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)

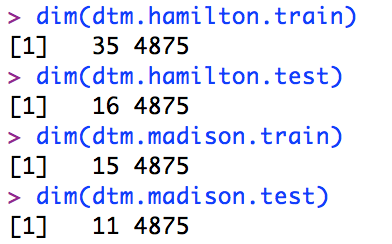
(see code)

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

Part d)

(see code)

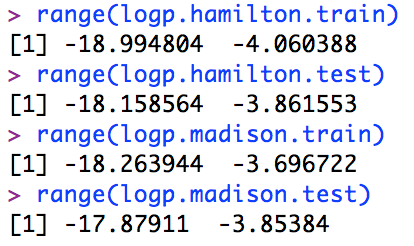
We create document term lists which have the following dimensions:



Part e)

(see code)

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



**Question 2**

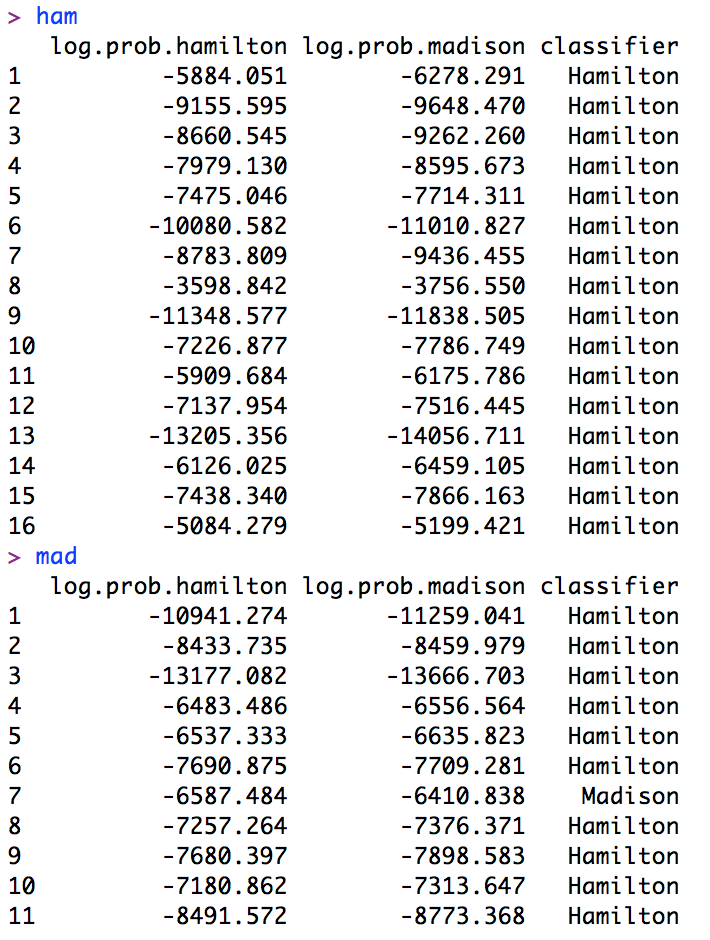
(see code)

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.

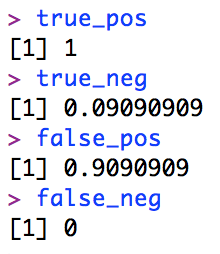
**Question 3**

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.



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



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.

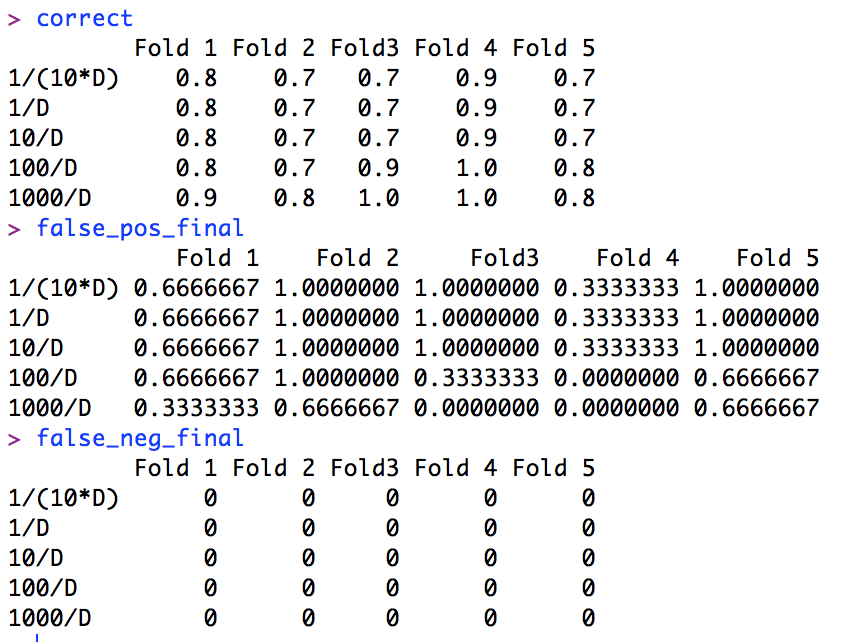
**Question 4**

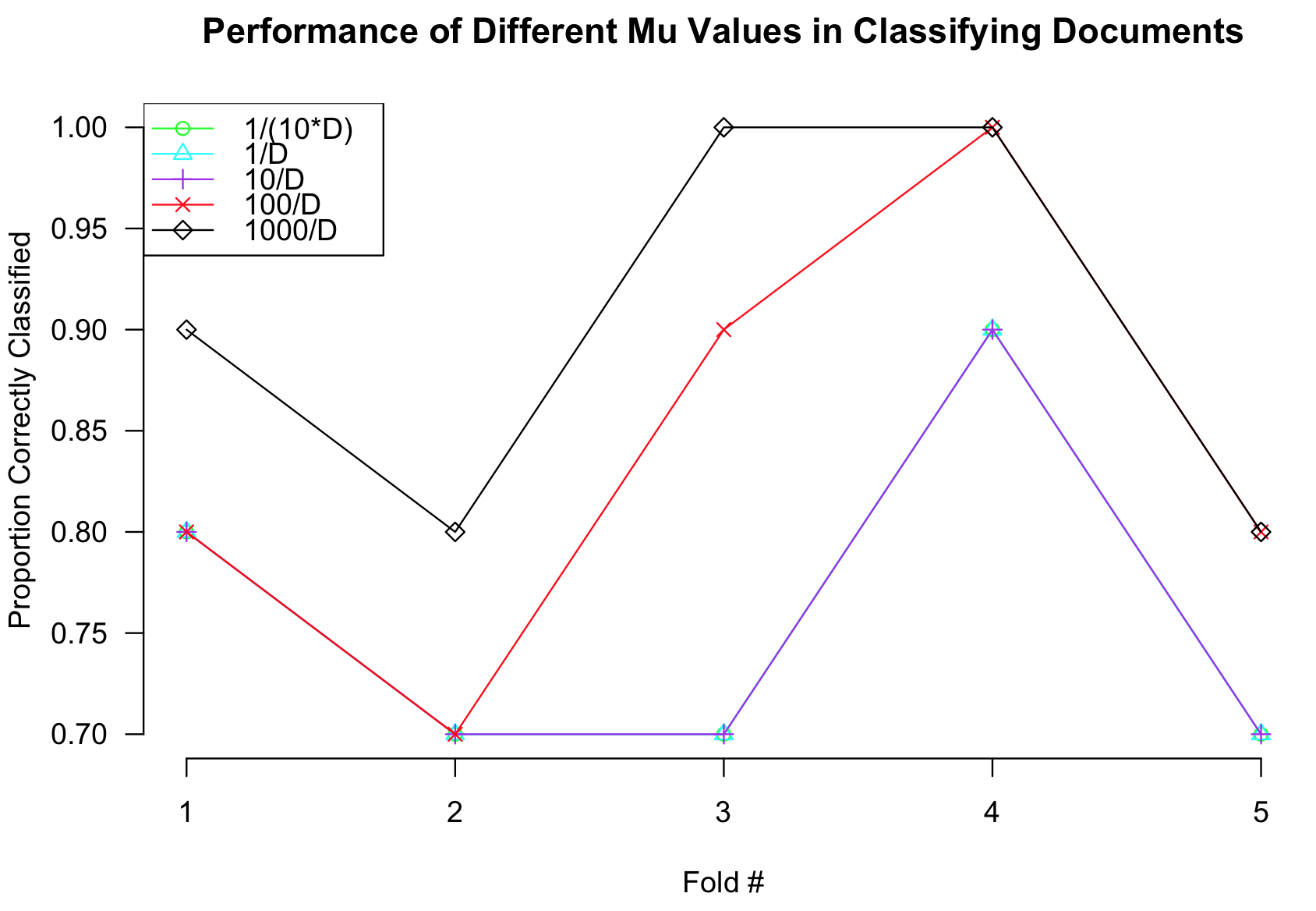
Part a)

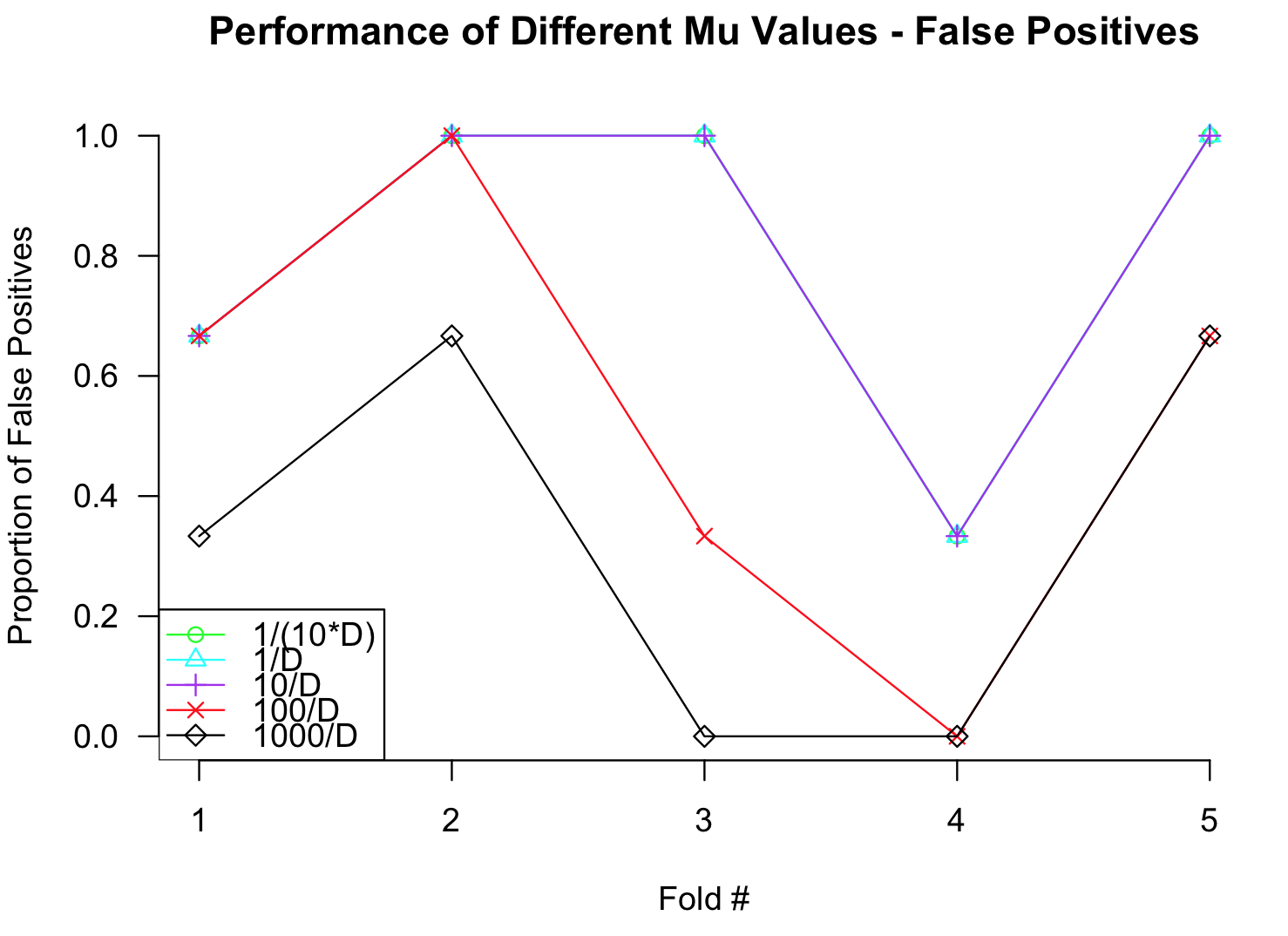
We have *a lot* of code for this part, but the basic steps boil down to:

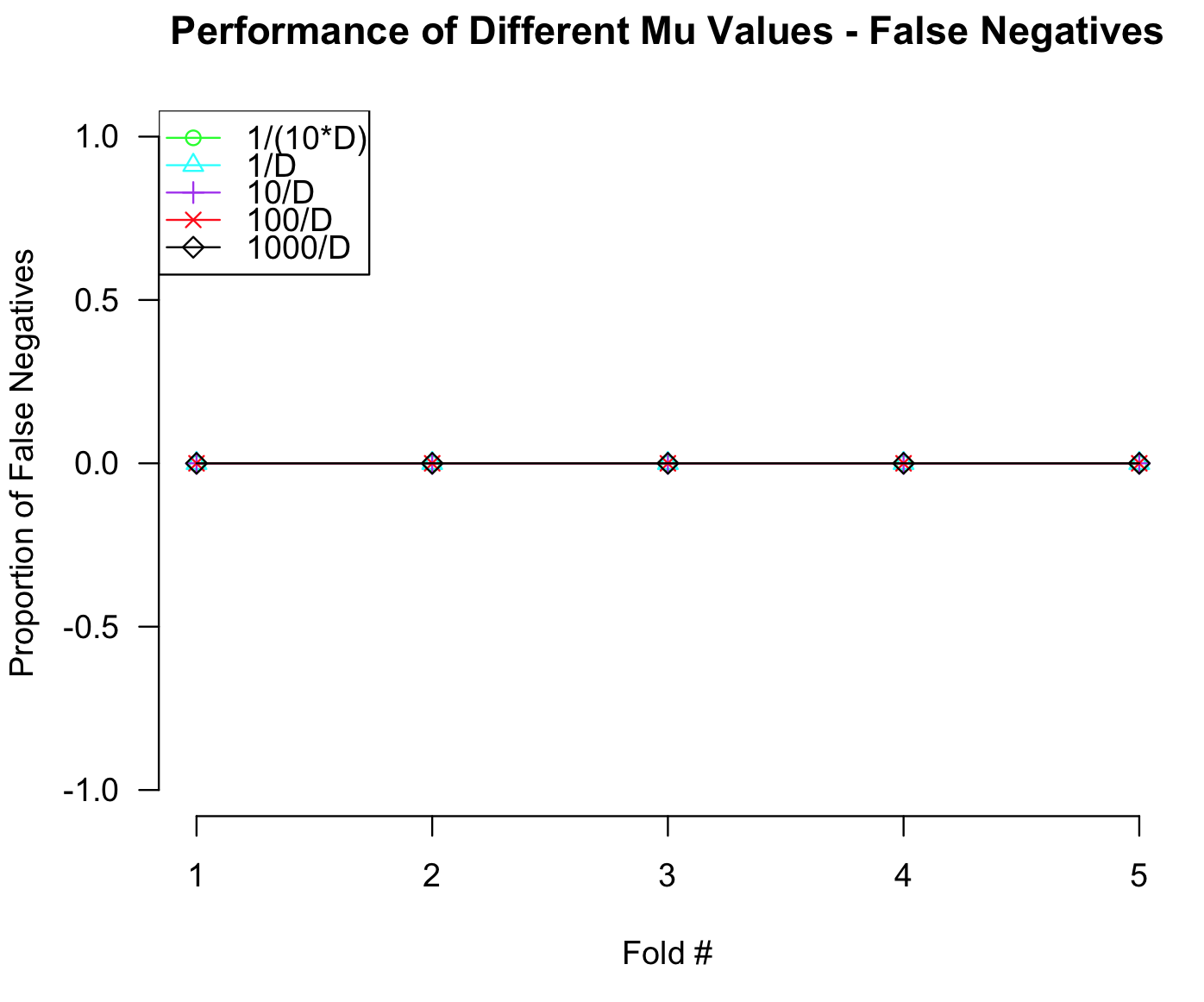
1. Create 5 folds of 10 randomly sampled documents each with 7 Hamilton and 3 Madison documents.
2. Generate document term matrices for Hamilton and Madison training and testing using the 5 folds from step 1.
3. Create 3 empty matrices to store proportion correctly classified, false positives, and false 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:







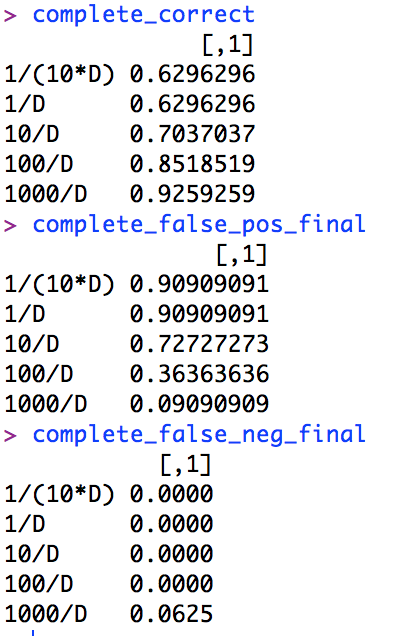


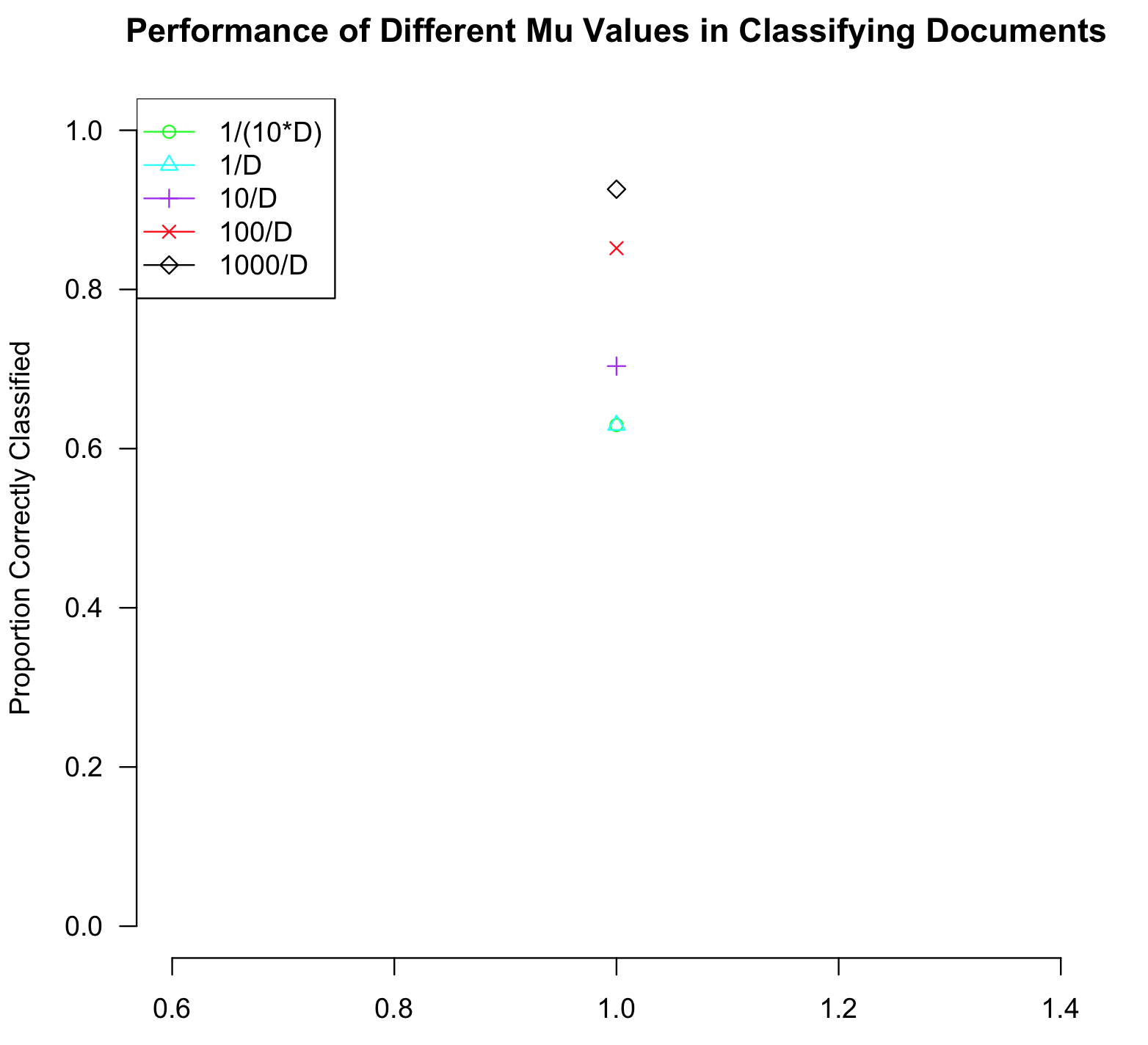
Part b)

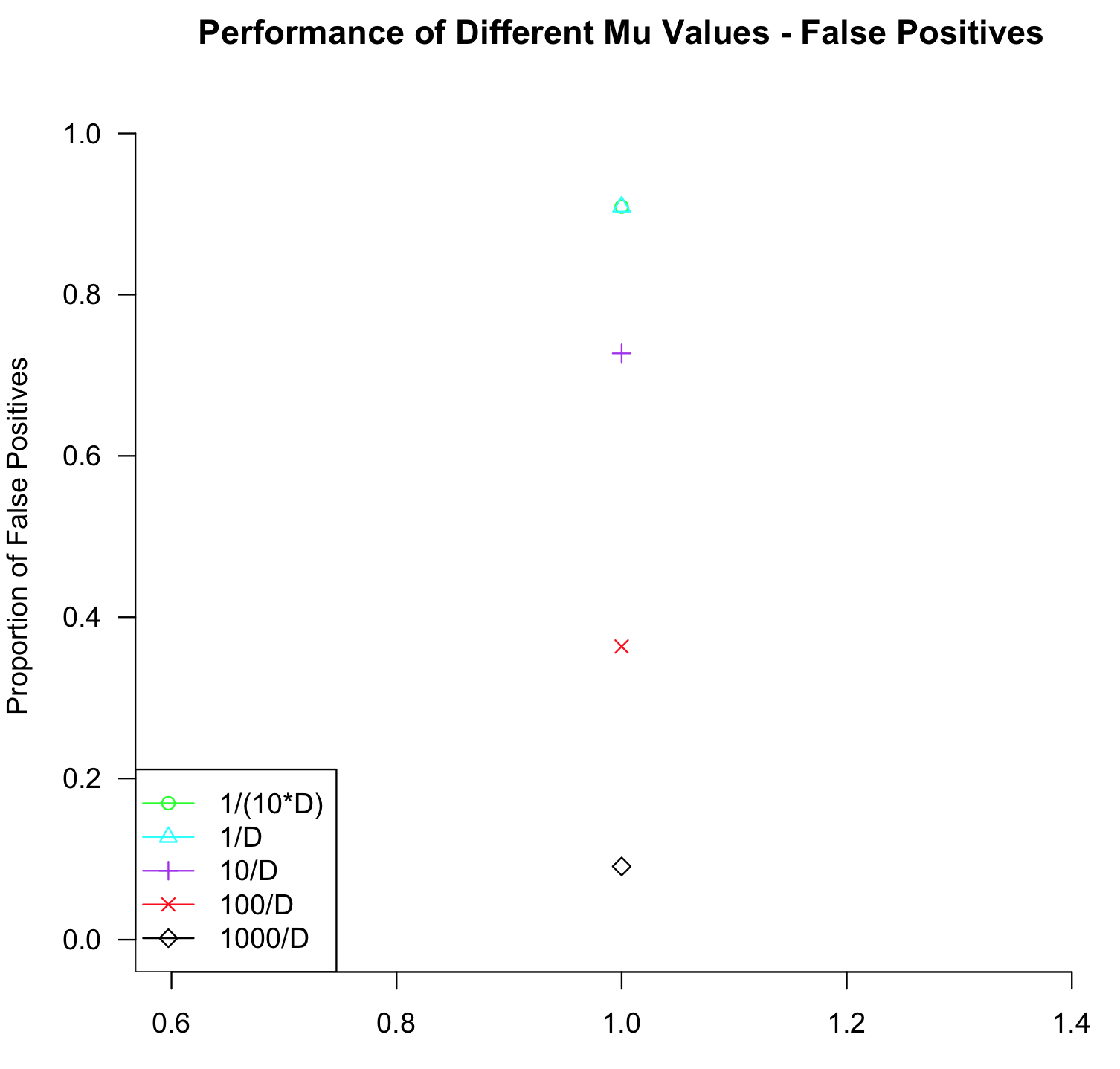
From our results from part a, it seems that the best value for mu is 1000/D because this mu value consistently has the highest percentage of correctly classified documents as well as the lowest rate of false positives. Since we ultimately care about the ability of our Naïve Bayes classifier to correctly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

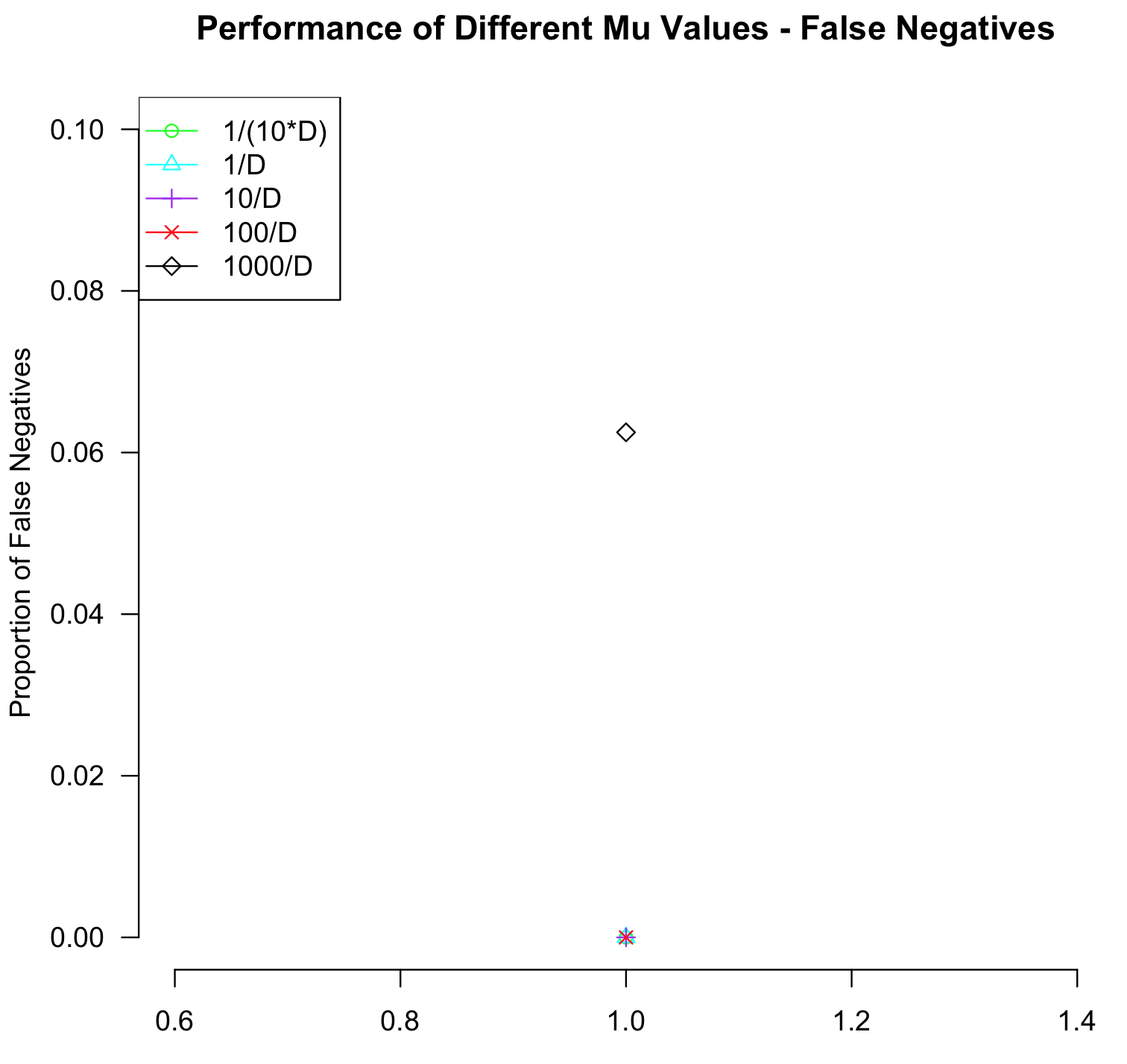
Part c)

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:





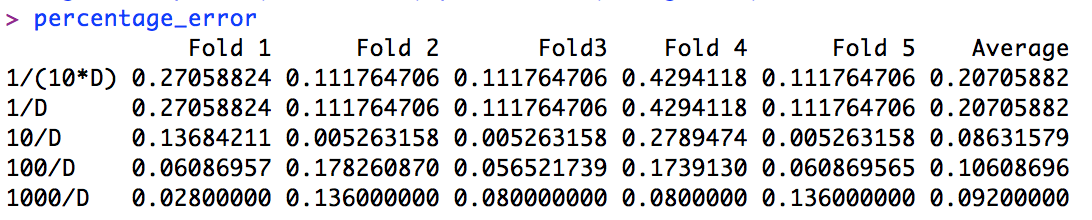




As we can see from our table and graphs, a mu value of 1000/D still appears to be our strongest option since it still provides us with the greatest proportion of correctly classified documents while having the lowest proportion of false positives.

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:



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

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