Kevin Gong

kg2445

HW03

Stat W4240

Section 2

**Homework 3**

**Problem #1 (James 3.7.3)**

Part a)

**The correct statement is iii (#3)**. Writing out our regression equation, we find that:

Y = (35 – 10\*GPA)\*Gender + constant

Where Gender = 0 indicates male. When GPA is high (35 – 10\*GPA <0, or GPA>3.5), female salaries will be slightly below those of males since (35 – 10\*GPA) is negative. This causes the earnings of females to be lower than those of males provided that GPA is sufficiently high, and thus the correct answer is iii (#3).

Part b)

IQ = 110, GPA = 4.0

Y = (35 – 10\*GPA)\*Gender + (50 + 20\*GPA+0.07\*IQ + 0.01\*GPA\*IQ)

Plugging in and calculating -> The predicted salary of the female with an IQ of 110 and GPA of 4.0 is **137.1 (in thousands of dollars)**.

Part c)

**False.** To test for significance of an interaction effect, we must look at the p values associated with either the T-stat or F-stat of the term. The size of the coefficient itself is not indicative of its significant.

**Problem #2 (James 3.7.4)**

Part a)

**There is not enough information to tell**. Compared to training RSS for a linear regression, training RSS for a cubic regression may be lower if x values very small (near 0) or higher if x values are very large.

Part b)

**Still not enough information to tell**. Like training RSS, test RSS varies depending on the x values, with small x values resulting in lower test RSS and vice versa.

Part c)

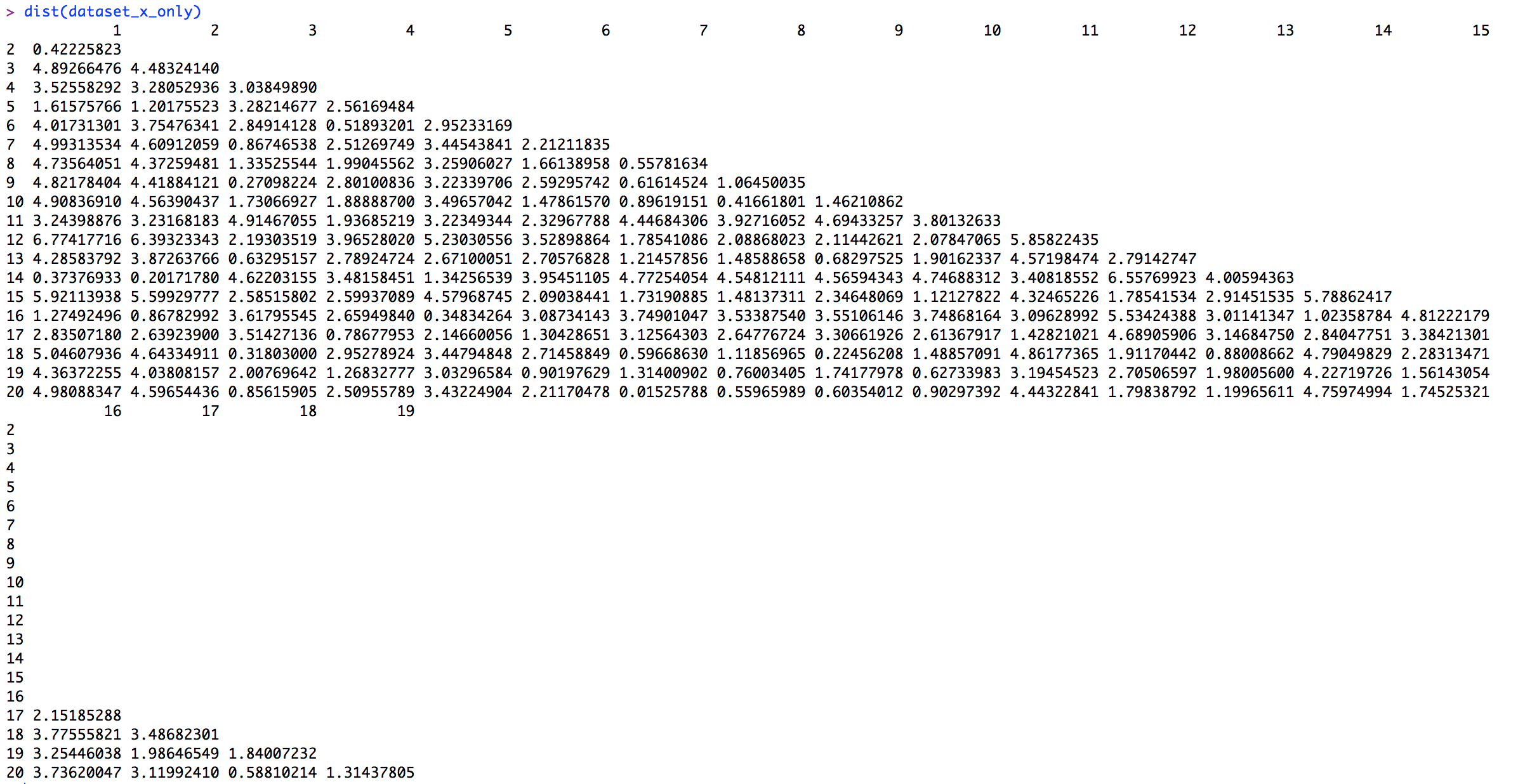
**There is not enough information to tell**. Depending on whether the X and Y data exhibits a relationship that is somewhat linear or cubic in nature, one would yield a lower training RSS than the other.

Part d)

**Still not enough information to tell**. Again, depending on whether the X and Y data exhibits a relationship that is somewhat linear or cubic in nature, one would yield a lower test RSS than the other.

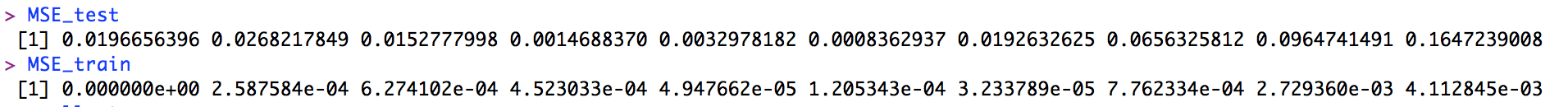
**Problem #3**

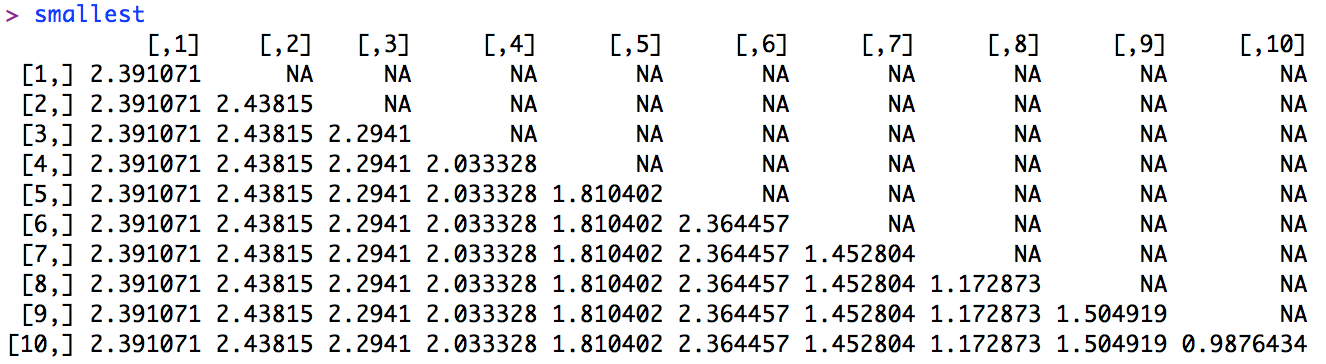
Part a)



We created a matrix with only the x1 and x2 columns before using the dist() function to generate the distances.

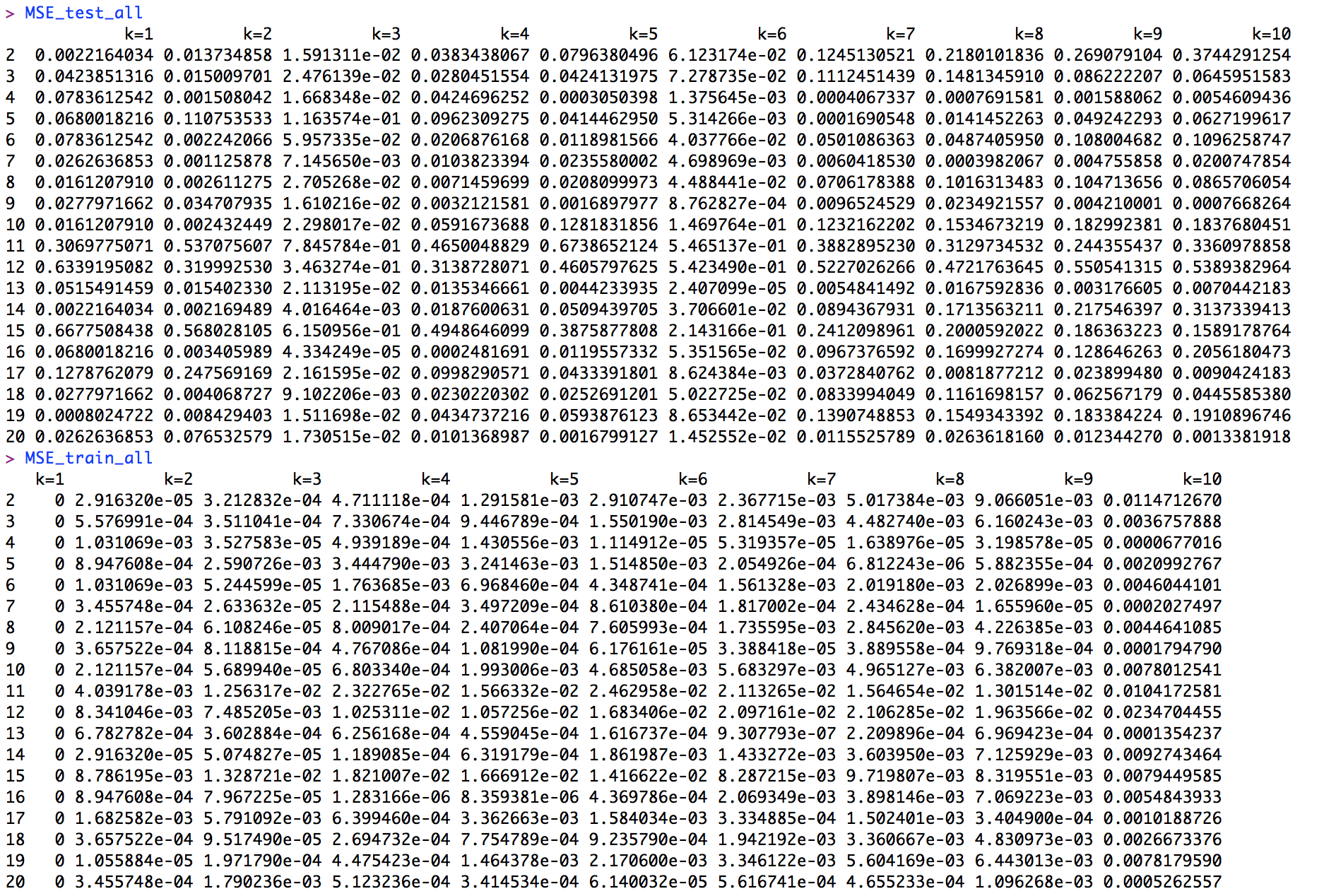
Part b)





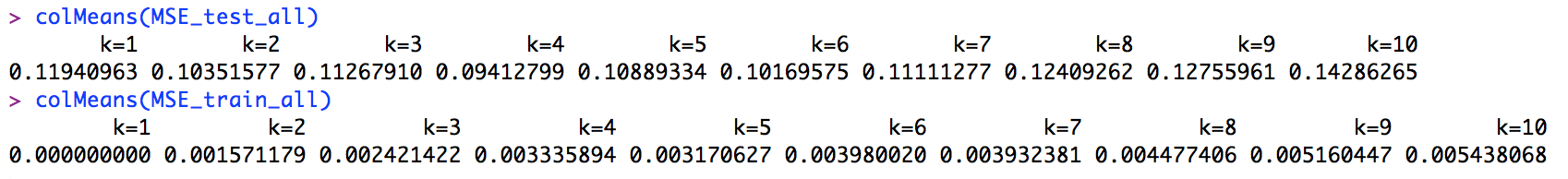
Our training and testing MSEs are show above, with the vector entry positions corresponding to values of k (eg: 0.0196 is the MSE\_test value when k=1). Our matrix “smallest” kept track of the y values being used to predict the y value of the first data point, allowing us to double check that our algorithm is working correctly. Thus, the average of each row represents y-hat for a certain k. The testing MSE was found by (y1 – yhat)^2, while the training MSE was found by [(y1 – yhat)^2]/19. As expected, our training MSE is 0 when k=1, since we include the data point itself when calculating training MSE.

Part c)

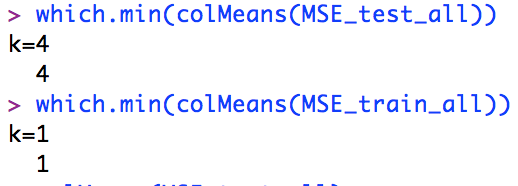


We generated our MSE test and MSE train by using nested for loops. The inner for loop controlled the k value, while the outer for loop determined the “ith” data point to be used as a testing set. After the MSEs for all 10 k values of each “ith” data point were calculated, we used rbind to append the MSEs to their respective matrix, thus generating two matrices which show training and testing MSEs for a given “ith” data point (row) and a given value of k (column). As expected, our training MSE is 0 when k=1.

Part d)



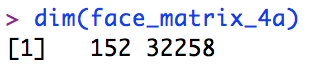
To simplify using our results in part c in order to determine the optimal value for k, we use colMeans to find the average testing and training MSEs by k value. Then, using the which.min function, we find that k=4 results in the smallest average testing MSE while k=1 results in the smallest average training MSE.

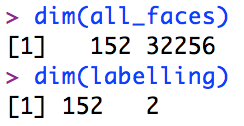


The optimal k value is k=4 since it minimizes test MSE, meaning that the predictions it generates of the y value of the testing data are on average the most accurate (lowest mean squared error) of all k values between 1 and 10. Although training MSE and testing MSE indicate different optimal k values, we should choose to base our choice of optimal k on the testing MSE since we ultimately care most about how well the model performs on our test set. Furthermore, k=1 is the optimal k value given by training MSE because all training MSEs are 0 for k=1. This is because training MSE includes the ith data point itself (unlike test MSE which excludes itself), and the data point is naturally the same as itself, thus resulting in a 0 value for training MSE.

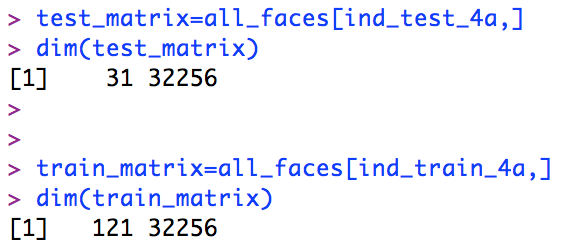
**Question 4**

Part a)

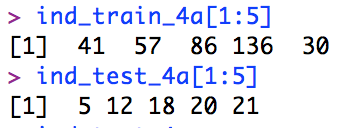




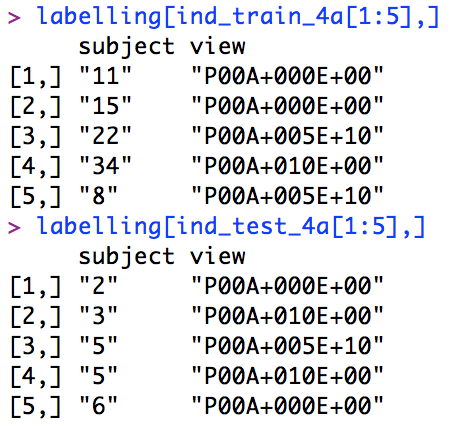
We generate a 152 x 32258 matrix of the vectors of the faces with the 4 indicated views. We can divide this matrix into 2 sets: one with the vectors of the faces, and the other with the subject numbers and views.



After dividing the data into training and testing sets, we place these sets into their own matrices, finding that the test matrix has 31 rows while the training matrix has 121 rows.

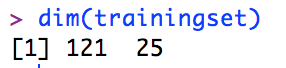


Using our training and testing index vectors, we see that the first 5 files in the training set reference rows 41, 57, 86, 136, and 30 of the original 152-row dataset, while the first 5 files of the test set reference rows 5, 12, 18,20, and 21 of the original 152-row dataset.

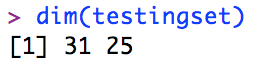


We then use our “labelling” matrix, which contains the subject and view of each of the 152 rows of our dataset, to identify the first 5 files in the training and testing sets, as seen in the above.

Part b)

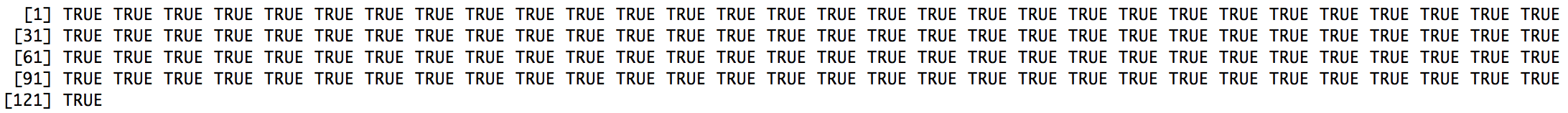


We perform PCA on our training set and use the first 25 scores to represent the data, generating a training set matrix with dimensions 121 x 25.



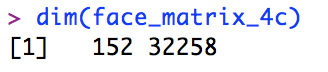
We then project our testing data into the score space so that it is also represented by the first 25 scores, giving us a test set matrix with dimensions 31 x 25.

We then use 1NN classification to identify the subject of each testing observation in the score space. We use the formula k(trainingset, testingset, cl, k=1), where trainingset and testingset reference the matrices we generated above, while cl represents the training index vector we generated in part a.

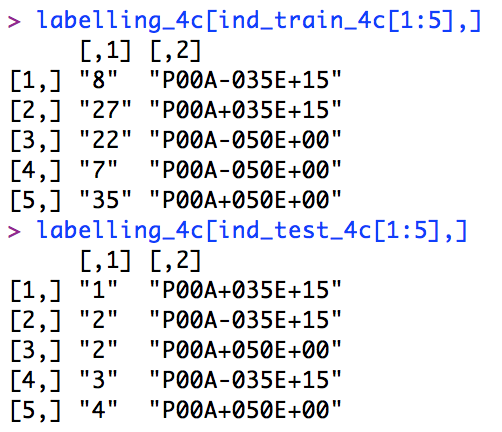


We check our Y\_test\_hat values against the subject numbers in our “cl” vector and find that all of our subjects are identified correctly (121 identified correctly, 0 identified incorrectly. This makes sense since in direct, frontal lighting (the views we loaded in part a), PCA is likely more accurate since it can better store information about and identify characteristics of faces (there is little noise to throw off PCA’s predictions).

Part c)

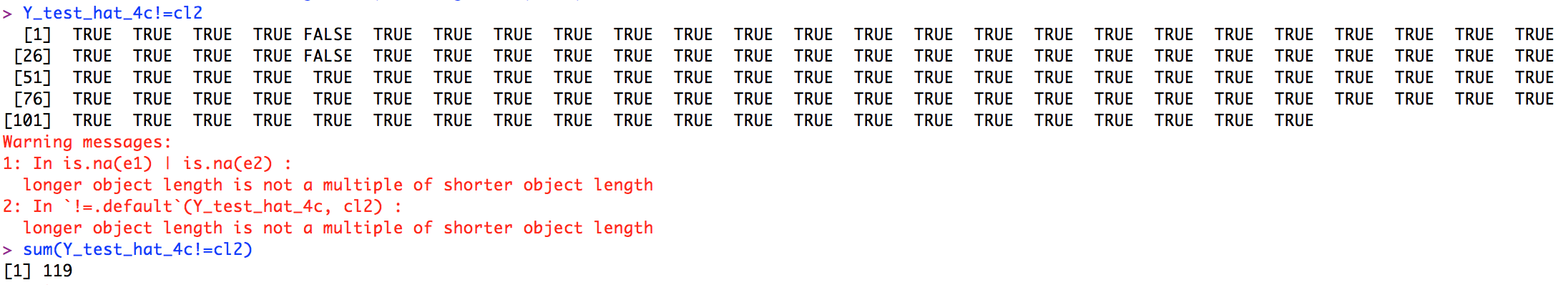


We load the 4 new views, again generating a 152 x 32258 matrix of the vectors of the faces.



We again use our “labelling” matrix to identify the first 5 files in the training and testing sets.

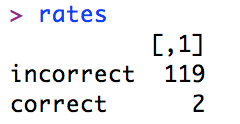
We again perform PCA on our training set and use the first 25 scores to represent the data, while also projecting the test set into the score space. This again results in a training set of dimension 121 x 25 and testing set of dimension 31 x 25.



Though our 1NN classification seems to misfire and suggest that 119 of the 121 subjects are misidentified, we know that in reality, approximately 27 of the subjects are misidentified, while the remainder (91 subjects) are identified correctly.

Part d)

We repeat our code from part c 10 times and generate 10 identical instances of the following correct and incorrect identification rates.



Though our code likely contains errors, these results tell us that PCA/kNN consistently misidentifies a significant number of subjects in these 4 lighting conditions.

Part e)

There are drastic differences the testing error rates in part b and part c. In part b, we identified all subjects correctly, while in part c, we failed to identify a significant number of subjects correctly. This tells us that PCA works better in certain lighting conditions than others. In part b, the lighting was straight on, generating little noise on the facial features and thus resulting in a very high identification accuracy rate for PCA/kNN. Meanwhile, the side lighting in part c generated a lot of shadows, noise, and obscurations of subjects’ facial features, thus greatly hindering PCA/kNN’s ability to match facial features and consequently committing a significant number of identification errors. This tells us that PCA is likely sensitive to – and thus probably easily thrown off by – noise in data and facial features.

Part f)

If we used uncropped photos, the testing error rate of PCA/kNN is likely to be very high (ie: a high number of resulting misidentifications). This is because noise created by hair, neck, clothing, and other non-facial features of a subject are likely to throw off PCA/kNN’s ability to match faces. PCA/kNN will instead try to match images based on their overall content, and this would probably render facial matching accuracy very low. This would likely be even further worsened in lighting conditions that do not illuminate a subject straight-on.

**CODE**

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

# Kevin Gong

# STAT W4240

# Homework 3 , Problem 3

# 3/5/14

#

# The following code loads the eigenfaces data and

# performs a set of simple loading and plotting functions

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

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

# 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/HW3")

# first include the relevant libraries

# note that a loading error might mean that you have to

# install the package into your R distribution.

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

# Problem 3a

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

# load the data and use dist() to get a distance matrix

#----- START YOUR CODE BLOCK HERE -----#

#import the data

dataset <- read.csv(file="hw03\_q3.csv")

head(dataset)

#set aside x1 and x2 in their own matrix

dataset\_x\_only <- dataset[,1:2]

head(dataset\_x\_only)

#compute the distances between x1 and x2 using dist()

distances <-as.matrix(dist(dataset\_x\_only))

distances

#----- END YOUR CODE BLOCK HERE -----#

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

# Problem 3b

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

#----- START YOUR CODE BLOCK HERE -----#

dataset\_y\_only <- dataset[,3]

head(dataset\_y\_only)

#separate data into testing and training sets

test <- dataset\_x\_only[1,]

test

train <- dataset\_x\_only[2:20,]

train

#allocate empty placeholder vectors and matrices for our for loop

smallest=matrix(nrow=10,ncol=10)

MSE\_test=vector()

MSE\_train=vector()

#First we calculate the training MSE. We make sure to include the point itself when doing so, hence [1:k]

for(k in 1:10){

abba <- order(distances[,1],decreasing=F)[1:k]

smallest[k,(1:k)] = dataset[abba,3]

yhat=sum(dataset[abba,3])/k

MSE\_train[k]=((dataset[1,3]-yhat)^2)/19

}

#Next we calculate the testing MSE. We make sure to avoid including the testing point itself when doing so, hence [2:(k+1)]

for(k in 1:10){

abba <- order(distances[,1],decreasing=F)[2:(k+1)]

smallest[k,(1:k)] = dataset[abba,3]

yhat=sum(dataset[abba,3])/k

MSE\_test[k]=(dataset[1,3]-yhat)^2

}

#examine the results. The "smallest" matrix allows us to check the y values used to approximate our test y for each k value.

MSE\_test

MSE\_train

smallest

#----- END YOUR CODE BLOCK HERE -----#

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

# Problem 3c

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

#----- START YOUR CODE BLOCK HERE -----#

#allocate empty placeholder vectors and matrices for our for loops

MSE\_test=vector()

MSE\_train=vector()

smallest=matrix(nrow=190,ncol=10)

MSE\_test\_all=NULL

MSE\_train\_all=NULL

#We calculate the training MSE. We make sure to include the point itself when doing so, hence [1:k]

for(i in 2:20){

MSE\_train=vector()

for(k in 1:10){

abba <- order(distances[,i],decreasing=F)[1:k]

smallest[k,(1:k)] = dataset[abba,3]

yhat=sum(dataset[abba,3])/k

MSE\_train[k]=((dataset[i,3]-yhat)^2)/19

}

MSE\_train\_all=rbind(MSE\_train\_all,MSE\_train)

}

#We calculate the testing MSE. We make sure to avoid including the testing point itself when doing so, hence [2:(k+1)]

for(i in 2:20){

MSE\_test=vector()

MSE\_train=vector()

for(k in 1:10){

abba <- order(distances[,i],decreasing=F)[2:(k+1)]

smallest[k,(1:k)] = dataset[abba,3]

yhat=sum(dataset[abba,3])/k

MSE\_test[k]=(dataset[i,3]-yhat)^2

}

MSE\_test\_all=rbind(MSE\_test\_all,MSE\_test)

}

#examine the resulting MSEs

MSE\_test\_all

MSE\_train\_all

#remane the column names to the different values of k

colnames(MSE\_test\_all) <-c("k=1","k=2","k=3","k=4","k=5","k=6","k=7","k=8","k=9","k=10")

colnames(MSE\_train\_all) <-c("k=1","k=2","k=3","k=4","k=5","k=6","k=7","k=8","k=9","k=10")

#rename the row names to the ith data point

rownames(MSE\_test\_all)<-c(2:20)

rownames(MSE\_train\_all)<-c(2:20)

#examine renamed MSE matrices

MSE\_test\_all

MSE\_train\_all

#average the mean MSEs by k value and see which k value has the lowest average mean squared error

colMeans(MSE\_test\_all)

colMeans(MSE\_train\_all)

which.min(colMeans(MSE\_test\_all))

which.min(colMeans(MSE\_train\_all))

#----- END YOUR CODE BLOCK HERE -----#

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

# End of Script

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

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

# Kevin Gong

# STAT W4240

# Homework 3 , Problem 4

# 3/5/14

#

# The following code loads the eigenfaces data and

# performs a set of simple loading and plotting functions

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

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

# 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/HW3")

# first include the relevant libraries

# note that a loading error might mean that you have to

# install the package into your R distribution.

library(pixmap)

library(class)

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

# Problem 4a

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

views\_4a = c('P00A+000E+00', 'P00A+005E+10', 'P00A+005E-10', 'P00A+010E+00' )

# load the data and save it as a matrix with the name face\_matrix\_4a

#----- START YOUR CODE BLOCK HERE -----#

# the list of pictures

dir\_list\_1 = dir(path="CroppedYale/",all.files=FALSE)

views\_4a = c('P00A+000E+00', 'P00A+005E+10', 'P00A+005E-10', 'P00A+010E+00' )

# preallocate an empty list

this\_face\_row = vector()

# initialize an empty matrix of faces data

face\_matrix\_4a = vector()

pic\_list = c( 01,02,03,04,05,06,07,08,09,10,11,12,13,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39)

#generating a matrix of photos of the 4 desired views

for (i in 1:38) {

placeholder\_matrix <- NULL

for (j in 1:4) {

filename= sprintf("CroppedYale/%s/%s\_%s.pgm",dir\_list\_1[i],dir\_list\_1[i],views\_4a[j])

print(filename)

face=read.pnm(file=filename)

pic\_data <-getChannels(face)

pic\_data\_vec <- as.vector(pic\_data)

placeholder\_matrix=rbind(placeholder\_matrix,c(pic\_data\_vec,i,views\_4a[j]))

}

face\_matrix\_4a=rbind(face\_matrix\_4a,placeholder\_matrix)

}

dim(face\_matrix\_4a)

all\_faces <- face\_matrix\_4a[,1:32256]

labelling <- face\_matrix\_4a[,32257:32258]

dim(all\_faces)

dim(labelling)

colnames(labelling) <-c("subject","view")

labelling

#----- END YOUR CODE BLOCK HERE -----#

# Get the size of the matrix for use later

fm\_4a\_size = dim(face\_matrix\_4a)

# Use 4/5 of the data for training, 1/5 for testing

ntrain\_4a = floor(fm\_4a\_size[1]\*4/5) # Number of training obs

ntest\_4a = fm\_4a\_size[1]-ntrain\_4a # Number of testing obs

set.seed(1) # Set pseudo-random numbers so everyone gets the same output

ind\_train\_4a = sample(1:fm\_4a\_size[1],ntrain\_4a) # Training indices

ind\_test\_4a = c(1:fm\_4a\_size[1])[-ind\_train\_4a] # Testing indices

#----- START YOUR CODE BLOCK HERE -----#

ind\_train\_4a

ind\_test\_4a

test\_matrix=all\_faces[ind\_test\_4a,]

dim(test\_matrix)

train\_matrix=all\_faces[ind\_train\_4a,]

dim(train\_matrix)

test\_id = labelling[ind\_test\_4a,]

dim(test\_id)

train\_id = labelling[ind\_train\_4a,]

dim(train\_id)

#find the rows referenced by the first 5 files of the training and testing sets

ind\_train\_4a[1:5]

ind\_test\_4a[1:5]

#find the first 5 files of the training and testing sets

labelling[ind\_train\_4a[1:5],]

labelling[ind\_test\_4a[1:5],]

#----- END YOUR CODE BLOCK HERE -----#

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

# Problem 4b

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

#----- START YOUR CODE BLOCK HERE -----#

#training data first

mean\_face <- colMeans(train\_matrix)

means\_mat=as.matrix(mean\_face)

#reshape this matrix into 192x168

means\_mat\_ref=matrix(means\_mat,192,168)

means\_face=pixmapGrey(means\_mat\_ref)

plot(means\_face)

pic\_mat\_size = dim(train\_matrix)

pic\_mat\_centered = mat.or.vec(pic\_mat\_size[1],pic\_mat\_size[2])

#subtract mean face from matrix

for (i in 1:pic\_mat\_size[1]){

pic\_mat\_centered[i,] = train\_matrix[i,] - mean\_face

}

dim(pic\_mat\_centered)

pca\_results <- prcomp(pic\_mat\_centered)

dim(pca\_results)

dim(as.matrix(loading\_temp))

dim(pic\_mat\_centered)

num\_comp\_mod = length(pca\_results$x[,1])

scores\_query = mat.or.vec(num\_comp\_mod,1)

trainingset=NULL

for (i in 1:121){

placeholder = NULL

for(j in 1:25){

loading\_temp = pca\_results$rotation[,j]

scores\_query = pic\_mat\_centered[i,] %\*% loading\_temp

placeholder = cbind(placeholder,scores\_query)

}

trainingset=rbind(trainingset,placeholder)

}

dim(trainingset)

mean\_face\_test <- colMeans(test\_matrix)

means\_mat\_test=as.matrix(mean\_face\_test)

#reshape this matrix into 192x168

means\_mat\_ref\_test=matrix(means\_mat\_test,192,168)

means\_face\_test=pixmapGrey(means\_mat\_ref\_test)

plot(means\_face\_test)

pic\_mat\_size\_test = dim(test\_matrix)

pic\_mat\_centered\_test = mat.or.vec(pic\_mat\_size\_test[1],pic\_mat\_size\_test[2])

# I am using a loop, but one could use apply()

for (i in 1:pic\_mat\_size\_test[1]){

pic\_mat\_centered\_test[i,] = test\_matrix[i,] - mean\_face

}

dim(pic\_mat\_centered\_test)

pca\_results\_test <- prcomp(pic\_mat\_centered\_test)

dim(as.matrix(pca\_results\_test))

dim(as.matrix(loading\_temp\_test))

dim(pic\_mat\_centered\_test)

num\_comp\_mod\_test = length(pca\_results\_test$x[,1])

scores\_query\_test = mat.or.vec(num\_comp\_mod\_test,1)

testingset=NULL

for (i in 1:31){

placeholder\_test = NULL

for(j in 1:25){

loading\_temp\_test = pca\_results\_test$rotation[,j] #note this is the same as training (I think it's supposed to be like this)

scores\_query\_test = pic\_mat\_centered\_test[i,] %\*% loading\_temp\_test

placeholder\_test = cbind(placeholder\_test,scores\_query\_test)

}

testingset=rbind(testingset,placeholder\_test)

}

dim(testingset)

#this works?

cl <- row.names(data.frame(trainingset))

cl2 <-factor(ind\_train\_4a)

length(indicator)

Y\_test\_hat <- knn(trainingset, testingset, cl2, k=1)

Y\_test\_hat!=cl2

sum(Y\_test\_hat!=cl)

#----- END YOUR CODE BLOCK HERE -----#

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

# Problem 4c

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

# Use different lighting conditions

views\_4c = c('P00A-035E+15', 'P00A-050E+00', 'P00A+035E+15', 'P00A+050E+00')

# load your data and save the images as face\_matrix\_4c

#----- START YOUR CODE BLOCK HERE -----#

# the list of pictures

dir\_list\_1 = dir(path="CroppedYale/",all.files=FALSE)

views\_4c = c('P00A-035E+15', 'P00A-050E+00', 'P00A+035E+15', 'P00A+050E+00')

# preallocate an empty list

this\_face\_row = vector()

# initialize an empty matrix of faces data

face\_matrix\_4c = vector()

pic\_list = c( 01,02,03,04,05,06,07,08,09,10,11,12,13,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39)

#generating a matrix of photos of the 4 desired views

for (i in 1:38) {

placeholder\_matrix <- NULL

for (j in 1:4) {

filename= sprintf("CroppedYale/%s/%s\_%s.pgm",dir\_list\_1[i],dir\_list\_1[i],views\_4c[j])

print(filename)

face=read.pnm(file=filename)

pic\_data <-getChannels(face)

pic\_data\_vec <- as.vector(pic\_data)

placeholder\_matrix=rbind(placeholder\_matrix,c(pic\_data\_vec,i,views\_4c[j]))

}

face\_matrix\_4c=rbind(face\_matrix\_4c,placeholder\_matrix)

}

dim(face\_matrix\_4c)

all\_faces\_4c <- face\_matrix\_4c[,1:32256]

labelling\_4c <- face\_matrix\_4c[,32257:32258]

#----- END YOUR CODE BLOCK HERE -----#

fm\_4c\_size = dim(face\_matrix\_4c)

# Use 4/5 of the data for training, 1/5 for testing

ntrain\_4c = floor(fm\_4c\_size[1]\*4/5)

ntest\_4c = fm\_4c\_size[1]-ntrain\_4c

set.seed(2) # Set pseudo-random numbers

# You are resetting so that if you have used a random number in between the last use of sample(), you will still get the same output

ind\_train\_4c = sample(1:fm\_4c\_size[1],ntrain\_4c)

ind\_test\_4c = c(1:fm\_4c\_size[1])[-ind\_train\_4c]

#----- START YOUR CODE BLOCK HERE -----#

ind\_train\_4c

ind\_test\_4c

test\_matrix\_4c=all\_faces\_4c[ind\_test\_4c,]

dim(test\_matrix\_4c)

train\_matrix\_4c=all\_faces\_4c[ind\_train\_4c,]

dim(train\_matrix\_4c)

test\_id\_4c = labelling\_4c[ind\_test\_4c,]

dim(test\_id\_4c)

train\_id\_4c = labelling\_4c[ind\_train\_4c,]

dim(train\_id\_4c)

#find the rows referenced by the first 5 files of the training and testing sets

ind\_train\_4c[1:5]

ind\_test\_4c[1:5]

#find the first 5 files of the training and testing sets

labelling\_4c[ind\_train\_4c[1:5],]

labelling\_4c[ind\_test\_4c[1:5],]

#training data first

mean\_face\_4c <- colMeans(train\_matrix\_4c)

means\_mat\_4c=as.matrix(mean\_face\_4c)

#reshape this matrix into 192x168

means\_mat\_ref\_4c=matrix(means\_mat\_4c,192,168)

means\_face\_4c=pixmapGrey(means\_mat\_ref\_4c)

plot(means\_face\_4c)

pic\_mat\_size\_4c = dim(train\_matrix\_4c)

pic\_mat\_centered\_4c = mat.or.vec(pic\_mat\_size\_4c[1],pic\_mat\_size\_4c[2])

#subtract mean face from matrix

for (i in 1:pic\_mat\_size\_4c[1]){

pic\_mat\_centered\_4c[i,] = train\_matrix\_4c[i,] - mean\_face\_4c

}

dim(pic\_mat\_centered\_4c)

pca\_results\_4c <- prcomp(pic\_mat\_centered\_4c)

dim(pca\_results\_4c)

dim(as.matrix(loading\_temp\_4c))

dim(pic\_mat\_centered\_4c)

num\_comp\_mod\_4c = length(pca\_results\_4c$x[,1])

scores\_query\_4c = mat.or.vec(num\_comp\_mod\_4c,1)

trainingset\_4c=NULL

for (i in 1:121){

placeholder\_4c = NULL

for(j in 1:25){

loading\_temp\_4c = pca\_results\_4c$rotation[,j]

scores\_query\_4c = pic\_mat\_centered\_4c[i,] %\*% loading\_temp\_4c

placeholder\_4c = cbind(placeholder\_4c,scores\_query\_4c)

}

trainingset\_4c=rbind(trainingset\_4c,placeholder\_4c)

}

dim(trainingset\_4c)

mean\_face\_test\_4c <- colMeans(test\_matrix\_4c)

means\_mat\_test\_4c=as.matrix(mean\_face\_test\_4c)

#reshape this matrix into 192x168

means\_mat\_ref\_test\_4c=matrix(means\_mat\_test\_4c,192,168)

means\_face\_test\_4c=pixmapGrey(means\_mat\_ref\_test\_4c)

plot(means\_face\_test\_4c)

pic\_mat\_size\_test\_4c = dim(test\_matrix\_4c)

pic\_mat\_centered\_test\_4c = mat.or.vec(pic\_mat\_size\_test\_4c[1],pic\_mat\_size\_test\_4c[2])

# I am using a loop, but one could use apply()

for (i in 1:pic\_mat\_size\_test\_4c[1]){

pic\_mat\_centered\_test\_4c[i,] = test\_matrix\_4c[i,] - mean\_face\_4c

}

dim(pic\_mat\_centered\_test\_4c)

pca\_results\_test\_4c <- prcomp(pic\_mat\_centered\_test\_4c)

dim(as.matrix(pca\_results\_test\_4c))

dim(as.matrix(loading\_temp\_test\_4c))

dim(pic\_mat\_centered\_test\_4c)

num\_comp\_mod\_test\_4c = length(pca\_results\_test\_4c$x[,1])

scores\_query\_test\_4c = mat.or.vec(num\_comp\_mod\_test\_4c,1)

testingset\_4c=NULL

for (i in 1:31){

placeholder\_test\_4c = NULL

for(j in 1:25){

loading\_temp\_test\_4c = pca\_results\_test\_4c$rotation[,j]

scores\_query\_test\_4c = pic\_mat\_centered\_test\_4c[i,] %\*% loading\_temp\_test\_4c

placeholder\_test\_4c = cbind(placeholder\_test\_4c,scores\_query\_test\_4c)

}

testingset\_4c=rbind(testingset\_4c,placeholder\_test\_4c)

}

dim(testingset\_4c)

#this works?

cl <- row.names(data.frame(testingset\_4c))

cl2 <- factor(ind\_train\_4c)

length(indicator)

Y\_test\_hat\_4c <- knn(trainingset\_4c, testingset\_4c, cl, k=1)

Y\_test\_hat\_4c!=cl

sum(Y\_test\_hat\_4c!=cl2)

#----- END YOUR CODE BLOCK HERE -----#

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

# Problem 4d

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

#----- START YOUR CODE BLOCK HERE -----#

#REPEAT 10 TIMES

fm\_4c\_size = dim(face\_matrix\_4c)

# Use 4/5 of the data for training, 1/5 for testing

ntrain\_4c = floor(fm\_4c\_size[1]\*4/5)

ntest\_4c = fm\_4c\_size[1]-ntrain\_4c

set.seed(2) # Set pseudo-random numbers

# You are resetting so that if you have used a random number in between the last use of sample(), you will still get the same output

ind\_train\_4c = sample(1:fm\_4c\_size[1],ntrain\_4c)

ind\_test\_4c = c(1:fm\_4c\_size[1])[-ind\_train\_4c]

test\_matrix\_4c=all\_faces\_4c[ind\_test\_4c,]

train\_matrix\_4c=all\_faces\_4c[ind\_train\_4c,]

test\_id\_4c = labelling\_4c[ind\_test\_4c,]

train\_id\_4c = labelling\_4c[ind\_train\_4c,]

#training data first

mean\_face\_4c <- colMeans(train\_matrix\_4c)

means\_mat\_4c=as.matrix(mean\_face\_4c)

#reshape this matrix into 192x168

means\_mat\_ref\_4c=matrix(means\_mat\_4c,192,168)

pic\_mat\_size\_4c = dim(train\_matrix\_4c)

pic\_mat\_centered\_4c = mat.or.vec(pic\_mat\_size\_4c[1],pic\_mat\_size\_4c[2])

#subtract mean face from matrix

for (i in 1:pic\_mat\_size\_4c[1]){

pic\_mat\_centered\_4c[i,] = train\_matrix\_4c[i,] - mean\_face\_4c

}

pca\_results\_4c <- prcomp(pic\_mat\_centered\_4c)

num\_comp\_mod\_4c = length(pca\_results\_4c$x[,1])

scores\_query\_4c = mat.or.vec(num\_comp\_mod\_4c,1)

trainingset\_4c=NULL

for (i in 1:121){

placeholder\_4c = NULL

for(j in 1:25){

loading\_temp\_4c = pca\_results\_4c$rotation[,j]

scores\_query\_4c = pic\_mat\_centered\_4c[i,] %\*% loading\_temp\_4c

placeholder\_4c = cbind(placeholder\_4c,scores\_query\_4c)

}

trainingset\_4c=rbind(trainingset\_4c,placeholder\_4c)

}

mean\_face\_test\_4c <- colMeans(test\_matrix\_4c)

means\_mat\_test\_4c=as.matrix(mean\_face\_test\_4c)

#reshape this matrix into 192x168

means\_mat\_ref\_test\_4c=matrix(means\_mat\_test\_4c,192,168)

pic\_mat\_size\_test\_4c = dim(test\_matrix\_4c)

pic\_mat\_centered\_test\_4c = mat.or.vec(pic\_mat\_size\_test\_4c[1],pic\_mat\_size\_test\_4c[2])

for (i in 1:pic\_mat\_size\_test\_4c[1]){

pic\_mat\_centered\_test\_4c[i,] = test\_matrix\_4c[i,] - mean\_face\_4c

}

pca\_results\_test\_4c <- prcomp(pic\_mat\_centered\_test\_4c)

num\_comp\_mod\_test\_4c = length(pca\_results\_test\_4c$x[,1])

scores\_query\_test\_4c = mat.or.vec(num\_comp\_mod\_test\_4c,1)

testingset\_4c=NULL

for (i in 1:31){

placeholder\_test\_4c = NULL

for(j in 1:25){

loading\_temp\_test\_4c = pca\_results\_test\_4c$rotation[,j]

scores\_query\_test\_4c = pic\_mat\_centered\_test\_4c[i,] %\*% loading\_temp\_test\_4c

placeholder\_test\_4c = cbind(placeholder\_test\_4c,scores\_query\_test\_4c)

}

testingset\_4c=rbind(testingset\_4c,placeholder\_test\_4c)

}

#this works?

cl2 <- factor(ind\_train\_4c)

Y\_test\_hat\_4c <- knn(trainingset\_4c, testingset\_4c, cl2, k=1)

Y\_test\_hat\_4c!=cl2

incorrect <- sum(Y\_test\_hat\_4c!=cl2)

correct <- sum(Y\_test\_hat\_4c==cl2)

rates <- matrix(c(incorrect, correct))

rownames(rates) <-c("incorrect","correct")

rates

#----- END YOUR CODE BLOCK HERE -----#

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

# End of Script

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