Kevin Gong
kg2445
HW06
Stat W4240
Section 2

Homework 6

Problem #1

Part a)

The log-transformed salaries with NAs removed is below:

```
> H2$Salary
  [1] 6.163315 6.173786 6.214608 4.516339 6.620073 4.248495 4.605170 4.317488 7.003065 6.248319 6.239301 6.309918 6.551080 5.480639
 [15] 6.652863 5.164786 4.905275 4.605170 4.744932 6.396930 6.655012 6.639876 6.562914 6.620073 6.437752 6.802395 4.700480 6.417549
 [29] 5.703782 6.745236 4.499810 4.212128 5.192957 5.720312 5.370638 5.511411 6.703188 6.774224 4.248495 7.090077 6.514713 6.028279
 [43] 5.828946 6.032287 7.207860 4.499810 5.616771 5.438079 5.416100 6.856462 4.317488 4.653960 5.768321 6.745236 6.282267 6.838762
 [57] 6.745236 5.347108 5.783825 5.616771 6.109248 7.588324 7.549609 6.396930 6.948578 4.700480 5.560682 6.163315 6.067268 7.106606
 [71] 4.248495 4.976734 6.388561 7.529116 5.703782 6.194405 7.807917 5.926926 6.620073 7.069023 4.248495 7.313220 5.953243 7.562978
 [85] 5.370638 6.802395 5.043425 6.551080 6.282267 5.893024 6.597600 5.298317 5.991465 5.991465 6.603266 6.214608 6.396930 6.496021
 [99] 6.856462 6.620073 5.695414 5.783825 4.471639 5.164786 4.499810 7.120848 6.063785 4.605170 5.105945 5.521461 7.170120 6.650710
[113] 6.916054 5.616771 6.652863 6.745236 5.899897 4.553877 4.700480 4.605170 5.625821 4.382027 6.396930 5.298317 6.487684 4.317488
[127] 7.788419 5.521461 5.043425 6.461468 5.703782 4.700480 6.715383 5.273000 6.109248 6.445720 4.460144 7.170120 6.907755 7.495542
[141] 7.177782 6.603266 6.437752 4.828314 6.950176 6.586172 5.703782 5.899897 4.317488 7.076090 5.310740 5.416100 6.263398 5.579730
[155] 6.668863 6.684612 6.375876 4.976734 6.040255 4.317488 6.354370 6.659294 4.499810 5.010635 6.551080 6.309918 6.476972 4.219508
[169] 4.605170 6.507278 5.164786 4.919981 7.662624 6.774224 4.787492 4.941642 5.347108 6.684612 5.480639 5.857933 5.164786 5.298317
[183] 7.570443 6.551080 6.620073 6.109248 5.147494 7.138867 6.620073 5.247024 6.363028 4.867534 6.109248 5.703782 5.521461 6.956545
[197] 5.370638 5.991465 6.327937 7.420579 6.189290 6.052089 6.214608 5.521461 5.991465 6.109248 6.620073 4.248495 6.774224 5.247024
[211] 5.252273 6.606650 5.521461 4.941642 4.579852 6.606650 4.941642 5.833837 6.907755 4.605170 4.499810 5.298317 4.905275 5.043425
[225] 6.163315 7.279319 5.010635 4.653960 5.857933 4.499810 6.272877 5.833837 6.845880 5.857933 5.788941 5.521461 6.606650 6.052089
[239] 6.829794 5.220356 6.824374 5.658321 5.501258 5.459586 7.047517 5.075174 6.052089 6.802395 6.214608 5.625821 6.620073 5.075174
[253] 7.170120 6.263398 6.309918 7.377759 4.787492 5.105945 6.551080 6.774224 5.953243 6.866933 6.907755
```

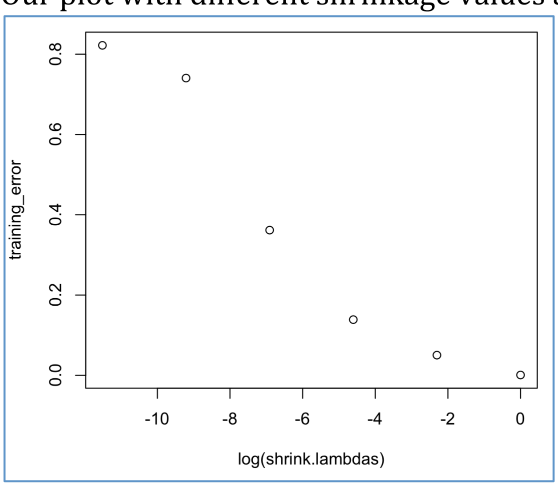
Part b)

We create a training set with 200 observations and a testing set with 63 observations.

```
> dim(H2.train)
[1] 200  20
> dim(H2.test)
[1] 63 20
```

Part c)

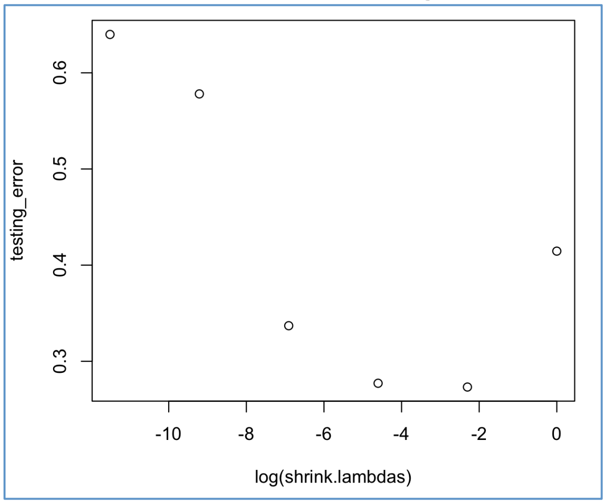
Our plot with different shrinkage values and training MSE is below.



We can see that setting lambda equal to 1 (or log(lambda equal to zero) results in the lowest training MSE of approximately 0.

Part d)

Our plot with different shrinkage values and testing MSE is below.



We can see that setting lambda equal to 0.1 results in the lowest testing MSE, while a lambda of 0.01 is not far behind.

Part e)

When comparing the test MSE results of the three techniques, we see that boosting is easily superior to lasso and best subset lm since it has a far lower test MSE than the other two techniques.

```
> boost_results
[1] 0.0008248453
> bestsub
[1] 0.6495298
> best.lasso
[1] 0.6475407
```

Part f)

The variables that appear to be the most important predictors in the boosted model are CHits and PutOuts, followed by Walks and HmRun.

```
> summary(boost_hitters_test)
                var      rel.inf
CHits         CHits 17.95507195
PutOuts     PutOuts 13.23335704
Walks         Walks  7.91197775
HmRun         HmRun  6.31250164
AtBat         AtBat  6.20184037
RBI             RBI  6.08033074
Assists     Assists  5.78184841
Hits           Hits  4.88893887
Runs           Runs  4.66571916
CAtBat       CAtBat  4.64782413
CWalks       CWalks  4.46288072
Years         Years  4.39029319
CRBI           CRBI  3.75408012
Errors       Errors  3.36363798
CHmRun       CHmRun  3.28946252
CRuns         CRuns  2.78928273
NewLeague NewLeague  0.18405574
Division   Division  0.06295538
League       League  0.02394158
```

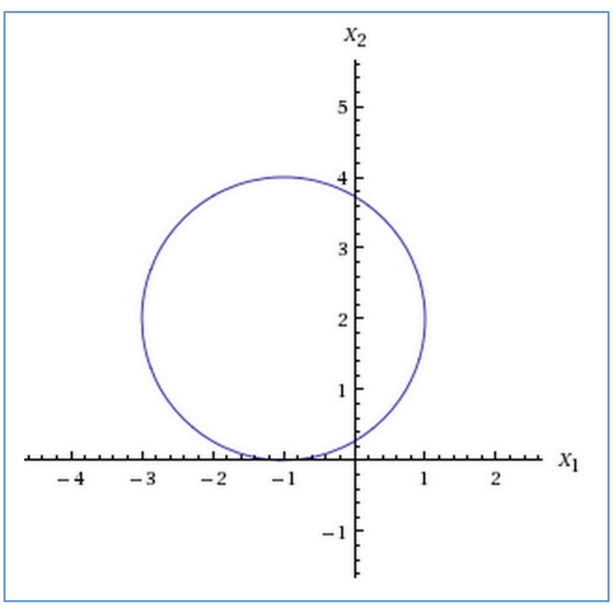
<u>Part g)</u>

The test MSE from the bagging approach is about 0.23. This is far higher than the lowest test MSE for boosting that we found in part e.

```
> mean((prediction_bag-H2.test$Salary)^2)
[1] 0.2347789
```

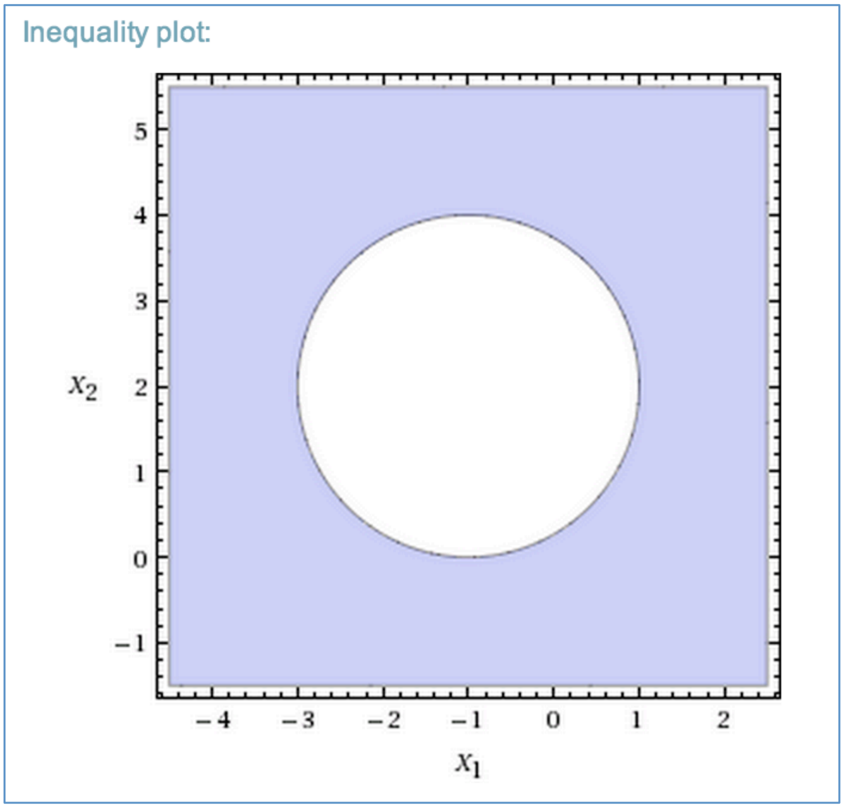
Problem #2

<u>Part a)</u>

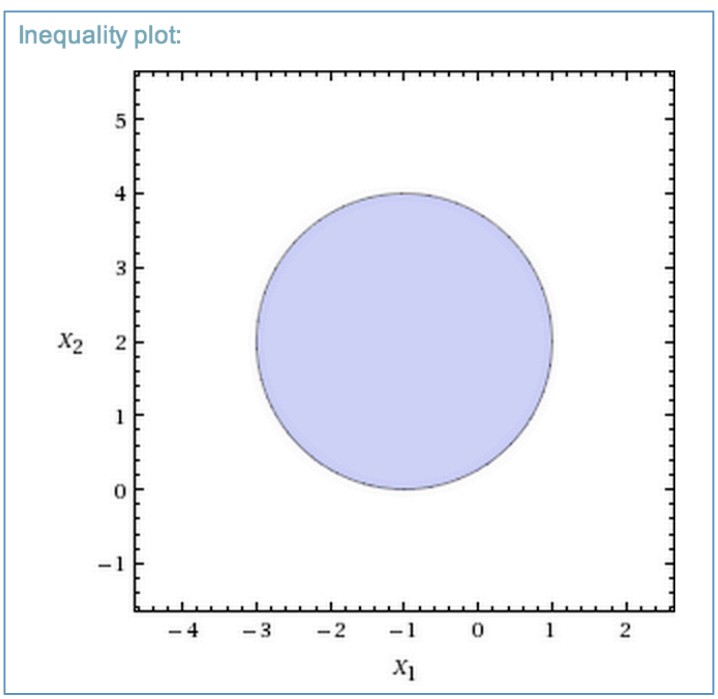


Part b)

$(1 + X_1)^2 + (2 - X_2)^2 > 4$ looks like this:



Meanwhile, $(1 + X_1)^2 + (2 - X_2)^2 <= 4$ looks like this:



Part c)

From simply inserting the (X1,X2) coordinates into the inequality, we find that (0,0) is classified as blue, (-1,1) is classified as red, (2,2) is classified as blue, and (3,8) is classified as blue. This can also be seen in the graph where the points are either inside (red) or outside (blue) the circle.

Part d)

The decision boundary in part c is linear in terms of X1, X1^2, X2, and X2^2 since the inequality which defines the boundary forms a circle. Thus, the decision boundary must be linear in terms of these variables since they form the continuous edge of the circle.

Problem #3

Part a)

```
> dim(train_set)
[1] 800  18
> dim(test_set)
[1] 270  18
```

We create a training set of 800 observations and a testing set of 270 observations.

Part b)

```
> summary(svm_train)

Call:
svm(formula = y ~ ., data = training, kernel = "linear", cost = 0.01, scale = TRUE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01
      gamma:  0.05555556

Number of Support Vectors:  432

 ( 215 217 )


Number of Classes:  2

Levels:
 CH MM
```

The support vector machine generated 432 support vectors and 2 classes: CH and MM.

Part c)

```
> train_error
[1] 0.16625
> test_error
[1] 0.1740741
```

The training error is about 0.166, while the testing error is about 0.174.

Part d)
Part e)

Part f)

Using a support vector machine with a radial kernel, we find that the support vector machine generated 617 support vectors. Meanwhile, the training error for this svm was about 0.38, while the testing error was about 0.41.

```
> summary(svm_train_radial)

Call:
svm(formula = y ~ ., data = training, kernel = "radial", cost = 0.01, scale = TRUE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  0.01
      gamma:  0.05555556

Number of Support Vectors:  617

 ( 306 311 )


Number of Classes:  2

Levels:
 CH MM
```

```
> train_error_radial
[1] 0.3825
> test_error_radial
[1] 0.4111111
```

Part g)

Using a support vector machine with a polynomial kernel and degree of 2, we find that the support vector machine generated 620 support vectors. Meanwhile, the training error for this svm was about 0.38, while the testing error was about 0.41.

```
> summary(svm_train_poly)

Call:
svm(formula = y ~ ., data = training, kernel = "polynomial", cost = 0.01, degree = 2, scale = TRUE)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  0.01
     degree:  2
      gamma:  0.05555556
     coef.0:  0

Number of Support Vectors:  620

 ( 306 314 )


Number of Classes:  2

Levels:
 CH MM



> train_error_poly
[1] 0.3825
> test_error_poly
[1] 0.4111111
```

Part h)

Overall, the best approach, or the one with the lowest testing error, seems to be the original svm with a linear kernel.

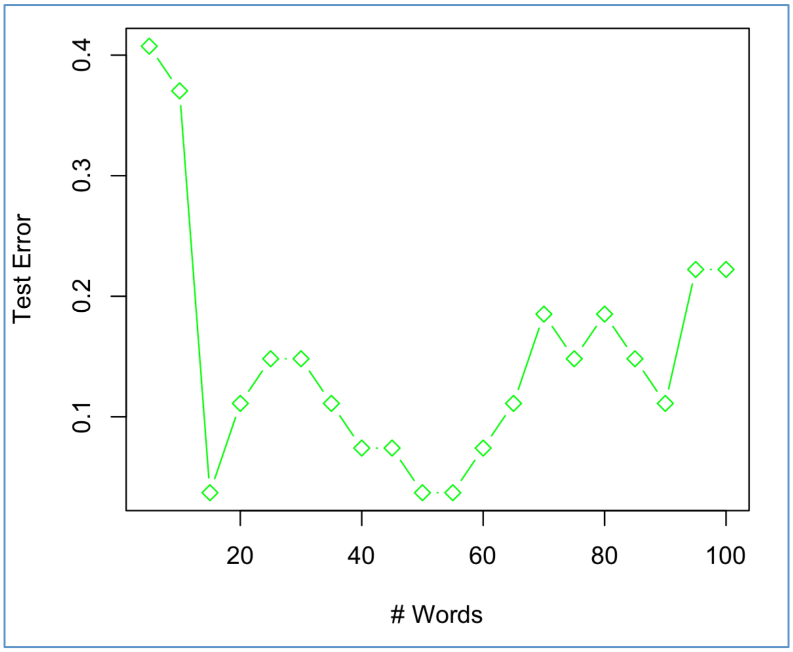
Problem #4

Part a)

Overall, 21 of 27 documents were classified correctly, or about 78%. This result is very good compared to what we obtained from previous problem sets with naïve bayes, but it still lacks the accuracy of some of the lasso techniques we used (which generated around a 91% correct classification rate).

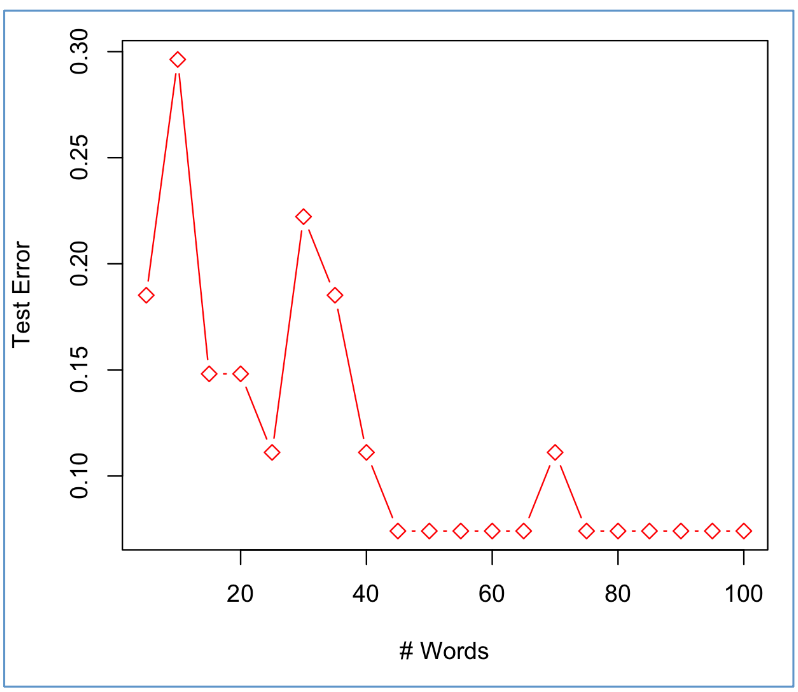
```
> test_prediction
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
 1  1  1  1  0  1  1  1  1  1  1  1  1  0  0  0  0  1  1  0  0  0  0  0  0  0  0
Levels: 0 1
> testing$y
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
Levels: 0 1
> sum(test_prediction==testing$y)
[1] 21
```

Part b)



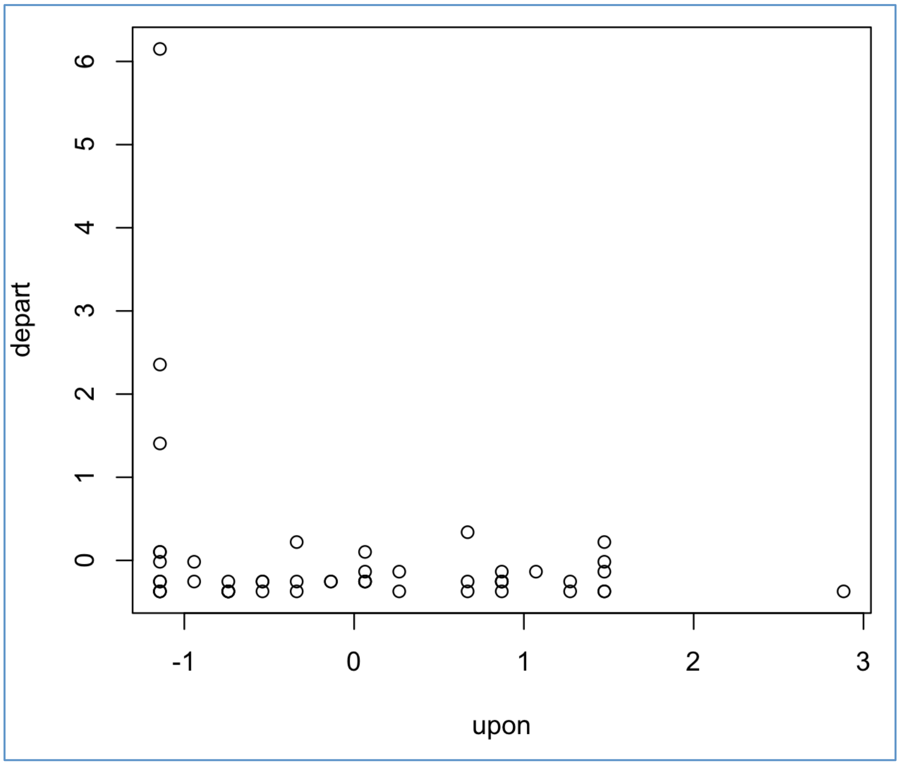
We see that test error is lowest when we use a linear SVM with the first 15, 50, or 55 words. While there seem to be multiple trends happening in the graph, it looks like there's a general trend of increasing test error as one moves away from the median number of words.

Part c)



We see that test error is lowest when we use a linear SVM with the first 45 to 100 words, with the exception of 70 words. There seems to be a general trend of decreasing test error as the number of words used increases.

Part d)



It appears that the authorship of a paper is more likely to be classified as Madison when the word "depart" is used, while authorship can be either classified as Madison or Hamilton when the word "upon" is used.

Problem #5

Part a)

Part b)

Each iteration of the K-means clustering algorithm decreases the objective function (in this case the squared error function) since each additional k increases clusters and moves the mean closer to the center of the data cluster, thus minimizing the average squared Euclidean distance of observations.

Code

```
############################
# Kevin Gong
# STAT W4240
# Homework 06, Problem 1
# May 05, 2014
#

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

#################
# 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/hw06")


# 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(ISLR)
library(gbm)
library(glmnet)
library(randomForest)


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

data("Hitters")
Hitters

H2 <- Hitters[!is.na(Hitters$Salary),,drop=F]
H2$Salary <- log(H2$Salary)
H2$Salary
Salary
#################
# Problem 1b
#################

H2.train <- H2[1:200,]
H2.test <- H2[201:nrow(H2),]
```

```
dim(H2.train)
dim(H2.test)


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

library(gbm)
set.seed(5)

shrink.lambdas=c(0.00001,0.0001,0.001,0.01,0.1,1)
training_error=rep(NA,length(shrink.lambdas))
testing_error=rep(NA,length(shrink.lambdas))
#shrink.lambdas = sl
#interaction.depth=1, n.cores=10


for (i in 1:length(shrink.lambdas)){

 #sl = shrink.lambdas[i]

 boost_hitters = gbm(Salary~., data=H2.train, distribution="gaussian",
 n.trees=1000, shrinkage = shrink.lambdas[i])

 predictions_train = predict(boost_hitters,H2.train, n.trees=1000)
 predictions_test = predict(boost_hitters,H2.test, n.trees=1000)

 training_error[i]=mean((predictions_train - H2.train$Salary)^2)
 testing_error[i]=mean((predictions_test - H2.test$Salary)^2)

 }


plot(log(shrink.lambdas),training_error)


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


plot(log(shrink.lambdas),testing_error)


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

```r
#boosting

boost_results = training_error[which.min(training_error)]


#best subset lm

library(leaps)

placeholder = regsubsets(Salary~., data=H2.train, nvmax=19)
placeholder_s = summary(placeholder)

a = placeholder_s$which[which.min(placeholder_s$cp),][2:20]

names = c(names,"Division","Salary")

placeholder.lm = lm(Salary~.,data=H2.train[,colnames(H2.train)%in%names])
subsetlm_prediction =
 predict(placeholder.lm,H2.test[,colnames(H2.train)%in%names])
bestsub = mean((subsetlm_prediction-H2.test$Salary)^2)





#best lasso

library(glmnet)

eliminate=c("League","Division","NewLeague")
reformat=model.matrix(~.,H2)[,-1]
reformat.train=reformat[1:200,]
reformat.test=reformat[201:nrow(reformat),]

fit_lasso=cv.glmnet(reformat.train[,colnames(reformat)!="Salary"],reformat.trai
 n[,"Salary"])


fit_lasso=glmnet(reformat.train[,colnames(reformat)!="Salary"],reformat.train[,
 "Salary"],lambda=fit$lambda.1se)



pred=predict(fit_lasso,reformat.test[,colnames(reformat)!="Salary"])
best.lasso=mean((pred[,1]-H2.test$Salary)^2)
```

```
#compare the test MSEs

boost_results
bestsub
best.lasso



#the lasso is the best by a really little bit on the test data, but boosting
 came in close.



#################
# Problem 1f
#################

boost_hitters_test = gbm(Salary~., data=H2.train, distribution="gaussian",
 n.trees=1000, shrinkage = shrink.lambdas[which.min(training_error)],
 interaction.depth=1, n.cores=10)

summary(boost_hitters_test)



#################
# Problem 1g
#################

library(randomForest)
set.seed(1)

bagging_hitters = randomForest(Salary~.,data=H2.train, mtry=ncol(H2.train)-1,
 importance=TRUE)
#ntree=500
#mtry=19

prediction_bag = predict(bagging_hitters,newdata=H2.test)
mean((prediction_bag-H2.test$Salary)^2)
```

```r
#############################
# Kevin Gong
# STAT W4240
# Homework 06, Problem 3
# May 05, 2014
#

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

#################
# 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/hw06")


# 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(ISLR)
library(e1071)


#################
# Problem 3a
#################

data("OJ")
OJ


set.seed(1)

rownumbers = sample(nrow(OJ))
train_set = OJ[rownumbers[1:800],]
test_set = OJ[-rownumbers[1:800],]

dim(train_set)
```

```r
dim(test_set)


#################
# Problem 3b
#################

Purchase = train_set[,1]

training = data.frame(x=train_set[,-1], y=Purchase)
svm_train = svm(y~.,data=training, kernel="linear", cost=0.01, scale=TRUE)
summary(svm_train)

??svm

#################
# Problem 3c
#################

test = data.frame(x=test_set[,-1],y=test_set[,1])
svm_test = svm(y~.,data=test, kernel="linear", cost=0.01, scale=TRUE)


train_predict = predict(svm_train,training)
test_predict = predict(svm_test,test)


train_error = sum(train_predict!=train_set[,1])/length(train_predict)
test_error = sum(test_predict!=test_set[,1])/length(test_predict)


train_error
test_error


#################
# Problem 3d
#################


#################
# Problem 3e
#################
```

```r
#################
# Problem 3f
#################


svm_train_radial = svm(y~.,data=training, kernel="radial", cost=0.01,
 scale=TRUE)
svm_test_radial = svm(y~.,data=test, kernel="radial", cost=0.01, scale=TRUE)

train_predict_radial = predict(svm_train_radial,training)
test_predict_radial = predict(svm_test_radial,test)

train_error_radial =
 sum(train_predict_radial!=train_set[,1])/length(train_predict_radial)
test_error_radial =
 sum(test_predict_radial!=test_set[,1])/length(test_predict_radial)

summary(svm_train_radial)
train_error_radial
test_error_radial


#################
# Problem 3g
#################

svm_train_poly = svm(y~.,data=training, kernel="polynomial", cost=0.01,
 scale=TRUE, degree=2)
svm_test_poly = svm(y~.,data=test, kernel="polynomial", cost=0.01, scale=TRUE,
 degree=2)

train_predict_poly = predict(svm_train_poly,training)
test_predict_poly = predict(svm_test_poly,test)

train_error_poly =
 sum(train_predict_poly!=train_set[,1])/length(train_predict_poly)
test_error_poly =
 sum(test_predict_poly!=test_set[,1])/length(test_predict_poly)

summary(svm_train_poly)
train_error_poly
test_error_poly
```

```
#############################
# Kevin Gong
# STAT W4240
# Homework 06, Problem 4
# May 05, 2014
#
# 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/hw06")
# 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)

# to get the svm function...
library(e1071)

# to load previous dtm code, etc.
source("../hw04/hw04.R")
setwd("~/Documents/academic/teaching/STAT_W4240_2014_SPRG/dropbox/Homework/hw04
 ")

#################
# Problem 4
#################

##########################################
# 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) {
```

```
    # 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.
 test))

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

##########################################
# 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)

# check if the 4 dtm datasets are correct
dim(dtm.hamilton.train)      # 35 4875
dim(dtm.hamilton.test)       # 16 4875
dim(dtm.madison.train)       # 15 4875
dim(dtm.madison.test)    # 11 4875
##########################################

##########################################
# make training and test sets w/
# y=0 if Madison; =1 if Hamilton
# & var names being dictionary words
dat.train <- as.data.frame(rbind(dtm.hamilton.train, dtm.madison.train))
dat.test <- as.data.frame(rbind(dtm.hamilton.test, dtm.madison.test))

names(dat.train) <- names(dat.test) <- as.vector(dictionary$word)

dat.train$y <- as.factor(c(rep(1, nrow(dtm.hamilton.train)), rep(0,
 nrow(dtm.madison.train))))
dat.test$y <- as.factor(c(rep(1, nrow(dtm.hamilton.test)), rep(0,
 nrow(dtm.madison.test))))
# note: as.factor() makes the y label as factor, which is helpful for svm later
 (so it will do classification)

dim(dat.train)       # 50 4876
dim(dat.test)        # 27 4876
```

```
###################################
# center and scale the data as in HW05
mean.train <- apply(dat.train[,-4876], 2, mean)        # col means of training
 x
sd.train <- apply(dat.train[,-4876], 2, sd)            # col sd of training x

x.train <- scale(dat.train[,-4876])                    # standardize training x
x.train[,sd.train==0] <- 0                             # let the var be 0 if
 its sd=0

x.test <- scale(dat.test[,-4876], center = mean.train, scale=sd.train)      #
 use training x mean & sd to standardize test x
x.test[,sd.train==0] <- 0                          # let the var be 0 if its
 sd=0

y.train <-dat.train$y
y.test <- dat.test$y


############################
# Problem 4
############################

###########
# Part a
###########

training = data.frame(x=x.train[,1:100],y=y.train)
testing = data.frame(x=x.test[,1:100],y=y.test)

svm_1 = svm(y~.,data=training, kernel="linear", cost=10, scale=FALSE)

test_prediction = predict(svm_1,testing)
test_prediction
testing$y
sum(test_prediction==testing$y)

###########
# Part b
###########

a = seq(from=5, to=100, by=5)

train_errors = rep(NA,20)
test_errors = rep(NA,20)

for(i in 1:20) {
```

```
 training_data = data.frame(x=x.train[,1:a[i]],y=y.train)
 testing_data = data.frame(x=x.test[,1:a[i]], y=y.test)

 train_prediction = predict(svm_1,training_data)
 test_prediction = predict(svm_1,testing_data)

 train_errors[i] = sum(train_prediction!=y.test)/length(train_prediction)
 test_errors[i] = sum(test_prediction!=y.test)/length(test_prediction)

}

plot(a, test_errors, col="green", pch=5, type="b", xlab="# Words", ylab="Test
 Error")
```

```
###########
# Part c
###########

svm_2 = svm(y~.,data=training, kernel="radial", cost=10, scale=FALSE)

a = seq(from=5, to=100, by=5)

train_errors = rep(NA,20)
test_errors = rep(NA,20)

for(i in 1:20) {

 training_data = data.frame(x.train[,1:a[i]],y=y.train)

 testing_data = data.frame(x=x.test[,1:a[i]], y=y.test)

 train_prediction.2 = predict(svm_2,training_data)
 test_prediction.2 = predict(svm_2,testing_data)

 train_errors[i] = sum(train_prediction.2!=y.test)/length(train_prediction.2)
 test_errors[i] = sum(test_prediction.2!=y.test)/length(test_prediction.2)
```

```
}

plot(a, test_errors, col="red", pch=5, type="b", xlab="# Words", ylab="Test
 Error")





###########
# Part d
###########


features = c("upon", "depart")

features_training = data.frame(x=x.train[,features],y=y.train)

svm_features = svm(y~., data=features_training, kernel="radial", cost=10,
 scale=FALSE)
plot(svm_features, x.train[,features])

with(svm_features,plot(x.train[,features]))
```