STAT 542: Homework 7

Spring 2022, by Kamila Makhambetova (kamilam3)

Due: Thursday, March 24, 11:59 PM CT

Instruction

Students are encouraged to work together on homework. However, sharing, copying, or providing any part of a homework solution or code is an infraction of the University's rules on Academic Integrity. Any violation will be punished as severely as possible. Final submissions must be uploaded to compass2g. No email or hardcopy will be accepted. For **late submission policy and grading rubrics** (https://teazrq.github.io/stat542/homework.html), please refer to the course website.

- What is expected for the submission to **Gradescope**
    - You are required to submit one rendered **PDF** file `HWx_yourNetID.pdf`. For example, `HW01_rqzhu.pdf`. Please note that this must be a `.pdf` file generated by a `.Rmd` file. `.html` format cannot be accepted.
    - Please follow the instructions on Gradescope to select corresponding PDF pages for each question.
- Please note that your homework file is a **PDF** report instead of a messy collection of R codes. This report should **include**:
    - Your Name and NetID. (Replace `Ruoqing Zhu (rqzhu)` by your name and NetID if you are using this template).
    - Make all of your `R` code chunks visible for grading.
    - Relevant outputs from your `R` code chunks that support your answers.
    - Provide clear answers or conclusions for each question. For example, you could start with `Answer: I fit SVM with the following choice of tuning parameters ...`
    - Many assignments require your own implementation of algorithms. **Basic comments are strongly encouraged** to explain the logic to our graders. However, line-by-line code comments are unnecessary.
- Requirements regarding the `.Rmd` file.
    - You do **NOT** need to submit `Rmd` files. However, your PDF file should be rendered directly from it.
    - Make sure that you **set random seeds** for simulation or randomized algorithms so that the results are reproducible. If a specific seed number is not provided in the homework, you can consider using your NetID.
    - For some questions, there will be restrictions on what packages/functions you can use. Please read the requirements carefully. As long as the question does not specify such restrictions, you can use anything.

Question 1 [35 Points] Local Linear Regression

We have implemented the Nadaraya-Watson kernel estimator in HW 6. In this question, we will investigate a local linear regression:

$$\widehat{f}\ (x) = \widehat{\beta}_0\ (x) + \widehat{\beta}_1\ (x)\, x,$$

where x is a testing point. Local coefficients $\widehat{\beta}_r\ (x)$ for $r = 0, 1$ are obtained by minimizing the object function

$$\underset{\beta_0(x),\, \beta_1(x)}{\text{minimize}} \quad \sum_{i=1}^{n} K_\lambda\ (x, x_i) \left[y_i - \beta_0(x) - \beta_1(x) x_i \right]^2.$$

In this question, we will use the Gaussian kernel $K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}}$.

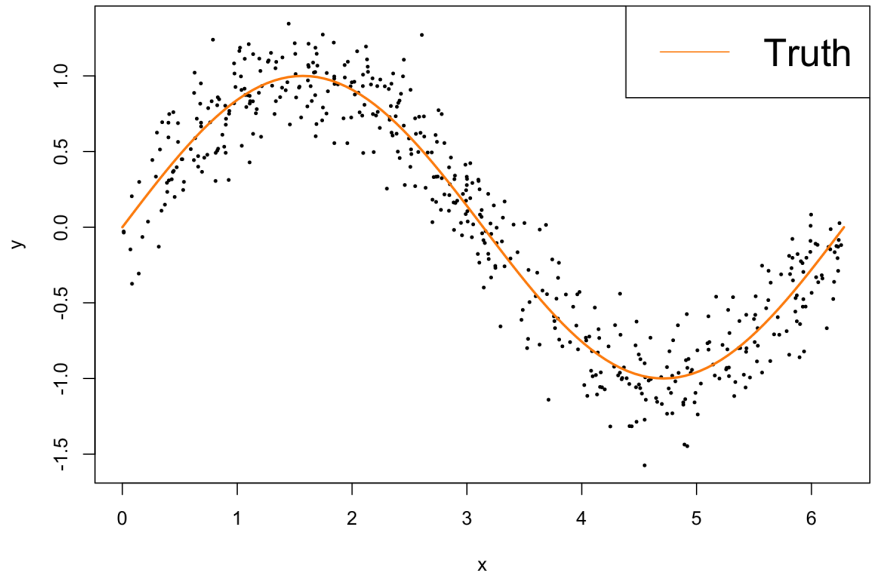
a. [20 pts] Write a function `myLocLinear(trainX, trainY, testX, lambda)`, where `lambda` is the bandwidth and `testX` is all testing samples. This function returns predictions on `testX`. The solution of $\beta_0(x)$ and $\beta_1(x)$ can be obtained by fitting a weighted linear regression. The formula is provided on Page 25 of our lecture note (https://teazrq.github.io/stat542/notes/Kernel.pdf).

```
library(matlib)

#gaussian kernel
gaus_ker<-function(x){
  c=2*3.14159265359
  (1/(c^0.5))*exp(-(x^2)/2)
}

#Local Linear function that outputs predicted Y
myLocLinear<-function(trainX, trainY, testX, lambda){

  y_pred=rep(0,length(testX))
  beta_pred=rep(0,length(testX))
  for(j in 1:length(testX)){
     w_matrix=matrix(0, length(trainX), length(trainX))
     for(i in 1:length(trainX)){
       x_dif=(abs(testX[j]-trainX[i])/lambda)
       kernel=(gaus_ker(x_dif))/lambda
       w_matrix[i,i]= kernel
     }

     beta_pred[j]=(1/(t(trainX)%*%w_matrix%*%trainX))%*%t(trainX)%*%w_matrix%*%tr
ainY
     y_pred[j]=testX[j]%*%beta_pred[j]

  }

   y_pred
}
```

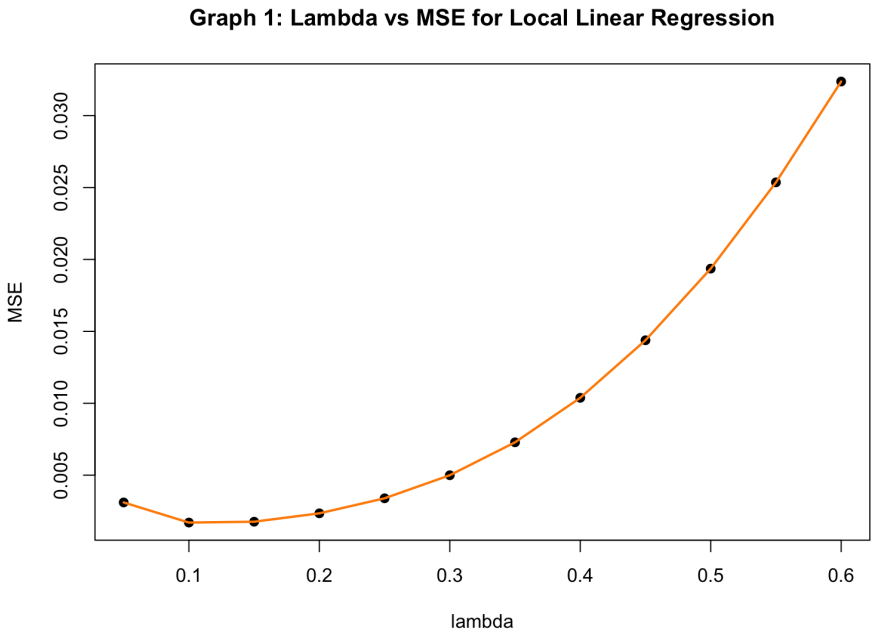
b. [15 pts] Fit a local linear regression with our given training data. The testing data are generated using the code given below. Try a set of bandwidth $\lambda = 0.05, 0.1, \ldots, 0.55, 0.6$ when calculating the kernel function.

- Provide a plot of testing MSE vs λ. Does your plot show a "U" shape?
- Report the best testing MSE with the corresponding λ.
- Plot three figures of your fitted testing data curve, with $\lambda = 0.05, 0.25$, and 0.5. Add the true function curve (see the following code for generating the truth) and the training data points onto this plot. Label each λ and your curves. Comment on the the shape of fitted curves as your λ changes.

```
train = read.csv('hw7_Q1_train.csv')
testX = 2 * pi * seq(0, 1, by = 0.01)
testY = sin(testX)
plot(train$x, train$y, pch = 19, cex = 0.3, xlab = "x", ylab = "y")
lines(testX, testY, col = "darkorange", lwd=2.0)
legend("topright", legend = c("Truth"),
       col = c("darkorange"), lty = 1, cex = 2)
```




```
library(matlib)

lambda= seq(0.05, 0.6, by = 0.05)

#calculate mse for each lambda
mse=rep(0,length(lambda))
for(i in 1:length(lambda)){
  mse[i]=mean((testY-myLocLinear(train$x, train$y, testX, lambda[i]))^2)
}

#plot lambda vs MSE graph
plot(lambda, mse, pch = 19, xlab = "lambda", ylab = "MSE")
lines(lambda, mse, col = "darkorange", lwd=2.0)
title("Graph 1: Lambda vs MSE for Local Linear Regression")
```



```
#find and report minimum MSE and corresponding lambda
mse_min=min(mse)
lambda_optim=lambda[which(mse == mse_min)]
print(paste0("Optimal lambda  = ",lambda_optim))
```

```
## [1] "Optimal lambda  = 0.1"
```

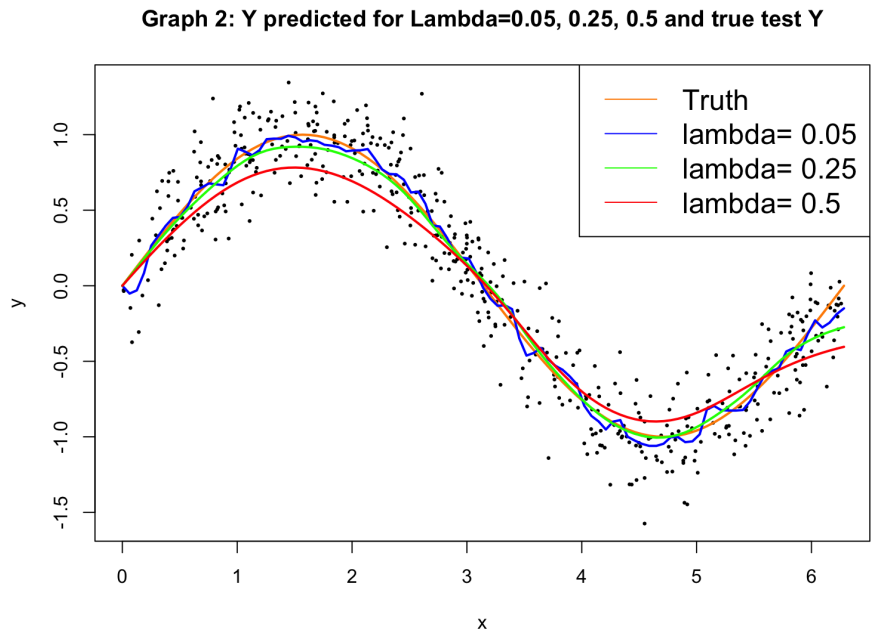
```
print(paste0("The corresponding minimum MSE at lambda optimal  = ",mse_min))
```

```
## [1] "The corresponding minimum MSE at lambda optimal  = 0.0017040990502758"
```

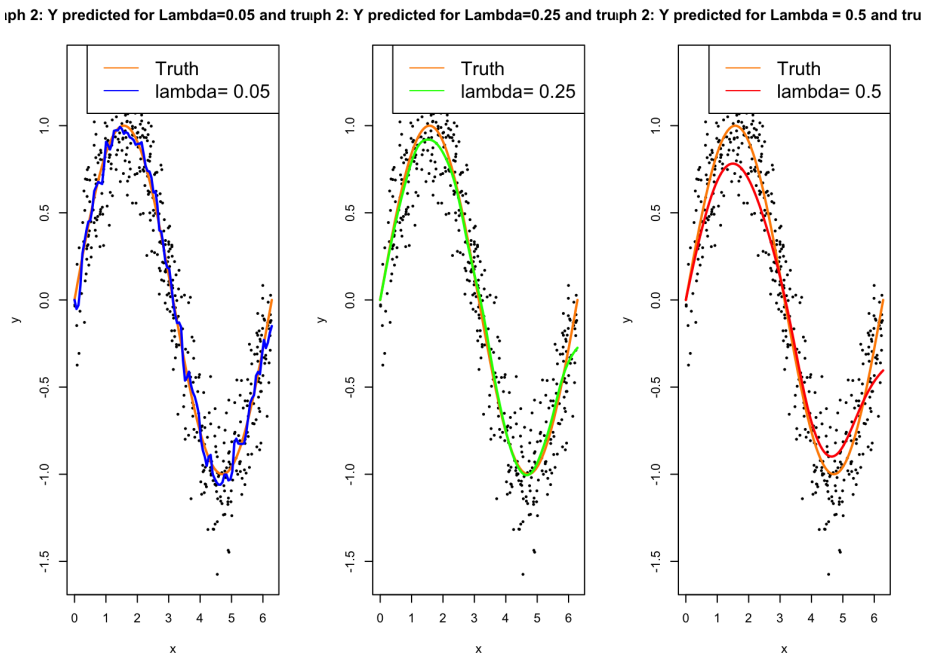
Comment

The best MSE is the minimum MSE. According to the graph 1, the best MSE is at $\lambda = 0.1$. The MSE at $\lambda = 0.1$ equals to 0.0017040990502758. Also, if we look at the graph 1: "lambda vs MSE" we can see clearly it has U-shape.

```
#plot graphs for lambda=0.05, 0.25 , and 0.5 on the same graph
  plot(train$x, train$y, pch = 19, cex = 0.3, xlab = "x", ylab = "y")
  lines(testX, testY, col = "darkorange", lwd=2.0)
  lines(testX, myLocLinear(train$x, train$y, testX, 0.05), col = "blue", lwd=2.0)
  lines(testX, myLocLinear(train$x, train$y, testX, 0.25),  col = "green", lwd=2.
0)
  lines(testX, myLocLinear(train$x, train$y, testX, 0.5),  col = "red", lwd=2.0)
  title("Graph 2: Y predicted for Lambda=0.05, 0.25, 0.5 and true test Y")
  legend("topright", legend = c("Truth", "lambda= 0.05", "lambda= 0.25", "lambda=
0.5"),
         col = c("darkorange","blue","green","red"), lty = 1, cex = 1.5)
```



```
#plot 3 seperate graphs for lambda=0.05, 0.25 , and 0.5
par(mfrow=c(1,3))

plot(train$x, train$y, pch = 19, cex = 0.3, xlab = "x", ylab = "y")
  lines(testX, testY, col = "darkorange", lwd=2.0)
  lines(testX, myLocLinear(train$x, train$y, testX, 0.05), col = "blue", lwd=2.0)
  title("Graph 2: Y predicted for Lambda=0.05 and true test Y")
  legend("topright", legend = c("Truth", "lambda= 0.05"),
         col = c("darkorange","blue"), lty = 1, cex = 1.5)

  plot(train$x, train$y, pch = 19, cex = 0.3, xlab = "x", ylab = "y")
  lines(testX, testY, col = "darkorange", lwd=2.0)
  lines(testX, myLocLinear(train$x, train$y, testX, 0.25),  col = "green", lwd=2.
0)
  title("Graph 2: Y predicted for Lambda=0.25 and true test Y")
  legend("topright", legend = c("Truth", "lambda= 0.25"),
         col = c("darkorange","green"), lty = 1, cex = 1.5)

  plot(train$x, train$y, pch = 19, cex = 0.3, xlab = "x", ylab = "y")
  lines(testX, testY, col = "darkorange", lwd=2.0)
  lines(testX, myLocLinear(train$x, train$y, testX, 0.5),  col = "red", lwd=2.0)
  title("Graph 2: Y predicted for Lambda = 0.5 and true test Y")
  legend("topright", legend = c("Truth", "lambda= 0.5"),
         col = c("darkorange", "red"), lty = 1, cex = 1.5)
```



Comment

According to graph 2 we can see that when lambda=0.05 there is overfitting, as the fluctuations are present. lambda=0.25 produces proper fitting curve. When lambda=0.5 there is underfitting.

Question 2 [35 Points] Linear Discriminant Analysis

For both question 2 and 3, you need to write your own code. We will use the handwritten digit recognition data from the `ElemStatLearn` package. We only consider the train-test split, with the pre-defined `zip.train` and `zip.test`. Simply use `zip.train` as the training data, and `zip.test` as the testing data for all evaluations and tuning. No cross-validation is needed in the training process.

- The data consists of 10 classes: digits 0 to 9 and 256 features (16×16 grayscale image).
- More information can be attained by code `help(zip.train)`.

```
library(ElemStatLearn)

# load training and testing data
dim(zip.train)
```

```
## [1] 7291  257
```

```
dim(zip.test)
```

```
## [1] 2007  257
```

```
# number of each digit
table(zip.train[, 1])
```

```
##
##    0    1    2    3    4    5    6    7    8    9
## 1194 1005  731  658  652  556  664  645  542  644
```

a. [10 pts] Estimate the mean, covariance matrix of each class and pooled covariance matrix. Basic built-in R functions such as `cov` are allowed. Do NOT print your results.

```
#mean
library(tidyverse)
train=as.data.frame(zip.train)

#create dataset of each class

group_0 <- train %>% filter(V1==0)
group_0 <- group_0 %>% select(- c(V1 ))

group_1<-train %>% filter(V1==1)
group_1 <- group_1 %>% select(- c(V1 ))

group_2<-train %>% filter(V1==2)
group_2 <- group_2 %>% select(- c(V1 ))

group_3<-train %>% filter(V1==3)
group_3 <- group_3 %>% select(- c(V1 ))

group_4<-train %>% filter(V1==4)
group_4 <- group_4 %>% select(- c(V1 ))

group_5<-train %>% filter(V1==5)
group_5 <- group_5 %>% select(- c(V1 ))

group_6<-train %>% filter(V1==6)
group_6 <- group_6 %>% select(- c(V1 ))

group_7<-train %>% filter(V1==7)
group_7 <- group_7 %>% select(- c(V1 ))

group_8<-train %>% filter(V1==8)
group_8<- group_8 %>% select(- c(V1 ))

group_9<-train %>% filter(V1==9)
group_9 <- group_9 %>% select(- c(V1 ))

class_list<-list(group_0, group_1, group_2, group_3, group_4, group_5, group_6, g
roup_7, group_8, group_9)

n_k_list<-sapply(class_list, nrow)

#create list of means of each class
mean_list<-list(colMeans(group_0[sapply(group_0, is.numeric)]) , colMeans(group_1
[sapply(group_1, is.numeric)]) , colMeans(group_2[sapply(group_2, is.numeric)]) ,
colMeans(group_3[sapply(group_3, is.numeric)]) , colMeans(group_4[sapply(group_4,
is.numeric)]) ,colMeans(group_5[sapply(group_5, is.numeric)]) , colMeans(group_6
[sapply(group_6, is.numeric)]) , colMeans(group_7[sapply(group_7, is.numeric)]) ,
colMeans(group_8[sapply(group_8, is.numeric)]) , colMeans(group_9[sapply(group_9,
is.numeric)]) )
```

```
#create list of covariance of each class
cov_list<-list(as.matrix(cov(group_0)),as.matrix(cov(group_1)), as.matrix(cov(gro
up_2)), as.matrix(cov(group_3)), as.matrix(cov(group_4)), as.matrix(cov(group_
5)), as.matrix(cov(group_6)), as.matrix(cov(group_7)), as.matrix(cov(group_8)), a
s.matrix(cov(group_9)))


#calculate pooled covariance matrix
pooled_covariance=matrix(0,256,256)
str(cov_list[1])
```

```
## List of 1
##  $ : num [1:256, 1:256] 0.00225 0.00234 0.00232 0.00226 0.00179 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:256] "V2" "V3" "V4" "V5" ...
##   .. ..$ : chr [1:256] "V2" "V3" "V4" "V5" ...
```

```
for(i in 1:length(cov_list)){
  pooled_covariance=pooled_covariance+as.numeric(unlist(lapply(cov_list[i], funct
ion(M) {M * ((n_k_list[i])-1)})))
}

 pooled_covariance= pooled_covariance/(nrow(train)-length(cov_list))
```

b. [15 pts] Write your own linear discriminate analysis (LDA) code following our lecture note. To perform this, you should calculate μ_k, π_k, and Σ from the data. You may consider saving μ_k's and π_k's as a list (with 10 elements in each list).

You are not required to write a single function to perform LDA, but you could consider defining a function as `myLDA(testX, mu_list, sigma_pool)`, where `mu_list` is the estimated mean vector for each class, and `sigma_pool` is the pooled variance estimation. This function should return the predicted class based on comparing **discriminant functions** $\delta_k(x) = w_k^T x + b_k$ given on page 32 of the lecture note (https://teazrq.github.io/stat542/notes/Class.pdf).

```
#calculate pi_k for training dataset
pi_k=list(n_k_list[1]/nrow(train))

for(i in 2:length(n_k_list)){
  pi_k <- append(pi_k, n_k_list[i]/nrow(train))
}


#implement LDA method which outputs vector of predicted Y
myLDA<-function(testX, mu_list, sigma_pool){

disc_funct=rep(0, nrow(testX))
b=rep(0, length(mu_list))
w=matrix(0, ncol(testX), length(mu_list))
Y_pred=rep(0, nrow(testX))

  for(i in 1: nrow(testX)){
    for(j in 1:length(mu_list)){
    w[,j]=solve(sigma_pool)%*%as.numeric(unlist(mu_list[j]))
    b[j]=-0.5%*%t(as.numeric(unlist(mu_list[j])))%*%solve(sigma_pool)%*%as.numeri
c(unlist(mu_list[j]))+log(as.numeric(pi_k[j]))
    disc_funct[j]=testX[i,]%*%w[,j]+b[j]
    }
    max_funct= max(disc_funct)
    index_max= which(disc_funct ==max_funct)
    Y_pred[i]= index_max-1
    }

  Y_pred

}

#calculate predicted Y using LDA method
test=as.matrix(as.data.frame(zip.test))
testX_2=test[,-1]
Y_pred_LDA<-myLDA(testX_2, mean_list, pooled_covariance)
```

c. [10 pts] Fit LDA model on the training data and predict with the testing data.

- Report the first 5 entries of the w coefficient vector and b for digit `0`.
- Report a 10×10 confusion matrix, where each **column** is true digit and each **row** is your predicted digit. You can use the `table()` function in R.
- Report a table of misclassification rate of each (true) digit. Hence, this is the $1-$ sensitivity of each digit in a multi-class problem. Only keep the first three digits after the decimal point for the rate. Also report the overall mis-classification rate.

```
#calculate b and w vectors for class 0
b=rep(0, length(mean_list))
w=vector(mode="list", length=length(mean_list))
w_0=solve(pooled_covariance)%*%as.numeric(unlist(mean_list[1]))
b_0=-0.5%*%t(as.numeric(unlist(mean_list[1])))%*%solve(pooled_covariance)%*%as.nu
meric(unlist(mean_list[1]))+log(as.numeric(pi_k[1]))

#report irst 5 entries of w and b for class 0
print("the first 5 entries for w_0: ")
```

```
## [1] "the first 5 entries for w_0: "
```

```
print(w_0[1:5])
```

```
## [1] -549.553021   68.575047  -39.112632   -3.095659   -9.656750
```

```
print(paste0(" b_0   = ",b_0))
```

```
## [1] " b_0   = -1156.4428919059"
```

```
print("Confusion matrix: ")
```

```
## [1] "Confusion matrix: "
```

```
#calculate Confusion matrix
Y_test<-test[,1]
conf_table<-table(Y_test, Y_pred_LDA)
conf_table
```

```
##       Y_pred_LDA
## Y_test   0   1   2   3   4   5   6   7   8   9
##      0 342   0   0   4   3   1   5   0   3   1
##      1   0 251   0   2   5   0   3   0   1   2
##      2   7   2 157   4  12   2   1   1  12   0
##      3   3   0   3 142   3   9   0   1   4   1
##      4   1   4   6   0 174   0   2   2   1  10
##      5   6   0   0  16   3 125   0   0   5   5
##      6   1   0   3   0   3   3 157   0   3   0
##      7   0   1   0   2   7   0   0 129   1   7
##      8   5   0   2  11   7   4   0   0 135   2
##      9   0   0   0   0   4   0   0   5   3 165
```

```
#calculate misclassification rate for each class [0:9]
misclass_rate=rep(0,10)
misclass=rep(0,10)
for(i in 1:10){
  for(j in 1:10){
    if(i!=j){
     misclass[i]=misclass[i]+conf_table[j,i]
    }

  }
  misclass_rate[i]=misclass[i]/sum(conf_table[,i])
}

# create table of misclassification rate
class<-c(0:9)
class.df<- data.frame(class, misclass_rate)
colnames(class.df)=c("class", "misclassification rate")
class.df
```

class <int>	misclassification rate <dbl>
0	0.06301370
1	0.02713178
2	0.08187135
3	0.21546961
4	0.21266968
5	0.13194444
6	0.06547619
7	0.06521739
8	0.19642857
9	0.14507772

1-10 of 10 rows

```
#overall misclassification rate
Y_test<-test[,1]
t_f<-table(Y_test==Y_pred_LDA)
overall_misclass_rate=t_f[1]/(t_f[1]+t_f[2])
print(paste0("Overall misclassification rate  = ",overall_misclass_rate))
```

```
## [1] "Overall misclassification rate  = 0.114598903836572"
```

Question 3 [30 points] Regularized quadratic discriminate analysis

QDA uses a quadratic discriminant function. However, QDA does not work directly in this example because we do not have enough samples to provide an invertible sample covariance matrix for each digit. An alternative idea to fix this issue is to consider a regularized QDA method, which uses

$$\widehat{\Sigma}_k(\alpha) = \alpha\widehat{\Sigma}_k + (1-\alpha)\widehat{\Sigma}$$

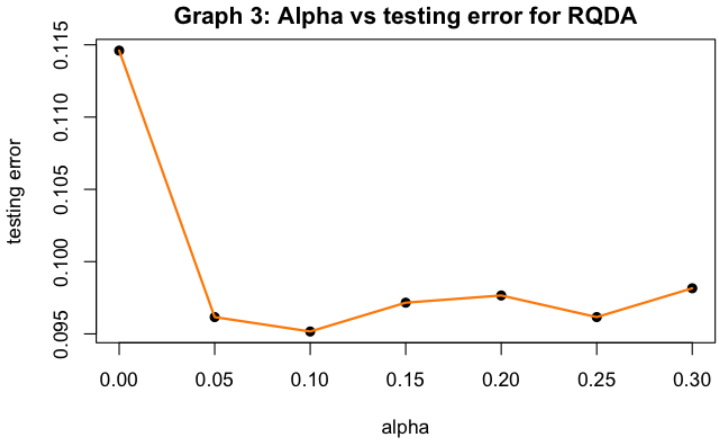
instead of Σ_k. Then, they are used in the decision rules given in page 36 of lecture notes. Complete the following questions.

a. [20 pts] Write your own function `myRQDA(testX, mu_list, sigma_list, sigma_pool, alpha)`, where `allpha` is a scaler `alpha` and `testX` is your testing covariate matrix. And you may need a new `sigma_list` for all the Σ_k. This function should return a vector of predicted digits.

```
#function to calculate regularized list of covariance matrices
sigma_alpha<-function(alpha, pooled_covariance, cov_list){
  sigma_list=vector(mode="list", length=length(cov_list))
  for(i in 1:length(cov_list)){
    sigma_list[[i]]=as.vector(alpha)*as.numeric(unlist(cov_list[i]))+as.vector((1
-alpha))*pooled_covariance
  }

  sigma_list
}

#implement RQDA method which outputs the vector of predicted Y
myRQDA<-function(testX, mu_list, sigma_list, sigma_pool, alpha){
  disc_funct=rep(0, length(mu_list))
  Y_pred=rep(0, nrow(testX))
  sigma_list2<-sigma_alpha(alpha,sigma_pool, sigma_list)

  for(i in 1: nrow(testX)){
    for(j in 1:length(mu_list)){

  disc_funct[j]=-0.5%*%log(abs(as.numeric(determinant((sigma_list2[[j]]), logarit
hm = TRUE)$modulus)))-0.5%*%t((testX[i,]-mean_list[[j]]))%*%solve(sigma_list2
[[j]])%*%(testX[i,]-mean_list[[j]])+log(pi_k[[j]])
    }
    max_funct= max(disc_funct)
    index_max= which(disc_funct ==max_funct)
    Y_pred[i]=index_max-1
  }

  Y_pred
}
```

b. [10 pts] Perform regularized QDA with the following sequence of α values. Plot the testing error (misclassification rate) against alpha. Report the minimum testing error and the corresponding α.

```
alpha_all = seq(0, 0.3, by = 0.05)

test=as.matrix(as.data.frame(zip.test))
testX=test[,-1]

Y_test<-test[,1]

Y_pred_matrix=matrix(0, nrow(testX),length(alpha_all))

#function to calculate misclassification rate
MSE<-function(Y_pred, Y_true){
  count=0
  for( i in 1:length(Y_true)){
    if(Y_true[i]!=Y_pred[i]){
      count=count+1
    }
  }
  count/length(Y_true)
}

#calculate testing error/misclassification rate for alpha_all
test_error=rep(0, length(alpha_all))
for(i in 1:length(alpha_all)){
  Y_predict=myRQDA(testX, mean_list, cov_list, pooled_covariance, alpha_all[i])
  Y_pred_matrix[,i]=Y_predict
  test_error[i]=MSE(Y_predict, Y_test)

}

#Find minimum testing error and corresponding alpha
test_error_min=min(test_error)
alpha_optim=alpha_all[which(test_error == test_error_min)]

#plot graph "Alpha vs testing error for RQDA"
plot(alpha_all, test_error, pch = 19, xlab = "alpha", ylab = "testing error")
lines(alpha_all, test_error, col = "darkorange", lwd=2.0)
title("Graph 3: Alpha vs testing error for RQDA")
```



Comment

The minimum testing error = 0.0951669157947185 and it corresponds to $\lambda = 0.1$