Simon Chaisouang

Assignment 2:

Part A

(a) For the training dataset the values of n and p, correspond to 59 and 401. For OLS regression to be applied here it requires that all p columns in the X matrix and the intercept column are linearly independent with each other.

However, for any matrix the number of linearly independent rows equal the number of linearly independent columns. Therefore, performing OLS regression is not possible since n < p + 1.

(b) After running PCR on the training data, we obtain the following data:

(i) Below is a table of ELOV values corresponding to different values of k (Principal Components):

|  |  |
| --- | --- |
| k | ELOV |
| 1 | 0.289 |
| 2 | 0.114 |
| 3 | 0.063 |

Since we are allowing are allowing up to a 10% loss of variability, the number of principal components (k) we will need is 3.

(ii) We obtain the following regression coefficients for PCR predictors:

(iii) The following R2 score is obtained as:

This shows the PCR predictors were able to explain about 97% of the overall variation in y.

(c) Now we perform PLS1 regression using the same number of predictors as used in the previous part.

(i) At each step the maximum covariance is obtained:

|  |  |
| --- | --- |
| Step | Max Covariance |
| 1 | 9.351 |
| 2 | 8.581 |
| 3 | 4.177 |

(ii) The R2 score is obtained after each step:

|  |  |
| --- | --- |
| Step | R2 Score |
| 1 | 0.312 |
| 2 | 0.779 |
| 3 | 0.976 |

(iii) We obtain the following fitted regression equation as follows:

(iv) The estimated regression coefficient of in the simple linear regression of X(0) 45 on the first PLS predictor is -0.554.The estimated coefficient of X(1) 260 in the expression of second PLS predictor is -0.143.

(d)

|  |  |  |
| --- | --- | --- |
| Its true Octane Number | Its predicted Octane Number using PCR | Its predicted Octane Number using PLS1 regression |
| 83.4 | 83.54266 | 83.59506 |

Part B

#### PCR Code

regression\_data =

read.table("Gasoline\_data.txt",header=TRUE)

## Read the names of variables in the data if the file includes variable names

print(colnames(regression\_data))

## Set y to be the column that you want to have as your response variable

y = matrix(regression\_data[ , 1],ncol=1)

## Start creating the X matrix

X = regression\_data[, c(2:402)]

X = as.matrix(X)

## Exclude 4th observation from training data to be used as test data for prediction

X\_new = X[4, , drop=FALSE]

Y\_new = y[4, , drop=FALSE] #Obtain true octance number

X = X[-4, ,drop=FALSE]

y = y[-4, , drop=FALSE]

## Calculate the value of n

n = length(y)

p = ncol(X)

scale\_X = array(1,p)

#Rescaling Variables

for (j in 1:p)

{

scale\_X[j] = sd(X[ ,j])

X[ ,j] = X[ ,j]/scale\_X[j]

}

## Performing PCR algorithm by doing PCA

## Compute the estimate of mu by sample mean

muX\_hat = colMeans(X)

## Since n < p, you cannot compute the eigenvectors of Sigma\_hat in a direct manner, instead define B such that S = B%\*%B\_transpose

B = matrix(0,p,n)

for (i in 1:n)

{

B[,i] = X[i, ] - muX\_hat

}

## Finding eigenvectors of Sigma\_hat

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

## As n < p, use the following:

B\_transpose\_B = t(B)%\*%B

eigen\_results = eigen(B\_transpose\_B,only.values=F)

sum\_eigs = sum(diag(B\_transpose\_B))

# Finding value of k based on ELOV threshold of 10%

ELOV = array(0,n)

for (i in 1:n)

{

ELOV[i] = 1 - sum(eigen\_results$values[1:i])/sum\_eigs

print(paste(i," ",ELOV[i],sep=""))

}

## k is set to 3 based on information found

k = 3

eigen\_kvalues = eigen\_results$values[1:k]

eigen\_kvectors = eigen\_results$vectors[,1:k]

eigvec\_Sigma\_hat\_k = matrix(0,p,k)

for (i in 1:k) eigvec\_Sigma\_hat\_k[ ,i] =

c(1/sqrt(eigen\_kvalues[i]))\*(B%\*%matrix(eigen\_kvectors[,i],ncol=1))

Ak\_hat = t(eigvec\_Sigma\_hat\_k)

## create the matrix of PC scores

PC\_representation = matrix(0,n,k)

for (i in 1:n) PC\_representation[i, ] = Ak\_hat%\*%( matrix( X[i, ] - muX\_hat,

ncol=1) )

## Now regress y on PC scores as in PCR

output\_PCR = lm(y~PC\_representation)

betahat\_z = matrix(output\_PCR$coefficients,ncol=1)

## how well the k PCR predictors explained y ?

## check R2

print(paste("k = ", k, ", R2 = ", summary.lm(output\_PCR)$r.squared,sep=""))

## prediction for a future observation:

## Convert X\_new from matrix to a vector

X\_new = as.vector(X\_new)

## rescale using the scale of training data

for (j in 1:p) X\_new[j] = X\_new[j]/scale\_X[j]

## create the vector of PC scores for this new observation

PC\_scores\_new = Ak\_hat%\*%( matrix( X\_new - muX\_hat, ncol=1) )

## remember that first member of betahat\_z is the intercept

y\_pred\_new = betahat\_z[1] + sum(betahat\_z[2:(k+1)]\*PC\_scores\_new)

print(y\_pred\_new)

#### PLS1 Code

regression\_data =

read.table("Gasoline\_data.txt",header=TRUE)

## Read the names of variables in the data if the file includes variable names

print(colnames(regression\_data))

## Set y to be the column that you want to have as your response variable

y = matrix(regression\_data[ , 1],ncol=1)

## Start creating the X matrix

X = regression\_data[, c(2:402)]

X = as.matrix(X)

## Exclude 4th observation from training data to be used as test data for prediction

X\_new = X[4, , drop=FALSE]

X = X[-4, ,drop=FALSE]

y = y[-4, , drop=FALSE]

## Calculate the value of n

n = length(y)

p = ncol(X)

scale\_X = array(1,p)

#Rescaling Variables

for (j in 1:p)

{

scale\_X[j] = sd(X[ ,j])

X[ ,j] = X[ ,j]/scale\_X[j]

}

## Using same number of predictors as previously used in PCR

k = 3

step = 0

## Computing quantities for new observation later

muX\_hat = colMeans(X)

muY\_hat = mean(y)

## calculate TSS

TSS = sum((y - muY\_hat)^2.0)

## Initialize PLS1 by centering the response and covariates

X\_temp = matrix(0,n,p)

for (j in 1:p) X\_temp[ ,j] = X[ ,j] - muX\_hat[j] ## subtracting sample mean from all x-variables

y\_temp = y - muY\_hat

## create objects to store necessary outputs

B\_hat = matrix(0,p,k)

R\_hat = matrix(0, p, k)

q\_hat = matrix(0,k,1)

## define a function to for covariance vector estimation at each step

Sigma\_XY\_Estimate\_Function =

function(X\_dummy,y\_dummy)

{c(1/(n-1))\*((t(X\_dummy)-colMeans(X\_dummy))%\*%(y\_dummy - mean(y\_dummy)))}

for (step in 1:k)

{

## First estimate covariance between X\_temp and y\_temp

Sigma\_XY\_hat = Sigma\_XY\_Estimate\_Function(X\_temp,y\_temp)

## Now use that estimate to construct the b vector

b\_hat = Sigma\_XY\_hat/sqrt(sum(Sigma\_XY\_hat^2))

## calculate the maximized covariance: b\_hat^T \times Sigma\_hat\_(Xy)

print(paste("Max Covariance at step ",step," = ",sum(b\_hat\*Sigma\_XY\_hat),sep=""))

## Now using the b\_hat vector to construct the PLS Predictor t

t\_predictor = X\_temp%\*%b\_hat

## Now running p separate simple linear regressions (no intercept) to determine coefficients r\_hat (vector)

r\_hat = array(0,p)

for (j in 1:p)

{

out = lm(X\_temp[ ,j]~0+t\_predictor) ## without intercept

r\_hat[j] = out$coefficients

## update the j-th column of X\_temp using residual vector from above equation

X\_temp[ ,j] = out$residuals

}

## Now run one regression (no intercept) to determine coefficient q\_hat (scalar)

out = lm(y\_temp~0+t\_predictor) ## without intercept

q\_hat\_value = out$coefficients

## update y\_temp using residual vector from above equation

y\_temp = out$residuals

## checking R2

print(paste("Step = ", step, ", R2 = ", 1 - sum(y\_temp^2)/TSS,sep=""))

## Now store the necessary outputs for use in future prediction

B\_hat[ , step] = b\_hat

R\_hat[ , step] = r\_hat

q\_hat[step] = q\_hat\_value

}

## Obtain values of estimated regression coefficients for part c-iv

R\_hat[45, 1]

R\_hat[260, 2]

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

## prediction for a future observation:

## Convert X\_new from matrix to a vector

X\_new = as.vector(X\_new)

## rescale using the scale of training data

for (j in 1:p) X\_new[j] = X\_new[j]/scale\_X[j]

## starting the prediction

step = 0

## subtract training data sample mean for all x-variables

X\_temp\_new = array(0,p)

for (j in 1:p) X\_temp\_new[j] = X\_new[j] - muX\_hat[j] ## subtracting training data sample mean for all variables

## Need to create a storage for PLS predictor values for new observation

t\_predictor\_new = array(0,k)

for (step in 1:k)

{

## Using the appropriate column of B\_hat to construct the PLS Predictor for the new observation

t\_predictor\_new[step] = sum(X\_temp\_new\*B\_hat[ ,step])

for (j in 1:p)

{

## Using appropriate coefficient from R\_hat matrix to create the deflated version of the j-th entry of X\_temp\_new as the residual

X\_temp\_new[j] = X\_temp\_new[j] - R\_hat[j,step]\*t\_predictor\_new[step]

}

}

q\_hat = as.vector(q\_hat) # Convert q\_hat to a numeric vector

y\_pred\_new = muY\_hat + sum(q\_hat\*t\_predictor\_new)

print(y\_pred\_new)