HW - 4 574

Shrey

2022-10-26

1.

Paper talks about using nearest shrunken centroids tenchnique for classifying cancer types using microarrays. They compute T statistics for each class and use a regularization terms S0

$$t_{kj} = \frac{\hat{\mu_{kj}} - \mu_j}{mk(sj + s0)}$$

The tkj is reduced to |tkj|-delta, with a threholding parameter delta and if the difference is less than zero then the class centroids are shrunked (soft thresholding) or tkj becomes tkjxI(|tkj|>delta) (hard thresholding). K discriminanat functions are find the distance of the test point from the K shrunken centroids and priors, which are used to classify the test points. The features are selected based on cross-validation log likelyhoods. A controling parameter θ can be used to regularize the errors.

2.

Best Subset selection of size M

We want to select the subset of size M that will give the most significant coefficients.

Since the Xj's are othogonal then the coefficient β_j for Xj will be independent of other X's and will be the same as obtained in univariate regression of Y on Xj.

Therefore we will select the feaures that are giving the largest M Betas. We will rank the coefficients and select the one's that are greater than the M'th coefficient

Therefore we can write

$\beta_j^{BestSubset} = \beta_j^{ols} I(|\beta_j^{ols}| > |\beta_M|)$

Ridge

We know that

$\beta^{OLS} = (X^T X)^{-1} X^T Y$

When the Xj's are scaled and they are orthonormal then,

$(X^T X) = I$,

therefore $\beta^{OLS} = X^T Y$

and we know in ridge regression $\beta^{ridge} = (X^T X + \lambda)^{-1} X^T Y$

In orthonormal conditions the above equation can be written as-

$\beta^{ridge} = (I + \lambda)^{-1} \beta^{OLS}$

Which can be written as $\beta_j^{ridge} = \frac{\beta_j^{OLS}}{1+\lambda}$

3.

(a)

$ RSS = (Y-X)^T(Y-X)$

To minimize the RSS w.r.t β

$\nabla_\beta RSS = 0$ $\{\}(Y-X)^T(Y-X) = 0$ $ $_{}(Y^TY-Y^TX-^TXTY+^TXTX) = 0 $

$0 - X^TY - X^TY + X^TX\beta + (\beta^TX^TX)^T = 0\ X^TX\beta = X^TY\ \hat{\beta} = (X^TX)^{-1}X^TY$

(b)

In OLS regression $\hat{\beta} = (X^TX)^{-1}X^TY$

Taking expectatio operator both sides

$E[\hat{\beta}] = E[(X^TX)^{-1}X^TY]\ E[\hat{\beta}] = (X^TX)^{-1}X^TE[Y]$

We know $Y = \beta X + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$

$E[\hat{\beta}] = (X^TX)^{-1}X^TE[\beta X + \epsilon]$

$E[\hat{\beta}] = (X^TX)^{-1}X^TXE[\beta] + (X^TX)^{-1}X^TE[\epsilon]$

$E[\hat{\beta}] = \beta$

Therefore we can say its an unbiased estimator of β

$$Cov(\hat{\beta}) = \begin{bmatrix} \sigma^2(\beta_0) & \sigma(\beta_0,\beta_1) & - & - & - & \sigma(\beta_0,\beta_p)) \\ \sigma(\beta_1,\beta_0) & \sigma^2(\beta_1) & - & - & - & \sigma(\beta_1,\beta_p) \\ | & | & | & | & | & | \\ | & | & | & | & | & | \\ \sigma(\beta_p,\beta_0) & \sigma(\beta_p,\beta_1) & - & - & - & \sigma^2(\beta_p) \end{bmatrix}$$

variance-covariance matrix of beta (p+1)x(p+1) matrix with diagonal elements being variance of $\beta_{j,j}$ and non diagonal elemenst being covariance between β_i and β_j

(c)

$\hat{\beta} = argminRSS + \lambda\beta^T\beta$ $ RSS = (Y-X)^T(Y-X)$ $\nabla_\beta[(Y - X\hat{\beta})^T(Y - X\hat{\beta}) + \lambda\beta^T\beta] = 0$ $ $_{}(Y^TY-Y^TX-^TXTY+^TXTX+^T) = 0$ $ $0 - X^TY - X^TY + X^TX\beta + (\beta^TX^TX)^T + 2\lambda\beta = 0\ (X^TX + \lambda I)\beta = X^TY$

$\hat{\beta_{ridge}} = (X^TX + \lambda I)^{-1}X^TY$

(d)

$\hat{\beta_{ridge}} = (X^TX + \lambda I)^{-1}X^TY$

$\hat{\beta_{OLS}} = (X^TX)^{-1}X^TY$

from above two equations we can write

$\hat{\beta_{ridge}} = (X^TX + \lambda I)(X^TX)\hat{\beta_{OLS}}$

Therefore $\hat{\beta_{ridge}} = K_\lambda\hat{\beta_{OLS}}$ where $K_\lambda = (X^TX + \lambda I)(X^TX)$

(e)

We know $E[\hat{\beta_{OLS}}] = \beta$

and $\hat{\beta_{ridge}} = \hat{\beta_{OLS}}$

Thus we can say $E[\hat{\beta_{ridge}}] = K_\lambda \beta$ Therfore $bias = \beta(1 - k_\lambda)$

When lambda approaches to zero K becomes becomes 1 and bias becomes zero

(f)

$\hat{\beta_{ridge}} = (X^T X + \lambda I)^{-1} X^T Y$

When lambda approaches to zero K becomes 1 and β_{ridge} tends towards β_{OLS}

When lambda approaches to infinity $(X^T X + \lambda I)^{-1}$ approaches to zero therefore β_{ridge} tends towards zero.

(g)

When p >> n X can have maximum rank of n and X transpose can also have maximum rank of n. Therefore max rank of $X^T X$ is n hence it is not full rank and it will be ininvertible.

In other words in reduced form one or more columns of $X^T X$ will have no pivots.

When it becomes $(X^T X + \lambda I)$ a non zero term is added to the diagonal elements therfore making making all the columns have pivot element and making the matrix full rank and invertible.

When ridge estimator is used we guaratntee that there will be one unique correct solution. If we use OLS and the matrix becomes non full rank then we may have infinite solutions, Therfore to avoid this it's better to use ridge instead of OLS

4.

(a)

```
library(MASS)

set.seed(2000)
green<-mvrnorm(100,c(2,1),diag(2))
red<-mvrnorm(100,c(1,2),diag(2))


set.seed(2014)
green_test<-mvrnorm(500,c(2,1),diag(2))
red_test<- mvrnorm(500,c(1,2),diag(2))

X_train = rbind(green,red)
y_train = factor(c(rep(1,100),rep(0,100)))


X_test = rbind(green_test,red_test)
y_test = factor(c(rep(1,500),rep(0,500)))

library(class)
num_neighbors = c(1, 4, 7, 10, 13, 16, 30, 45, 60, 80, 100, 150, 200)
train_errors = c()
test_errors = c()
for (x in num_neighbors)
{
 m1 = knn(X_train,X_train,y_train,k=x)
 m2 = knn(X_train,X_test,y_train,k=x)
 train_errors = append(train_errors, mean(m1!=y_train))
 test_errors = append(test_errors,mean(m2!=y_test))
}

print(train_errors)
```

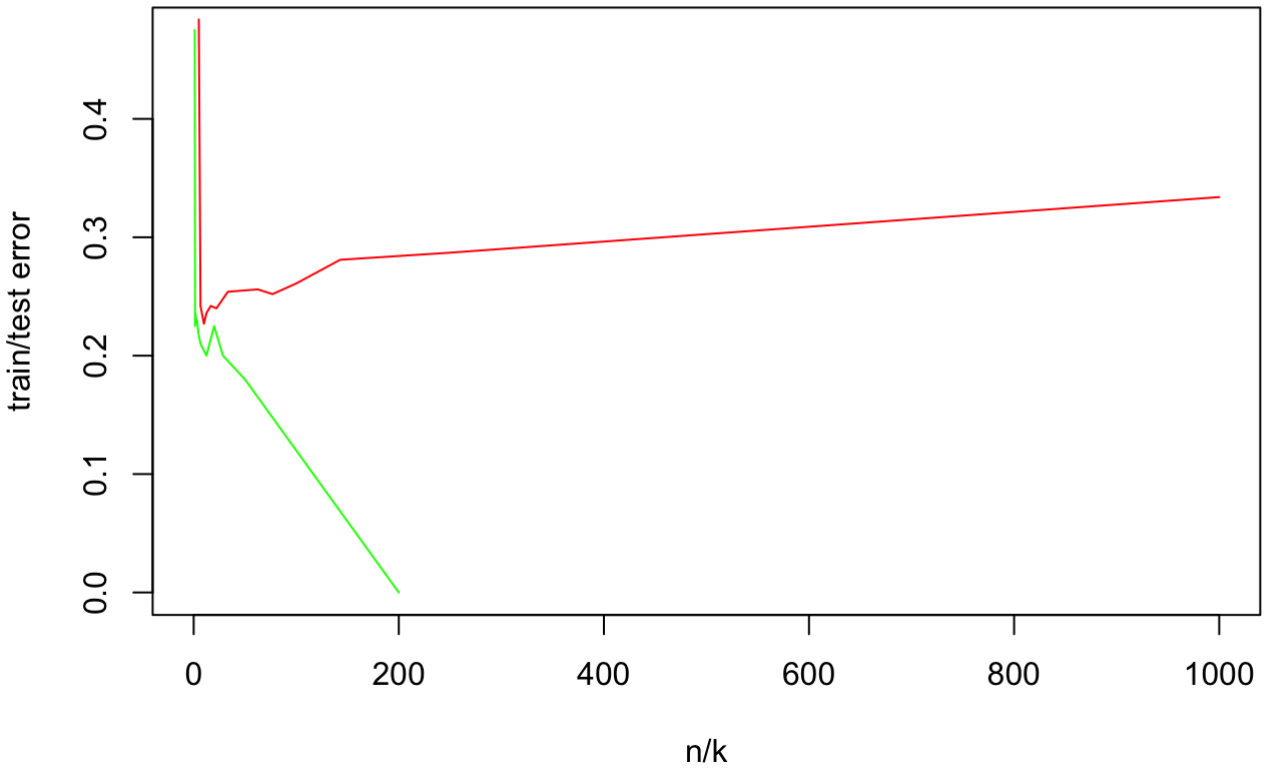
```
##  [1] 0.000 0.180 0.200 0.225 0.210 0.200 0.210 0.220 0.230 0.230 0.235 0.225
## [13] 0.475
```

```
print(test_errors)
```

```
##  [1] 0.334 0.287 0.281 0.261 0.252 0.256 0.254 0.240 0.242 0.236 0.227 0.242
## [13] 0.484
```

```
dof_train =  200/num_neighbors
dof_test = 1000/num_neighbors

plot(dof_train,train_errors,xlab = "n/k",ylab = "train/test error",type="l",col = "g
reen",xlim= c(0,1000))
lines(dof_test,test_errors,xlab = "n/k",ylab = "test error",type="l",col="red")
```



(b)

```
library(MASS)
#generate ten centers, which are treated as fixed parameters
Sig <- matrix(c(1,0,0,1),nrow=2)
seed_center <- 16
set.seed(seed_center)
center_green <- mvrnorm(n=10,c(1,0),Sig)
center_red <- mvrnorm(n=10,c(0,1),Sig)
##define a function "gendata2" first
gendata2 <-function(n,mu1,mu2,Sig1,Sig2,myseed)
{
set.seed(myseed)
mean1 <- mu1[sample(1:10,n,replace=T),]
mean2 <- mu2[sample(1:10,n,replace=T),]
green <- matrix(0,ncol=2,nrow=n)
red <- matrix(0,ncol=2,nrow=n)
for(i in 1:n){
green[i,] <- mvrnorm(1,mean1[i,],Sig1)
red[i,] <- mvrnorm(1,mean2[i,],Sig2)
}
x <- rbind(green,red)
return(x)
}
#generate the training set
seed_train <- 2000
ntrain <- 100
train2 <- gendata2(ntrain,center_green,center_red,Sig/5,Sig/5,seed_train)
ytrain <- c(rep(1,ntrain),rep(0,ntrain))

seed_test <- 2014
ntest <- 500
test2 <- gendata2(ntest,center_green,center_red,Sig/5,Sig/5,seed_test)
ytest <- c(rep(1,ntest),rep(0,ntest))

 library(class)
 num_neighbors = c(1, 4, 7, 10, 13, 16, 30, 45, 60, 80, 100, 150, 200)
 train_errors2 = c()
 test_errors2 = c()

 for (x in num_neighbors)
 {
  m1 = knn(train2,train2,ytrain,k=x)
  m2 = knn(train2,test2,ytrain,k=x)
  train_errors2 = append(train_errors2, mean(m1!=ytrain))
  test_errors2 = append(test_errors2,mean(m2!=ytest))
 }

 print(train_errors2)
```

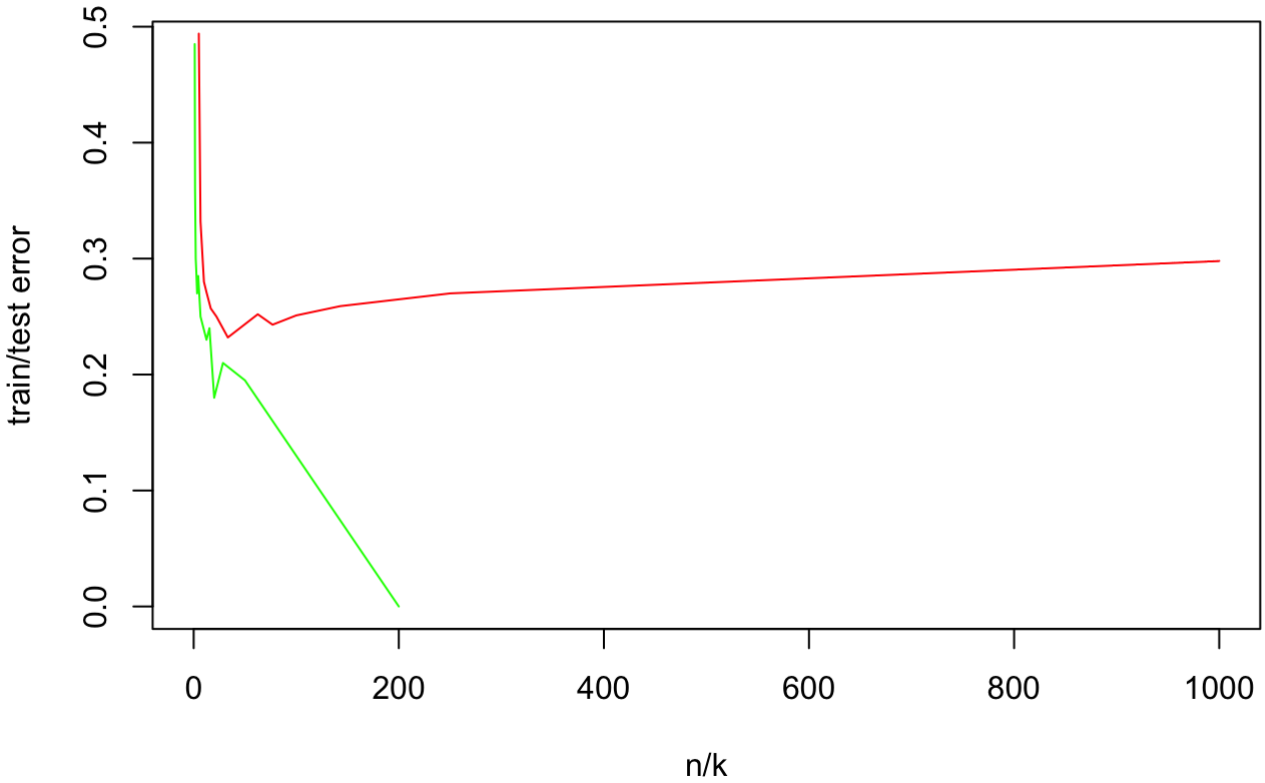
```
##  [1] 0.000 0.195 0.210 0.180 0.240 0.230 0.250 0.285 0.270 0.290 0.300 0.360
## [13] 0.485
```

```
 print(test_errors2)
```

```
##  [1] 0.298 0.270 0.259 0.251 0.243 0.252 0.232 0.250 0.257 0.271 0.280 0.332
## [13] 0.494
```

```
 dof_train =  200/num_neighbors
 dof_test = 1000/num_neighbors

 plot(dof_train,train_errors2,xlab = "n/k",ylab = "train/test error",type="l",col =
"green",xlim= c(0,1000))
 lines(dof_test,test_errors2,xlab = "n/k",ylab = "test error",type="l",col="red")
```



(c)

For each of the two plots we observe following things:- (things:-) 1. Train and test errors are maximum when n/k is very small i.e K is very large 2. The test errors decrease when n/k increases and reach a minimum and starts to increase again 3. The train error keeps decreasing with increasing degree of freedom

We will select the value of k which is giving the minimum test errror

```
k_scenario_1 = num_neighbors[which.min(test_errors)]
k_scenario_2 = num_neighbors[which.min(test_errors2)]

print(k_scenario_1)
```

```
## [1] 100
```

```
print(k_scenario_2)
```

```
## [1] 30
```

5.

(a)

```
train_data = read.table(gzfile("/Users/shreyarora/Downloads/zip.train"))
test_data = read.table(gzfile("/Users/shreyarora/Downloads/zip.test"))

train_data = train_data[(train_data[,1] == 1 | train_data[,1] == 2 | train_data[,1] =
= 3),]
test_data = test_data[(test_data[,1] == 1 | test_data[,1] == 2 | test_data[,1] == 3
),]

x_train = train_data[,2:257]
y_train =factor(train_data[,1])

x_test = test_data[,2:257]
y_test = factor(test_data[,1])

neighbors = c(1, 3, 5, 7, 15)
library(class)
train_errors = c()
test_errors = c()

for (x in neighbors)
{
 m1 = knn(x_train,x_train,y_train,k=x)
 m2 = knn(x_train,x_test,y_train,k=x)
 train_errors = append(train_errors, mean(m1!=y_train))
 test_errors = append(test_errors,mean(m2!=y_test))
}

print(train_errors)
```

```
## [1] 0.000000000 0.004177109 0.004594820 0.005847953 0.010860485
```

```
print(test_errors)
```

```
## [1] 0.02229299 0.02388535 0.02388535 0.02707006 0.03184713
```

(b)

```
xtrain2 = x_train[-16]
xtest2 = x_test[-16]

model = lda(xtrain2,y_train)
train_errors_lda = mean(predict(model,xtrain2)$class != y_train)
test_errors_lda = mean(predict(model,newdata = xtest2)$class != y_test)

print(train_errors_lda)
```

```
## [1] 0.004177109
```

```
print(test_errors_lda)
```

```
## [1] 0.03343949
```

6

(a)

```
library(MASS)

 set.seed(2000)
 green<-mvrnorm(100,c(2,1),diag(2))
 red<-mvrnorm(100,c(1,2),diag(2))

 set.seed(2014)
 green_test<-mvrnorm(500,c(2,1),diag(2))
 red_test<- mvrnorm(500,c(1,2),diag(2))

 X_train = rbind(green,red)
 y_train = factor(c(rep(1,100),rep(0,100)))
 train_data_cv = data.frame(X_train,y_train)

makefolds<- function(df,s)
  {
  n = length(df)
  class1_index = which(df['y_train']==1)
  class0_index = which(df['y_train']==0)
  set.seed(s)
  class1_index_random = sample(class1_index)
  class0_index_random = sample(class0_index)
  return(cbind(class1_index_random,class0_index_random))

  }

  shuffled_indexes = makefolds(train_data_cv,10)

  ##cross validation
  errors_lda = c()
  for (k in seq(1,5))
  {

    test_indexes = shuffled_indexes[(20*k-19):(20*k),]
    test_indexes = append(test_indexes[,1],test_indexes[,2])
    test_data = X_train[test_indexes,]
    train_data = X_train[-test_indexes,]
    ##lda
    model1 = lda(train_data, y_train[-test_indexes])
    error1 = mean(predict(model1,newdata = test_data)$class != y_train[test_indexes])
    errors_lda = append(errors_lda,error1)

  }

  print(mean(errors_lda))
```

```
## [1] 0.225
```

```
  print(errors_lda)
```

```
## [1] 0.225 0.175 0.225 0.300 0.200
```

```
shuffled_indexes = makefolds(train_data_cv,19)

errors_logistic = c()
for (k in seq(1,5))
  {

    test_indexes = shuffled_indexes[(20*k-19):(20*k),]
    test_indexes = append(test_indexes[,1],test_indexes[,2])
    test_data = X_train[test_indexes,]
    train_data = X_train[-test_indexes,]

    ##logitic regrression
    train_logistic = data.frame(train_data,y_train[-test_indexes])
    colnames(train_logistic) = c("X1","X2","Y")

    test_logistic = data.frame(test_data,y_train[test_indexes])
    colnames(test_logistic) = c("X1","X2","Y")

    model2 = glm(Y~.,family = binomial(link="logit"), data = train_logistic)
    error2 = mean(as.numeric(predict(model2, newdata = test_logistic[,1:2])>0)!= y_tr
ain[test_indexes])
    errors_logistic = append(errors_logistic,error2)

  }

print(mean(errors_logistic))
```

```
## [1] 0.215
```

```
print(errors_logistic)
```

```
## [1] 0.200 0.225 0.125 0.200 0.325
```