875 unique words (length 4875).

Part d)

(see code)

We create document term lists which have the following dimensions:

```
> dim(dtm.hamilton.train)
[1]   35 4875
> dim(dtm.hamilton.test)
[1]   16 4875
> dim(dtm.madison.train)
[1]   15 4875
> dim(dtm.madison.test)
[1]   11 4875
```

Part e)

(see code)

We calculate the log probability vectors for all document term matrices, giving us the following log probability values:

```
> range(logp.hamilton.train)
[1] -18.994804  -4.060388
> range(logp.hamilton.test)
[1] -18.158564  -3.861553
> range(logp.madison.train)
[1] -18.263944  -3.696722
> range(logp.madison.test)
[1] -17.87911  -3.85384
```

Question 2

(see code)

While our actual code includes other factors, on a simple level, our Naive Bayes classifier classifies documents by comparing the log probability of a document being written by Madison versus the log probability of a document being written by Hamilton. Whichever log probability is less negative (smaller in absolute value) is thus deemed the author of a particular document. Our classifier function includes additional considerations such as the probability that each word in a particular document is more likely to have been used by Hamilton or Madison.

Question 3

After setting our mu to 1/4875, Hamilton log prior to log(35/50), and Madison log prior to log(15/50), we find that our NB classifier correctly classified 17 of 27 test papers, or approximately 63%.

Interestingly, 26 of our 27 test files were classified as Hamilton, while only 1 file was (correctly) classified as Madison. In the screenshot below, “ham” contains the documents actually written by Hamilton with their estimated classification from Naïve Bayes, while “mad” contains the documents actually written by Madison with their estimated classification from Naïve Bayes.

```
> ham
   log.prob.hamilton log.prob.madison classifier
1          -5884.051        -6278.291   Hamilton
2          -9155.595        -9648.470   Hamilton
3          -8660.545        -9262.260   Hamilton
4          -7979.130        -8595.673   Hamilton
5          -7475.046        -7714.311   Hamilton
6         -10080.582       -11010.827   Hamilton
7          -8783.809        -9436.455   Hamilton
8          -3598.842        -3756.550   Hamilton
9         -11348.577       -11838.505   Hamilton
10         -7226.877        -7786.749   Hamilton
11         -5909.684        -6175.786   Hamilton
12         -7137.954        -7516.445   Hamilton
13        -13205.356       -14056.711   Hamilton
14         -6126.025        -6459.105   Hamilton
15         -7438.340        -7866.163   Hamilton
16         -5084.279        -5199.421   Hamilton
> mad
   log.prob.hamilton log.prob.madison classifier
1         -10941.274       -11259.041   Hamilton
2          -8433.735        -8459.979   Hamilton
3         -13177.082       -13666.703   Hamilton
4          -6483.486        -6556.564   Hamilton
5          -6537.333        -6635.823   Hamilton
6          -7690.875        -7709.281   Hamilton
7          -6587.484        -6410.838    Madison
8          -7257.264        -7376.371   Hamilton
9          -7680.397        -7898.583   Hamilton
10         -7180.862        -7313.647   Hamilton
11         -8491.572        -8773.368   Hamilton
```

We find the following true positives, true negatives, false positives, and false negatives:

```
> true_pos
[1] 1
> true_neg
[1] 0.09090909
> false_pos
[1] 0.9090909
> false_neg
[1] 0
```

As expected from our classification results, our true positive is 1 (or false negative is 0) since we correctly classified all documents actually written by Hamilton. Meanwhile, our performance on the Madison documents was much poorer, only correctly classifying 1 out of 11 Madison documents as written by Madison.

Question 4

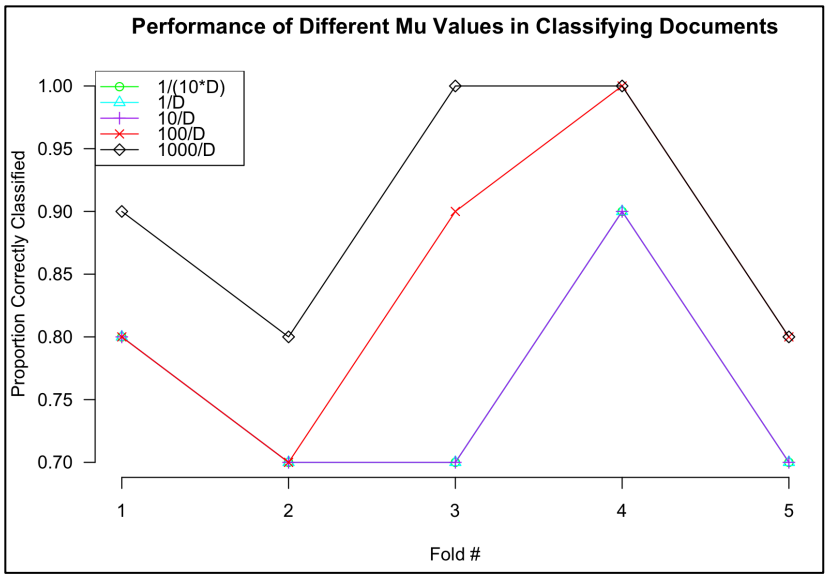
Part a)

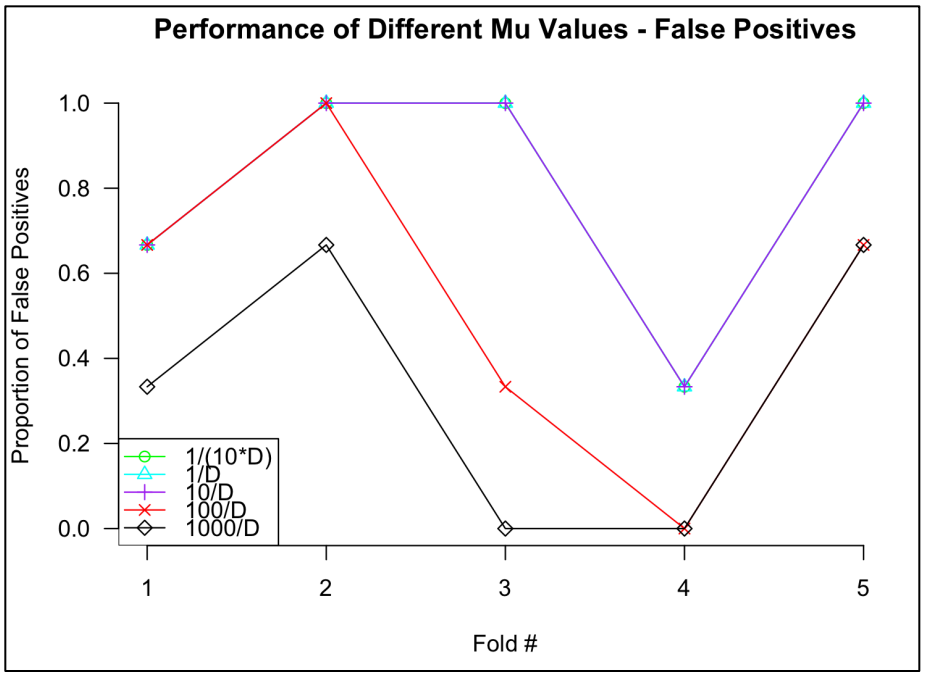
We have *a lot* of code for this part, but the basic steps boil down to:

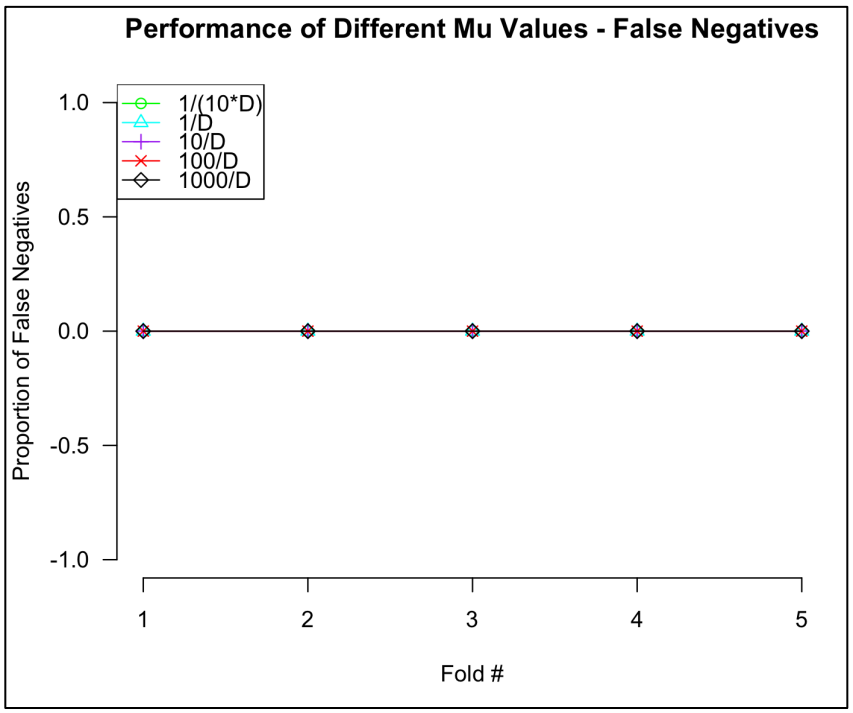
1) Create 5 folds of 10 randomly sampled documents each with 7 Hamilton and 3 Madison documents.
2) Generate document term matrices for Hamilton and Madison training and testing using the 5 folds from step 1.
3) Create 3 empty matrices to store proportion correctly classified, false positives, and false negatives, while also putting our different mu values into a vector.
4) Run giant for loop to perform the Naïve Bayes classification across multiple mu values and folds.
5) Graph the results.

Our data table and graphs are below:

```
> correct
          Fold 1 Fold 2 Fold3 Fold 4 Fold 5
1/(10*D)    0.8    0.7   0.7    0.9    0.7
1/D         0.8    0.7   0.7    0.9    0.7
10/D        0.8    0.7   0.7    0.9    0.7
100/D       0.8    0.7   0.9    1.0    0.8
1000/D      0.9    0.8   1.0    1.0    0.8
> false_pos_final
           Fold 1    Fold 2    Fold3    Fold 4    Fold 5
1/(10*D) 0.6666667 1.0000000 1.0000000 0.3333333 1.0000000
1/D      0.6666667 1.0000000 1.0000000 0.3333333 1.0000000
10/D     0.6666667 1.0000000 1.0000000 0.3333333 1.0000000
100/D    0.6666667 1.0000000 0.3333333 0.0000000 0.6666667
1000/D   0.3333333 0.6666667 0.0000000 0.0000000 0.6666667
> false_neg_final
          Fold 1 Fold 2 Fold3 Fold 4 Fold 5
1/(10*D)      0      0     0      0      0
1/D           0      0     0      0      0
10/D          0      0     0      0      0
100/D         0      0     0      0      0
1000/D        0      0     0      0      0
```







Part b)

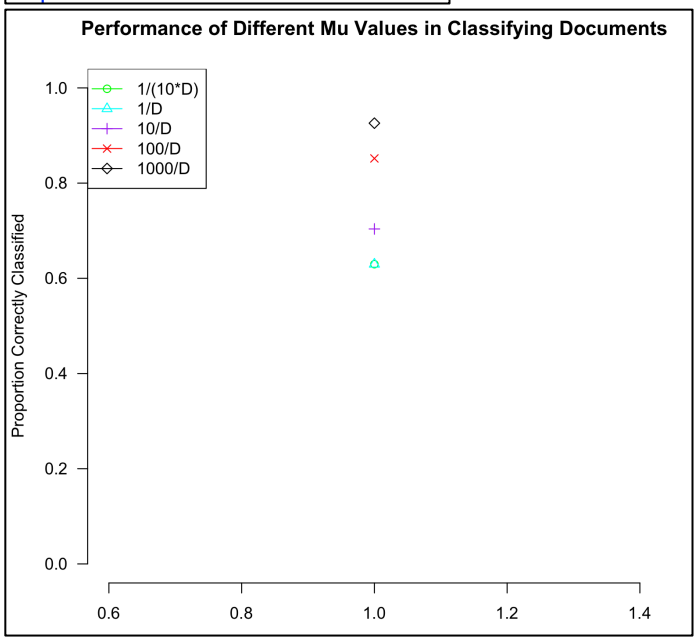
From our results from part a, it seems that the best value for mu is 1000/D because this mu value consistently has the highest percentage of correctly classified documents as well as the lowest rate of false positives. Since we ultimately care about the ability of our Naïve Bayes classifier to correctly classify documents, the

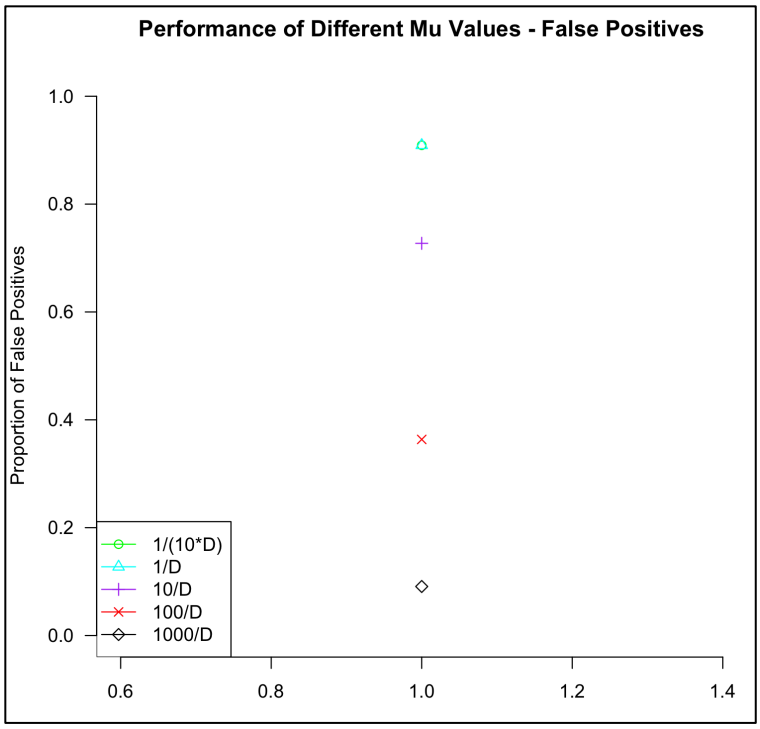
mu value of 1000/D is clearly the best since it has the highest correct classification rate of the five values.

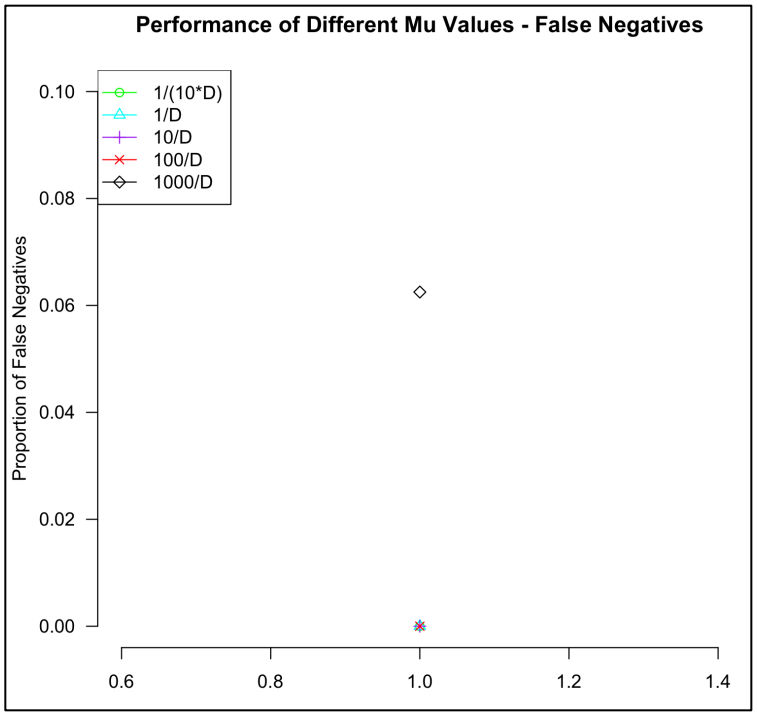
Part c)

We use each value of μ to train on the full training set and test on the full testing set, giving us the results below:

```
> complete_correct
              [,1]
1/(10*D) 0.6296296
1/D      0.6296296
10/D     0.7037037
100/D    0.8518519
1000/D   0.9259259
> complete_false_pos_final
              [,1]
1/(10*D) 0.90909091
1/D      0.90909091
10/D     0.72727273
100/D    0.36363636
1000/D   0.09090909
> complete_false_neg_final
             [,1]
1/(10*D) 0.0000
1/D      0.0000
10/D     0.0000
100/D    0.0000
1000/D   0.0625
```







As we can see from our table and graphs, a mu value of 1000/D still appears to be our strongest option since it still provides us with the greatest proportion of correctly classified documents while having the lowest proportion of false positives.

Part d)

The percentage error of the estimated rates using different mu values across five folds are given below, along with the average percentage error for each mu value:

```
> percentage_error
             Fold 1      Fold 2       Fold3    Fold 4      Fold 5    Average
1/(10*D) 0.27058824 0.111764706 0.111764706 0.4294118 0.111764706 0.20705882
1/D      0.27058824 0.111764706 0.111764706 0.4294118 0.111764706 0.20705882
10/D     0.13684211 0.005263158 0.005263158 0.2789474 0.005263158 0.08631579
100/D    0.06086957 0.178260870 0.056521739 0.1739130 0.060869565 0.10608696
1000/D   0.02800000 0.136000000 0.080000000 0.0800000 0.136000000 0.09200000
```

Overall, the average percentage errors for the different mu values ranged from about 8.6% to 20.7%. The differences in the estimated and actual estimated rates can likely be accounted for by the fact that the full training and testing sets are larger than the training and testing folds from cross validation. Thus, the larger training and especially testing sets lead to lower bias.

One way that the differences between the cross-validation rate estimates and the rates on the training sets could be minimized would be to increase the number of folds in cross validation or find more training data. This would reduce the bias of the CV rate estimates and thus also reduce the average percentage errors of the estimated rates.

Code

```
############################
# Kevin Gong
# STAT W4240
# Homework 04
# 4/2/14
#
# The following code analyzes the federalist papers
############################

#################
# 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/hw04")

# 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(Snowball)


#################
# Problem 1a
#################

########################################
# This code uses tm to preprocess the papers into a format useful for NB
preprocess.directory = function(dirname){

 # the directory must have all the relevant text files
 ds = DirSource(dirname)
 # Corpus will make a tm document corpus from this directory
 fp = Corpus( ds )
 # inspect to verify
 # inspect(fp[1])
 # another useful command
 # identical(fp[[1]], fp[["Federalist01.txt"]])
 # now let us iterate through and clean this up using tm functionality
 for (i in 1:length(fp)){
      # make all words lower case
      fp[i] = tm_map( fp[i] , tolower);
      # remove all punctuation
      fp[i] = tm_map( fp[i] , removePunctuation);
      # remove stopwords like the, a, and so on.
      fp[i] = tm_map( fp[i], removeWords, stopwords("english"));
      # remove stems like suffixes
      fp[i] = tm_map( fp[i], stemDocument)
      # remove extra whitespace
      fp[i] = tm_map( fp[i], stripWhitespace)
 }
 # now write the corpus out to the files for our future use.
 # MAKE SURE THE _CLEAN DIRECTORY EXISTS
 writeCorpus( fp , sprintf('%s_clean',dirname) )
```

```
}
########################################

source('hw04.R')

#process documents in each of the 4 directories
preprocess.directory("fp_hamilton_test")
preprocess.directory("fp_hamilton_train")
preprocess.directory("fp_madison_test")
preprocess.directory("fp_madison_train")



#################
# Problem 1b
#################

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

#read in the words from the documents
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")

length(hamilton.train)
dim(hamilton.train)

#################
# Problem 1c
#################

########################################
# 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)
}
#########################################

#generate word lists for each set of documents
list1 = make.sorted.dictionary.df(hamilton.train)
list2 = make.sorted.dictionary.df(hamilton.test)
list3 = make.sorted.dictionary.df(madison.train)
list4 = make.sorted.dictionary.df(madison.test)

#turn the word columns back into vector form
as.vector(list1[,1])
as.vector(list2[,1])
as.vector(list3[,1])
as.vector(list4[,1])

#create a complete dictionary by concatenating the 4 different word lists
complete_dictionary =
 c(as.vector(list1[,1]),as.vector(list2[,1]),as.vector(list3[,1]),as.vector(list4[,1]))


#alphabetize the complete dictionary
complete_dictionary = sort(complete_dictionary)

#check to make sure lengths add up
length(complete_dictionary) == length(as.vector(list1[,1]))+ length(as.vector(list2[,1]))
 + length(as.vector(list3[,1])) + length(as.vector(list4[,1]))

complete_dictionary
length(complete_dictionary)

#remove duplicate words from dictionary
complete_dictionary = unique(complete_dictionary)

test1 = unique(c(list1[,1],list2[,1],list3[,1],list4[,1]))

#place final dictionary into a dataframe
dictionary2 = as.data.frame(complete_dictionary)

#################
# Problem 1d
#################

########################################
# 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;
        }
    }
return(dtm);
}
#########################################

#create dtm matrices of the Hamilton and Madison training and testing sets
dtm.hamilton.train <- make.document.term.matrix(hamilton.train, dictionary2)
dtm.hamilton.test <- make.document.term.matrix(hamilton.test,dictionary2)
dtm.madison.train <- make.document.term.matrix(madison.train,dictionary2)
dtm.madison.test <- make.document.term.matrix(madison.test,dictionary2)

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

dim(list1)

#################
# Problem 1e
#################

########################################
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.hamilton.train <- make.log.pvec(dtm.hamilton.train,mu)
logp.hamilton.test <- make.log.pvec(dtm.hamilton.test,mu)
logp.madison.train <- make.log.pvec(dtm.madison.train,mu)
logp.madison.test <- make.log.pvec(dtm.madison.test,mu)

#double check ranges to make sure they roughly between -2 to -25
range(logp.hamilton.train)
range(logp.hamilton.test)
range(logp.madison.train)
range(logp.madison.test)


#################
# Problem 2
#################
```

```
naive.bayes <- function(logp.hamilton.train, logp.madison.train, log.prior.hamilton,
 log.prior.madison, dtm.test)

{

  #clear variables for each usage of function
  mads_document <- vector()
  hams_document <- vector()
  mads_words <- vector()
  hams_words <- vector()
  classification <- vector()


for (i in 1:nrow(dtm.test)){

      #sum probabilities of individual words being written by Hamilton or Madison
    mads_words[i] <- sum(dtm.test[i,]*logp.madison.train)
    hams_words[i] <- sum(dtm.test[i,]*logp.hamilton.train)

    #sum probabilities of documents being written by Hamilton or Madison
    mads_document[i] <- sum(mads_words[i]) + log.prior.madison
    hams_document[i] <- sum(hams_words[i]) + log.prior.hamilton

 }

for (k in 1:nrow(dtm.test))

  {

      #compare summed probabilities of a document to determine final classification
    if(hams_document[k] > mads_document[k]){

      classification[k] <- "Hamilton"

      }

    else{

      classification[k] <- "Madison"

      }
  }

  #display classification results in a matrix
  results_matrix <- data.frame(log.prob.hamilton = hams_document, log.prob.madison =
 mads_document, classifier = classification)
  return(results_matrix)
}



#################
# Problem 3
#################

#set our log priors
log.prior.hamilton = log(35/50)
log.prior.madison = log(15/50)
```

```
#run NB classification
ham <- naive.bayes(logp.hamilton.train, logp.madison.train, log.prior.hamilton,
 log.prior.madison, dtm.hamilton.test)
mad <- naive.bayes(logp.hamilton.train, logp.madison.train, log.prior.hamilton,
 log.prior.madison, dtm.madison.test)

#determine our true and false negative and positive proportions
true_pos = (sum(as.vector(ham$classifier)=="Hamilton"))/16
true_neg = (sum(as.vector(mad$classifier)=="Madison"))/11
false_pos = (sum(as.vector(mad$classifier)=="Hamilton"))/11
false_neg = (sum(as.vector(ham$classifier)=="Madison"))/16

ham
mad
correct_id = (sum(as.vector(ham$classifier)=="Hamilton") +
 sum(as.vector(mad$classifier)=="Madison"))/27
correct_id
true_pos
true_neg
false_pos
false_neg




#################
# Problem 4a
#################


#randomly sample from a number list to determine the rows of the original dtm files to
 use as testing
hamilton_numbers = 1:35
madison_numbers = 1:15

#hamilton samples - use 7/10 to approximate the full training set's ratio of 35/50
A <- sample(hamilton_numbers,7)
E <- sample(hamilton_numbers[-A],7)
J <- sample(hamilton_numbers[-c(A,E)],7)
N <- sample(hamilton_numbers[-c(A,E,J)],7)
R <- sample(hamilton_numbers[-c(A,E,J,N)],7)

A
E
J
N
R

#madison samples - use 3/10 to approximate the full training set's ratio of 15/50
B <- sample(madison_numbers,3)
G <- sample(madison_numbers[-B],3)
K <- sample(madison_numbers[-c(B,G)],3)
O <- sample(madison_numbers[-c(B,G,K)],3)
S <- sample(madison_numbers[-c(B,G,K,O)],3)

B
G
K
O
```

S

```
#generate Hamilton and Madison training and testing sets using the random samples from
 above
ham_testing1 <- dtm.hamilton.train[A,]
mad_testing1 <- dtm.madison.train[B,]
ham_training1 <- dtm.hamilton.train[-A,]
mad_training1 <- dtm.madison.train[-B,]

ham_testing2 <- dtm.hamilton.train[E,]
mad_testing2 <- dtm.madison.train[G,]
ham_training2 <- dtm.hamilton.train[-E,]
mad_training2 <- dtm.madison.train[-G,]

ham_testing3 <- dtm.hamilton.train[J,]
mad_testing3 <- dtm.madison.train[K,]
ham_training3 <- dtm.hamilton.train[-J,]
mad_training3 <- dtm.madison.train[-K,]

ham_testing4 <- dtm.hamilton.train[N,]
mad_testing4 <- dtm.madison.train[O,]
ham_training4 <- dtm.hamilton.train[-N,]
mad_training4 <- dtm.madison.train[-O,]

ham_testing5 <- dtm.hamilton.train[R,]
mad_testing5 <- dtm.madison.train[S,]
ham_training5 <- dtm.hamilton.train[-R,]
mad_training5 <- dtm.madison.train[-S,]

dim(ham_testing1)
dim(ham_training1)



#allocate empty matrices to hold the proportion of correctly identified documents, false
 positives, and false negatives

correct <- matrix(nrow=5,ncol=5)
correct

false_pos_final <- matrix(nrow=5,ncol=5)
false_neg_final <- matrix(nrow=5,ncol=5)

false_pos_final
false_neg_final



#place our 5 mu values into a vector and define our log priors

mu_values = c(1/(10*4875), 1/4875, 10/4875, 100/4875, 1000/4875)

log.prior.hamilton = log(35/50)
log.prior.madison = log(15/50)
```

```
#use for loops to fill in our three 5x5 matrices; outer i loop cycles through mu values,
 inner k loop and if/else functions cycle through different CV folds

for(i in 1:5){
 logp.hamilton.train_temp <- vector()
 logp.madison.train_temp <- vector()
 hams1 = vector()
 mads1 = vector()
 correct1 = vector()
 false_positive1 = vector()
 false_negative1 = vector()
 hams2 = vector()
 mads2 = vector()
 correct2 = vector()
 false_positive2 = vector()
 false_negative2 = vector()
 hams3 = vector()
 mads3 = vector()
 correct3 = vector()
 false_positive3 = vector()
 false_negative3 = vector()
 hams4 = vector()
 mads4 = vector()
 correct4 = vector()
 false_positive4 = vector()
 false_negative4 = vector()
 hams5 = vector()
 mads5 = vector()
 correct5 = vector()
 false_positive5 = vector()
 false_negative5 = vector()

for(k in 1:5){

 if(k==1){

 logp.hamilton.train_temp <- make.log.pvec(ham_training1,mu_values[i])
 logp.madison.train_temp <- make.log.pvec(mad_training1,mu_values[i])


 hams1 <- naive.bayes(logp.hamilton.train_temp, logp.madison.train_temp,
 log.prior.hamilton, log.prior.madison, ham_testing1)
 mads1 <- naive.bayes(logp.hamilton.train_temp, logp.madison.train_temp,
 log.prior.hamilton, log.prior.madison, mad_testing1)

 correct1 = ((sum(as.vector(hams1$classifier)=="Hamilton")) +
 (sum(as.vector(mads1$classifier)=="Madison")))/10
 false_positive1 = (sum(as.vector(mads1$classifier)=="Hamilton"))/3
 false_negative1 = (sum(as.vector(hams1$classifier)=="Madison"))/7


 correct[i,k] = correct1
 false_pos_final[i,k] = false_positive1
 false_neg_final[i,k] = false_negative1

 }


 else if(k==2){
```

```
logp.hamilton.train_temp <- make.log.pvec(ham_training2,mu_values[i])
logp.madison.train_temp <- make.log.pvec(mad_training2,mu_values[i])


hams2 <- naive.bayes(logp.hamilton.train_temp, logp.madison.train_temp,
log.prior.hamilton, log.prior.madison, ham_testing2)
mads2 <- naive.bayes(logp.hamilton.train_temp, logp.madison.train_temp,
log.prior.hamilton, log.prior.madison, mad_testing2)

correct2 = ((sum(as.vector(hams2$classifier)=="Hamilton")) +
(sum(as.vector(mads2$classifier)=="Madison")))/10
false_positive2 = (sum(as.vector(mads2$classifier)=="Hamilton"))/3
false_negative2 = (sum(as.vector(hams2$classifier)=="Madison"))/7


correct[i,k] = correct2
false_pos_final[i,k] = false_positive2
false_neg_final[i,k] = false_negative2

}

else if(k==3){

logp.hamilton.train_temp <- make.log.pvec(ham_training3,mu_values[i])
logp.madison.train_temp <- make.log.pvec(mad_training3,mu_values[i])


hams3 <- naive.bayes(logp.hamilton.train_temp, logp.madison.train_temp,
log.prior.hamilton, log.prior.madison, ham_testing3)
mads3 <- naive.bayes(logp.hamilton.train_temp, logp.madison.train_temp,
log.prior.hamilton, log.prior.madison, mad_testing3)

correct3 = ((sum(as.vector(hams3$classifier)=="Hamilton")) +
(sum(as.vector(mads3$classifier)=="Madison")))/10
false_positive3 = (sum(as.vector(mads3$classifier)=="Hamilton"))/3
false_negative3 = (sum(as.vector(hams3$classifier)=="Madison"))/7


correct[i,k] = correct3
false_pos_final[i,k] = false_positive3
false_neg_final[i,k] = false_negative3

}


else if(k==4){

logp.hamilton.train_temp <- make.log.pvec(ham_training4,mu_values[i])
logp.madison.train_temp <- make.log.pvec(mad_training4,mu_values[i])


hams4 <- naive.bayes(logp.hamilton.train_temp, logp.madison.train_temp,
log.prior.hamilton, log.prior.madison, ham_testing4)
mads4 <- naive.bayes(logp.hamilton.train_temp, logp.madison.train_temp,
log.prior.hamilton, log.prior.madison, mad_testing4)

correct4 = ((sum(as.vector(hams4$classifier)=="Hamilton")) +
(sum(as.vector(mads4$classifier)=="Madison")))/10
false_positive4 = (sum(as.vector(mads4$classifier)=="Hamilton"))/3
false_negative4 = (sum(as.vector(hams4$classifier)=="Madison"))/7
```

```
correct[i,k] = correct4
false_pos_final[i,k] = false_positive4
false_neg_final[i,k] = false_negative4

}


else {

logp.hamilton.train_temp <- make.log.pvec(ham_training5,mu_values[i])
logp.madison.train_temp <- make.log.pvec(mad_training5,mu_values[i])


hams5 <- naive.bayes(logp.hamilton.train_temp, logp.madison.train_temp,
log.prior.hamilton, log.prior.madison, ham_testing5)
mads5 <- naive.bayes(logp.hamilton.train_temp, logp.madison.train_temp,
log.prior.hamilton, log.prior.madison, mad_testing5)

correct5 = ((sum(as.vector(hams5$classifier)=="Hamilton")) +
(sum(as.vector(mads5$classifier)=="Madison")))/10
false_positive5 = (sum(as.vector(mads5$classifier)=="Hamilton"))/3
false_negative5 = (sum(as.vector(hams5$classifier)=="Madison"))/7


correct[i,k] = correct5
false_pos_final[i,k] = false_positive5
false_neg_final[i,k] = false_negative5

}



}



}



correct
false_pos_final
false_neg_final

#rename rows and columns of the three matrices

rownames(correct) <- c("1/(10*D)","1/D","10/D","100/D","1000/D")
colnames(correct) <- c("Fold 1","Fold 2", "Fold3", "Fold 4", "Fold 5")

rownames(false_pos_final) <- c("1/(10*D)","1/D","10/D","100/D","1000/D")
colnames(false_pos_final) <- c("Fold 1","Fold 2", "Fold3", "Fold 4", "Fold 5")

rownames(false_neg_final) <- c("1/(10*D)","1/D","10/D","100/D","1000/D")
colnames(false_neg_final) <- c("Fold 1","Fold 2", "Fold3", "Fold 4", "Fold 5")


correct
false_pos_final
```

```
false_neg_final



#generate graphs of the three matrices

colors <- c("green", "cyan", "purple", "red", "black")
markers = 1:5
categories = c("1/(10*D)","1/D","10/D","100/D","1000/D")
matplot(t(correct), type="o",lty=1, col=colors, pch=markers ,bty="n",las=1,
 ylab="Proportion Correctly Classified", xlab="Fold #", main="Performance of Different Mu
 Values in Classifying Documents")
legend("topleft",col=colors, pch=markers,categories,lwd=1)


colors <- c("green", "cyan", "purple", "red", "black")
markers = 1:5
categories = c("1/(10*D)","1/D","10/D","100/D","1000/D")
matplot(t(false_pos_final), type="o",lty=1, col=colors, pch=markers ,bty="n",las=1,
 ylab="Proportion of False Positives", xlab="Fold #", main="Performance of Different Mu
 Values - False Positives")
legend("bottomleft",col=colors, pch=markers,categories,lwd=1)


colors <- c("green", "cyan", "purple", "red", "black")
markers = 1:5
categories = c("1/(10*D)","1/D","10/D","100/D","1000/D")
matplot(t(false_neg_final), type="o",lty=1, col=colors, pch=markers ,bty="n",las=1,
 ylab="Proportion of False Negatives", xlab="Fold #", main="Performance of Different Mu
 Values - False Negatives")
legend("topleft",col=colors, pch=markers,categories,lwd=1)



#################
# Problem 4c
#################


#allocate empty matrices to hold the proportion of correctly identified documents, false
 positives, and false negatives

complete_correct = matrix(nrow=5,ncol=1)
complete_correct

complete_false_pos_final <- matrix(nrow=5,ncol=1)
complete_false_neg_final <- matrix(nrow=5,ncol=1)

complete_false_pos_final
complete_false_neg_final


#place our 5 mu values into a vector and define our log priors

mu_values = c(1/(10*4875), 1/4875, 10/4875, 100/4875, 1000/4875)

log.prior.hamilton = log(35/50)
log.prior.madison = log(15/50)
```

```
#use for loop to fill in our three matrices

for(i in 1:5){


 logp.hamilton.train_temp <- make.log.pvec(dtm.hamilton.train,mu_values[i])
 logp.madison.train_temp <- make.log.pvec(dtm.madison.train,mu_values[i])


 hams1 <- naive.bayes(logp.hamilton.train_temp, logp.madison.train_temp,
 log.prior.hamilton, log.prior.madison, dtm.hamilton.test)
 mads1 <- naive.bayes(logp.hamilton.train_temp, logp.madison.train_temp,
 log.prior.hamilton, log.prior.madison, dtm.madison.test)

 correct1 = ((sum(as.vector(hams1$classifier)=="Hamilton")) +
 (sum(as.vector(mads1$classifier)=="Madison")))/27
 false_positive1 = (sum(as.vector(mads1$classifier)=="Hamilton"))/11
 false_negative1 = (sum(as.vector(hams1$classifier)=="Madison"))/16


 complete_correct[i,1] = correct1
 complete_false_pos_final[i,1] = false_positive1
 complete_false_neg_final[i,1] = false_negative1

 }



complete_correct
complete_false_pos_final
complete_false_neg_final

#rename rows of the three matrices

rownames(complete_correct) <- c("1/(10*D)","1/D","10/D","100/D","1000/D")
rownames(complete_false_pos_final) <- c("1/(10*D)","1/D","10/D","100/D","1000/D")
rownames(complete_false_neg_final) <- c("1/(10*D)","1/D","10/D","100/D","1000/D")

complete_correct
complete_false_pos_final
complete_false_neg_final



#generate graphs of the three matrices


colors <- c("green", "cyan", "purple", "red", "black")
markers = 1:5
categories = c("1/(10*D)","1/D","10/D","100/D","1000/D")
matplot(t(complete_correct), lty=1, col=colors, pch=markers ,bty="n",las=1,
 ylab="Proportion Correctly Classified", xlab="Fold #", main="Performance of Different Mu
 Values in Classifying Documents",ylim=c(0,1))
legend("topleft",col=colors, pch=markers,categories,lwd=1)


colors <- c("green", "cyan", "purple", "red", "black")
markers = 1:5
categories = c("1/(10*D)","1/D","10/D","100/D","1000/D")
```

```
matplot(t(complete_false_pos_final), lty=1, col=colors, pch=markers ,bty="n",las=1,
 ylab="Proportion of False Positives", main="Performance of Different Mu Values - False
 Positives",ylim=c(0,1))
legend("bottomleft",col=colors, pch=markers,categories,lwd=1)


colors <- c("green", "cyan", "purple", "red", "black")
markers = 1:5
categories = c("1/(10*D)","1/D","10/D","100/D","1000/D")
matplot(t(complete_false_neg_final), lty=1, col=colors, pch=markers ,bty="n",las=1,
 ylab="Proportion of False Negatives",  main="Performance of Different Mu Values - False
 Negatives",ylim=c(0,0.1))
legend("topleft",col=colors, pch=markers,categories,lwd=1)


#################
# Problem 4d
#################

#allocate matrix to calculate percentage error

percentage_error = matrix(nrow=5,ncol=6)
percentage_error

#rename rows and columns of matrix
rownames(percentage_error) <- c("1/(10*D)","1/D","10/D","100/D","1000/D")
colnames(percentage_error) <- c("Fold 1","Fold 2", "Fold3", "Fold 4", "Fold 5","Average")


#calculate percentage errors
for(i in 1:5){
percentage_error[,i]= abs(complete_correct-correct[,i])/complete_correct
}

#calculate average percentage errors across 5 folds
for(i in 1:5){
percentage_error[i,6]= mean(percentage_error[i,1:5])
}

percentage_error


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