

(P1)

Stat 154
HW3

3.12

X is centered matrix $\cancel{p \times p}$
 Y is the output vector

$$\hat{X} = \begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix} \quad \lambda I \text{ is of dimension } p \times p.$$

$$\hat{Y} = \begin{bmatrix} Y \\ 0 \end{bmatrix} \quad 0 \text{ is the zeroes matrix of size } p \times 1$$

The original problem has the least squares solution

$$\beta_{LS} = (X^T X)^{-1} X^T Y$$

$$\therefore \hat{\beta}_{LS} = (\hat{X}^T \hat{X})^{-1} \hat{X}^T \hat{Y} \quad (\text{solution to the new eqn}) \quad \text{--- ①}$$

If we perform expand the equations

$$\hat{X} = \begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix} \quad \hat{X}^T = \begin{bmatrix} X^T & \lambda I \end{bmatrix}$$

$$\therefore \hat{X}^T \hat{X} = \begin{bmatrix} X^T & \sqrt{\lambda} I \end{bmatrix} \begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix} = X^T X + \lambda I_{p \times p}$$

$$\hat{X}^T \hat{Y} = \begin{bmatrix} X^T & \sqrt{\lambda} I \end{bmatrix} \begin{bmatrix} Y \\ 0_{p \times 1} \end{bmatrix} = X^T Y + 0$$

Resubstituting in ①

$$\hat{\beta}_{LS} = (X^T X + \lambda I)^{-1} X^T Y \quad \text{--- ②}$$

If we go back to ridge regression solution

$$\hat{\beta}^{\text{ridge}} = \underset{\beta}{\operatorname{argmin}} \left(\sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j)^2 + \lambda \sum_{j=1}^p \beta_j^2 \right)$$

where $\lambda \geq 0$ is the complexity parameter.

by poso

replacing x_{ij} with \bar{x}_j (recentering the data points) and estimating β_0 using $\bar{y} = \frac{1}{N} \sum_{i=1}^n y_i$

Since X has now been recentered we know X has p columns now.

$\text{RSS}(\lambda) = (y - X\beta)^T (y - X\beta) + \lambda \beta^T \beta$
from which we get ridge regression solution:

$$\hat{\beta}_{\text{ridge}} = (X^T X + \lambda I)^{-1} X^T y \quad \text{--- (3)}$$

This is equivalent to eq ②

Hence proved.

example both set not working having a set of

have the classes helped?
Doesn't help how hard was it

$$T(X^T X) = X^T X = X$$

$$T(X^T X) = X^T X = X^T X$$

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

(in particular)

$$②. - Y^T X^T (T(X^T X)) = 0$$

what does this mean again in term of sub-1

$$(A + \lambda I)^{-1} (A^T A) = \lambda^{-1} I$$

so we can write $(A + \lambda I)^{-1} A^T = \lambda^{-1} I$

P2)

HTF 3.7

- * Assumption: $y_i \sim N(\beta_0 + x_i^T \beta, \sigma^2)$ $i = 1, 2, \dots, N$.
- * $\beta_j \sim N(0, \tau^2)$ $j = 1, \dots, p$ & they are independent of each other.
- * σ^2 & τ^2 are known.
- * β_0 is not governed by a prior.

We know.

~~For now:~~: The data $X = (x_1, x_2, \dots, x_p)$ is constant.

Acc to Bayes' Rule:

$$Pr(\beta|y) = \frac{Pr(y|\beta) \cdot Pr(\beta)}{Pr(y)}$$

$Pr(y)$ is a constant \because it is the marginal density

$$\therefore Pr(\beta|y) \propto Pr(y|\beta) \cdot Pr(\beta)$$

This means that the probability of β given the data (posterior distribution) is proportional to $Pr(y|\beta) \cdot Pr(\beta)$

We know that $Pr(y|\beta) \sim N(\beta_0 + X\beta, \sigma^2)$

$$Pr(\beta) \sim N(0, \tau^2)$$

If $p(\beta)$ is the pdf then

$$p(\beta) = C \exp\left(-\frac{\|\beta\|^2}{2\tau^2}\right)$$

$$p(y|\beta) = C_1 \exp\left(-\frac{\|y - X\beta\|^2}{2\sigma^2}\right)$$

$$p(\beta|y) = p((\beta) \times (y|\beta))$$

$$= C \exp\left(-\frac{\|y - X\beta\|^2}{2\sigma^2} - \frac{\|\beta\|^2}{2\tau^2}\right)$$

$$= C \exp\left(-\frac{\|y - X\beta\|^2}{2\sigma^2} + \frac{(\sigma^2/\tau^2)\|\beta\|^2}{2\sigma^2}\right)$$

This is the distribution for β given y .

It is a normal distribution but not centered at 0.

taking the negative log of RHS & LHS.

$$\begin{aligned}-\log(p(\beta|y)) &= \frac{1}{2\sigma^2} \left(\|y - X\beta\|^2 + \sigma^2/\gamma^2 \|\beta\|^2 \right) + \log(C') \\&= \frac{1}{2\sigma^2} \left(\sum_{i=1}^N (y_i - \beta_0 - \sum_{j=1}^p x_{ij}\beta_j)^2 + \sigma^2/\gamma^2 \sum_{j=1}^p (\beta_j)^2 \right) + \log(C')\end{aligned}$$

$$-\log(p(\beta|y)) = \frac{1}{2\sigma^2} \sum_{i=1}^N (y_i - \beta_0 - \sum_{j=1}^p x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^p \beta_j^2 \quad \text{where } \lambda = \sigma^2/\gamma^2$$

$\frac{C'}{2\sigma^2}$ is a constant

β_0 is not governed by a prior

$$\therefore X^\top \beta = f_0 + \left(\sum_{j=1}^p x_{ij}\beta_j \right)$$

$C'/2\sigma^2$ is a constant

$$\therefore -\log(p(\beta|y)) \propto \sum_{i=1}^N (y_i - \beta_0 - \sum_{j=1}^p (x_{ij}\beta_j))^2 + \lambda \sum_{j=1}^p \beta_j^2$$

Hence proved

Writing of real multivariate stat on one
to handle the real multivariate lesson 1 with

(Q3)

(3)

HTF 4.2

features $x \in \mathbb{R}^p$ class sizes: N_1, N_2 a) For each class ω_k the Bayes' discriminant function ($\delta_k(x)$)

$$\delta_k(x) = \log_e(p(x|\omega_k)) + \log_e(\pi_k) \quad \text{--- (1)} \quad \pi_i = \frac{N_i}{N}$$

where $p(x|\omega_k)$ is the conditional density.

The function form:

$$\begin{aligned} p(x|\omega_k) &= \mathcal{N}(x; \mu_k, \Sigma_k) \\ &= \frac{1}{\sqrt{(2\pi)^p |\Sigma_k|}} e^{-\frac{1}{2} (x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k)} \quad \text{--- (2)} \end{aligned}$$

$$\therefore \ln(p(x|\omega_k)) = -\frac{1}{2} (x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) - \left(\frac{p}{2} \ln(2\pi) + \frac{1}{2} \ln(|\Sigma_k|) \right) \quad \text{--- (3)}$$

Substituting in eqn (1)

$$\delta_k(x) = -\frac{1}{2} (x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) - \frac{p}{2} \ln(2\pi) - \frac{1}{2} \ln(|\Sigma_k|) + \ln(\pi_k)$$

* LDA corresponds to equal variance covariance matrices ($\Sigma_k = \Sigma$)* We can eliminate $-\frac{p}{2} \ln(2\pi) - \frac{1}{2} \ln(|\Sigma|)$ for decision purposes.

$$\therefore \delta_k(x) = -\frac{1}{2} (x - \mu_k)^T \Sigma^{-1} (x - \mu_k) + \ln(\pi_k)$$

$$= -\frac{1}{2} (x^T \Sigma^{-1} x - x^T \Sigma^{-1} \mu_k - \mu_k^T \Sigma^{-1} x + \mu_k^T \Sigma^{-1} \mu_k) + \ln(\pi_k)$$

= common term with equal
values in discriminant function

$$\delta_k(x) = \frac{1}{2} x^T \Sigma^{-1} \mu_k + \frac{1}{2} \mu_k^T \Sigma^{-1} x - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \ln(\pi_k)$$

we know $x^T \Sigma^{-1} \mu_k$ is scalar \therefore

$$\begin{aligned} x^T \Sigma^{-1} \mu_k &= (x^T \Sigma^{-1} \mu_k)^T \\ &= \mu_k^T (\Sigma^{-1})^T x \\ &= \mu_k^T \Sigma^{-1} x \end{aligned}$$

but $\Sigma^{-1} = \Sigma^{-1}$ (diagonal)
& Σ^{-1} is symmetric

$$S_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \ln(\pi_k)$$

We know $i=1, 2$ $\pi_i = \frac{N_i}{N}$

According to majority rule we pick $\delta_2(x)$ if $\delta_2(x) > \delta_1(x)$

Expanding this inequality

$$x^T \Sigma^{-1} \mu_2 - \frac{1}{2} \mu_2^T \Sigma^{-1} \mu_2 + \ln\left(\frac{N_2}{N}\right) > x^T \Sigma^{-1} \mu_1 - \frac{1}{2} \mu_1^T \Sigma^{-1} \mu_1 + \ln\left(\frac{N_1}{N}\right)$$

$$\Rightarrow x^T \Sigma^{-1} \mu_2 - x^T \Sigma^{-1} \mu_1 > \frac{1}{2} \mu_2^T \Sigma^{-1} \mu_2 - \frac{1}{2} \mu_1^T \Sigma^{-1} \mu_1 - \left(\ln\left(\frac{N_2}{N}\right) - \ln\left(\frac{N_1}{N}\right) \right)$$

$$\Rightarrow x^T \Sigma^{-1} (\mu_2 - \mu_1) > \frac{1}{2} \mu_2^T \Sigma^{-1} \mu_2 - \frac{1}{2} \mu_1^T \Sigma^{-1} \mu_1 + \ln\left(\frac{N_1}{N}\right) - \ln\left(\frac{N_2}{N}\right)$$

~~QED~~ Shown

b) If $[\beta_0 \ \beta]^T$ is a solution to the eqn it should satisfy the normal equation:

$$X^T X \begin{bmatrix} \beta_0 \\ \beta \end{bmatrix} = X^T \beta y.$$

$$\Rightarrow \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_{N_1+N_2} \end{bmatrix} \begin{bmatrix} 1 & x_1^T \\ 1 & x_2^T \\ \vdots & \vdots \\ 1 & x_{N_1+N_2}^T \end{bmatrix} = \begin{bmatrix} N & \sum_{i=1}^N x_i^T \\ \sum_{i=1}^N x_i & \sum_{i=1}^N x_i x_i^T \end{bmatrix}$$

If we modify the response to $\frac{-N}{N_1}$ for class 1 & $\frac{N}{N_2}$ where

$N = N_1 + N_2$ then ~~BBP8~~:

$$\Rightarrow X^T y = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_N \end{bmatrix} \begin{bmatrix} -N/N_1 \\ -N/N_2 \\ \vdots \\ N/N_2 \end{bmatrix}$$

$$= \begin{bmatrix} N_1(-N/N_1) + N_2(N/N_2) \\ \left(\sum_{i=1}^{N_1} x_i\right)(-N/N_1) + \left(\sum_{i=N_1+1}^N x_i\right)(N/N_2) \end{bmatrix} = \begin{bmatrix} 0 \\ -N\mu_1 + N\mu_2 \end{bmatrix}$$

4

where μ_1 & μ_2 are the means of each class.

For $X^T X$ we simplify:

$$\sum_{i=1}^N x_i = \sum_{i=1}^{N_1} x_i + \sum_{i=N_1+1}^N x_i \\ = N_1 \mu_1 + N_2 \mu_2$$

As seen in the book (HTF) we show covariance matrix $\hat{\Sigma}$

$$\hat{\Sigma} = \frac{1}{N-K} \sum_{k=1}^K \sum_{i: g_i=k} (x_i - \mu_k)(x_i - \mu_k)^T \quad g_i \text{ is the group.}$$

∴ This is 2-class classification $K=2$

$$\begin{aligned} \hat{\Sigma} &= \frac{1}{N-2} \left[\sum_{i: g_i=1} (x_i - \mu_1)(x_i - \mu_1)^T + \sum_{i: g_i=2} (x_i - \mu_2)(x_i - \mu_2)^T \right] \\ &= \frac{1}{N-2} \left[\sum_{i: g_i=1} x_i x_i^T - N_1 \mu_1 \mu_1^T + \sum_{i: g_i=2} (x_i - \mu_2) x_i^T - N_2 \mu_2 \mu_2^T \right] \end{aligned}$$

$\sum_{i=1}^N x_i x_i^T = (N-2)\hat{\Sigma} + N_1 \mu_1 \mu_1^T + N_2 \mu_2 \mu_2^T$

Substituting in matrix multiplication

$$\begin{bmatrix} N & N_1 \mu_1^T + N_2 \mu_2^T \\ N_1 \mu_1 + N_2 \mu_2 & (N-2)\hat{\Sigma} + N_1 \mu_1 \mu_1^T + N_2 \mu_2 \mu_2^T \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta \end{bmatrix} = \begin{bmatrix} 0 \\ -N \mu_1 + N \mu_2 \end{bmatrix}$$

$$\therefore N\beta_0 + \beta(N_1 \mu_1^T + N_2 \mu_2^T) = 0$$

$$\therefore \beta_0 = -\frac{\beta}{N}(N_1 \mu_1^T + N_2 \mu_2^T)$$

∴ Putting this in the second matrix equation

$$(N_1 \mu_1 + N_2 \mu_2) \left(\frac{N_1 \mu_1^T + N_2 \mu_2^T}{N} \right) (-\beta) + ((N-2)\hat{\Sigma} + N_1 \mu_1 \mu_1^T + N_2 \mu_2 \mu_2^T)(\beta) = N(\mu_2 - \mu_1)$$

$$\Rightarrow \left(\frac{-N_1^2 \mu_1 \mu_1^T - N_2^2 \mu_2 \mu_2^T}{N} - \frac{2N_1 N_2 \mu_1 \mu_2^T + N_1 \mu_1 \mu_2^T + N_2 \mu_2 \mu_1^T}{N} \right) \quad \begin{array}{l} \text{(expanding part} \\ \text{of the eqn)} \end{array}$$

$$\Rightarrow \left[\frac{-N_1^2 + N_1}{N} \mu_1 \mu_1^T - \frac{2N_1 N_2}{N} \mu_1 \mu_2^T + \left(\frac{-N_2^2 + N_2}{N} \mu_2 \mu_2^T \right) \right]$$

$$\Rightarrow \left[\frac{N_1}{N} (-N_1 + N) \mu_1 \mu_1^T - \frac{2N_1 N_2}{N} \mu_1 \mu_2^T + \frac{N_2}{N} (N - N_2) \mu_2 \mu_2^T \right]$$

$$\Rightarrow \left[\frac{N_1 N_2}{N} \mu_1 \mu_1^T - \frac{2N_1 N_2}{N} \mu_1 \mu_2^T + \frac{N_1 N_2}{N} \mu_2 \mu_2^T \right]$$

$$\Rightarrow \left[\frac{N_1 N_2}{N} (\mu_1 \mu_1^T - 2\mu_1 \mu_2^T - \mu_2 \mu_2^T) \right]$$

$$\Rightarrow \left[\frac{N_1 N_2}{N} (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \right]$$

let $\hat{\Sigma}_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T$ as stated in the question

$$\therefore \left[(N-2)\hat{\Sigma} + \frac{N_1 N_2}{N} \hat{\Sigma}_B \right] \beta = N(\mu_2 - \mu_1)$$

shown

c) We know $\hat{\Sigma}_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T$

$$\hat{\Sigma}_B \beta = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T (\beta)$$

$\therefore \hat{\Sigma}_B \beta$ is a scalar value \therefore no direction

$$\left[(N-2)\hat{\Sigma} + \frac{N_1 N_2}{N} \hat{\Sigma}_B \right] \beta = N(\mu_2 - \mu_1)$$

We already know $\frac{N_1 N_2}{N} \hat{\Sigma}_B$ is in the direction of $(\mu_2 - \mu_1)$

$$\therefore \text{Solution } \beta = (\hat{\Sigma}^{-1}) \frac{N}{N_2 - 2} (\mu_2 - \mu_1)$$

$$\beta \propto (\hat{\Sigma}^{-1}) (\mu_2 - \mu_1)$$

d) Revisiting the previous problem

$$\beta_0 = \frac{1}{N}$$

Let $U_i \in \mathbb{R}^n$ be class indicator of group i
 We have 2 classes $\therefore U = U_1 + U_2$
 where U has all entries equal to 1.

Let us encode classes arbitrarily

$$\therefore \text{let } a_1 = -N/N_1 \quad a_2 = N/N_2$$

$$\therefore Y = U_1 a_1 + U_2 a_2 \quad \text{--- (5)}$$

$$\therefore \text{RSS}(\beta, \beta_0) = \sum_{i=1}^N (y_i - \beta_0 - \beta^T x_i)^2 = (Y - \beta_0 U - X\beta)^T (Y - \beta_0 U - X\beta)$$

Equating the gradient to 0.

$$\nabla_{\beta} \text{RSS} = 2X^T X \beta - 2X^T Y + 2\beta_0 X^T U = 0$$

$$\begin{aligned} \nabla_{\beta_0} \text{RSS} &= 2U^T U \beta_0 - 2U^T (Y - X\beta) \\ &= 2N\beta_0 - 2U^T (Y - X\beta) = 0 \end{aligned}$$

$$\Rightarrow 2N\beta_0 = 2U^T (Y - X\beta)$$

$$\boxed{\beta_0 = \frac{1}{N} U^T (Y - X\beta)} \quad \text{--- (2)}$$

Plugging this value:

$$\left(X^T X - \frac{1}{N} X^T U U^T X \right) \hat{\beta} = X^T Y - \frac{1}{N} X^T U U^T Y$$

We know $X^T U_i = N_i \hat{\mu}_i$ for $i = 1, 2$

$$\text{LHS: } X^T U = X^T (U_1 + U_2) = N_1 \hat{\mu}_1 + N_2 \hat{\mu}_2$$

$$X^T X - \frac{1}{N} (N_1^2 \hat{\mu}_1 \hat{\mu}_1^T + N_2^2 \hat{\mu}_2 \hat{\mu}_2^T + N_1 N_2 \hat{\mu}_1 \hat{\mu}_2^T + N_1 N_2 \hat{\mu}_2 \hat{\mu}_1^T) \quad \text{--- (4)}$$

$$(N-2)\hat{\Sigma} = X^T X - N_1 \hat{\mu}_1 \hat{\mu}_1^T - N_2 \hat{\mu}_2 \hat{\mu}_2^T \quad (\text{matrix used in LDA})$$

$$\text{We already know } \hat{\Sigma}_{\beta} = (\hat{\mu}_2 - \hat{\mu}_1)(\hat{\mu}_2 - \hat{\mu}_1)^T$$

$$\therefore (N-2)\hat{\Sigma} + \frac{N_1 N_2}{N} \hat{\Sigma}_{\beta} = X^T X + \left(\frac{N_1 N_2 - N_1}{N} \right) \hat{\mu}_1 \hat{\mu}_1^T + \left(\frac{N_1 N_2 - N_2}{N} \right) \hat{\mu}_2 \hat{\mu}_2^T$$

$$- \frac{N_1 N_2}{N} \hat{\mu}_2 \hat{\mu}_1^T - \frac{N_1 N_2}{N} \hat{\mu}_1 \hat{\mu}_2^T$$

$$= X^T X - \frac{N_1^2}{N} \hat{\mu}_1 \hat{\mu}_1^T - \frac{N_2^2}{N} \hat{\mu}_2 \hat{\mu}_2^T - \frac{N_1 N_2}{N} (\hat{\mu}_2 \hat{\mu}_1^T + \hat{\mu}_1 \hat{\mu}_2^T) \text{ which is (4)}$$

$$\text{RHS} \stackrel{I}{=} X^T Y = X^T(a_1 U_1 + a_2 U_2) \\ = a_1 N_1 U_1 + a_2 N_2 U_2$$

$$\stackrel{II}{=} \frac{1}{N} X^T U U^T Y = \frac{1}{N} (N_1 \bar{U}_1 + N_2 \bar{U}_2)(a_1 N_1 + a_2 N_2) \\ = \frac{1}{N} (a_1 N_1^2 + a_2 N_1 N_2) \bar{U}_1 + (a_2 N_2^2 + a_1 N_1 N_2) \bar{U}_2$$

We know $N_1 = N_1 + N_2$ (combining combns I & II)

$$\Rightarrow \frac{1}{N} (a_1 N_1 N_2 - a_2 N_1 N_2) \bar{U}_1 + (a_2 N_1 N_2 - a_1 N_1 N_2) \bar{U}_2$$

$$\Rightarrow \frac{N_1 N_2}{N} (a_1 - a_2) (\bar{U}_1 - \bar{U}_2)$$

At this point we can see that a_1, a_2 can be any encoding & the proof following this step will give the same result as part c)
 \therefore This is true for any arbitrary encoding

c) When we assume/encode $a_1 = N/N_1$ & $a_2 = N/N_2$

$$U^T Y = a_1 N_1 + a_2 N_2 \\ = (-N/N_1) N_1 + (N/N_2) N_2 = 0$$

To calculate value of $\hat{\beta}_0$:

$$\hat{\beta}_0 = \frac{1}{N} (U^T X) \hat{\beta} \approx \frac{1}{N} (\bar{U}_1 N_1 + \bar{U}_2 N_2) \hat{\beta}$$

\therefore decision function $S(x)$ where $x \in \mathbb{R}$

$$S(x) = x^T \hat{\beta} + \hat{\beta}_0 = x^T \hat{\beta} - \frac{1}{N} (\bar{U}_1 N_1 + \bar{U}_2 N_2) \hat{\beta} \\ = \hat{\beta} \left(x^T - \frac{\bar{U}_1 N_1}{N} - \frac{\bar{U}_2 N_2}{N} \right)$$

(6)

We know $\hat{\beta} = \lambda \hat{\Sigma}^{-1} (\mu_2 - \mu_1)$ for a $\lambda \in \mathbb{R}$
 from part a) of the solution:

$$x^T \hat{\Sigma}^{-1} (\mu_2 - \mu_1) > \frac{1}{2} \mu_2^T \hat{\Sigma}^{-1} \hat{\mu}_2 - \frac{1}{2} \hat{\mu}_1^T \hat{\Sigma}^{-1} \hat{\mu}_1 + \log\left(\frac{N_1}{N}\right) - \log\left(\frac{N_2}{N}\right)$$

If $N_1 = N_2$ then this equation will be simplified

$$x^T \hat{\Sigma}^{-1} (\hat{\mu}_2 - \hat{\mu}_1) > \frac{1}{2} \hat{\mu}_2^T \hat{\Sigma}^{-1} \hat{\mu}_2 - \frac{1}{2} \hat{\mu}_1^T \hat{\Sigma}^{-1} \hat{\mu}_1$$

∴ This simplifies $S(x)$.

$$x^T \hat{\Sigma}^{-1} (\hat{\mu}_2 - \hat{\mu}_1) > \left(\frac{N_1}{N} \hat{\mu}_1 + \frac{N_2}{N} \hat{\mu}_2 \right)^T \hat{\Sigma}^{-1} (\hat{\mu}_2 - \hat{\mu}_1)$$

R Notebook

Mitali Yadav (3034158469)

Q4.

```
install.packages("fda.usc")
install.packages("pls")

library("fda.usc")

## Warning: package 'fda.usc' was built under R version 4.0.4

## Loading required package: fda

## Warning: package 'fda' was built under R version 4.0.4

## Loading required package: splines

## Loading required package: Matrix

## Loading required package: fds

## Warning: package 'fds' was built under R version 4.0.4

## Loading required package: rainbow

## Warning: package 'rainbow' was built under R version 4.0.4

## Loading required package: MASS

## Loading required package: pcaPP

## Warning: package 'pcaPP' was built under R version 4.0.3

## Loading required package: RCurl

## Warning: package 'RCurl' was built under R version 4.0.3

##
## Attaching package: 'fda'
```

```

## The following object is masked from 'package:graphics':
##
##      matplot

## Loading required package: mgcv

## Loading required package: nlme

## This is mgcv 1.8-31. For overview type 'help("mgcv-package")'.

##
## -----
## Functional Data Analysis and Utilities for Statistical Computing
## fda.usc version 2.0.2 (built on 2020-02-17) is now loaded
## fda.usc is running sequentially usign foreach package
## Please, execute ops.fda.usc() once to run in local parallel mode
## Deprecated functions: min.basis, min.np, anova.hetero, anova.onefactor, anova.RPm
## New functions: optim.basis, optim.np, fanova.hetero, fanova.onefactor, fanova.RPm
## -----
library("pls")

## Warning: package 'pls' was built under R version 4.0.4

##
## Attaching package: 'pls'

## The following object is masked from 'package:stats':
##
##      loadings

library("ggplot2")

#importing the CSV files (as an alternate method)
X_data = read.csv("tecator_X.csv")
Y_data = read.csv("tecator_Y.csv")

#combining them using cbind into a matrix
data = cbind(X_data, Y_data)
data = as.data.frame(data)
typeof(data)

## [1] "list"

dim(data)

## [1] 215 101

```

```

#splitting into training and testing
tec_train = data[1:172,]
tec_test = data[173:215,]

head(tec_train)

##          X1          X2          X3          X4          X5          X6          X7          X8          X9
## 1 2.61776 2.61814 2.61859 2.61912 2.61981 2.62071 2.62186 2.62334 2.62511
## 2 2.83454 2.83871 2.84283 2.84705 2.85138 2.85587 2.86060 2.86566 2.87093
## 3 2.58284 2.58458 2.58629 2.58808 2.58996 2.59192 2.59401 2.59627 2.59873
## 4 2.82286 2.82460 2.82630 2.82814 2.83001 2.83192 2.83392 2.83606 2.83842
## 5 2.78813 2.78989 2.79167 2.79350 2.79538 2.79746 2.79984 2.80254 2.80553
## 6 3.00993 3.01540 3.02086 3.02634 3.03190 3.03756 3.04341 3.04955 3.05599
##          X10         X11         X12         X13         X14         X15         X16         X17         X18
## 1 2.62722 2.62964 2.63245 2.63565 2.63933 2.64353 2.64825 2.65350 2.65937
## 2 2.87661 2.88264 2.88898 2.89577 2.90308 2.91097 2.91953 2.92873 2.93863
## 3 2.60131 2.60414 2.60714 2.61029 2.61361 2.61714 2.62089 2.62486 2.62909
## 4 2.84097 2.84374 2.84664 2.84975 2.85307 2.85661 2.86038 2.86437 2.86860
## 5 2.80890 2.81272 2.81704 2.82184 2.82710 2.83294 2.83945 2.84664 2.85458
## 6 3.06274 3.06982 3.07724 3.08511 3.09343 3.10231 3.11185 3.12205 3.13294
##          X19         X20         X21         X22         X23         X24         X25         X26         X27
## 1 2.66585 2.67281 2.68008 2.68733 2.69427 2.70073 2.70684 2.71281 2.71914
## 2 2.94929 2.96072 2.97272 2.98493 2.99690 3.00833 3.01920 3.02990 3.04101
## 3 2.63361 2.63835 2.64330 2.64838 2.65354 2.65870 2.66375 2.66880 2.67383
## 4 2.87308 2.87789 2.88301 2.88832 2.89374 2.89917 2.90457 2.90991 2.91521
## 5 2.86331 2.87280 2.88291 2.89335 2.90374 2.91371 2.92305 2.93187 2.94060
## 6 3.14457 3.15703 3.17038 3.18429 3.19840 3.21225 3.22552 3.23827 3.25084
##          X28         X29         X30         X31         X32         X33         X34         X35         X36
## 1 2.72628 2.73462 2.74416 2.75466 2.76568 2.77679 2.78790 2.79949 2.81225
## 2 3.05345 3.06777 3.08416 3.10221 3.12106 3.13983 3.15810 3.17623 3.19519
## 3 2.67892 2.68411 2.68937 2.69470 2.70012 2.70563 2.71141 2.71775 2.72490
## 4 2.92043 2.92565 2.93082 2.93604 2.94128 2.94658 2.95202 2.95777 2.96419
## 5 2.94986 2.96035 2.97241 2.98606 3.00097 3.01652 3.03220 3.04793 3.06413
## 6 3.26393 3.27851 3.29514 3.31401 3.33458 3.35591 3.37709 3.39772 3.41828
##          X37         X38         X39         X40         X41         X42         X43         X44         X45
## 1 2.82706 2.84356 2.86106 2.87857 2.89497 2.90924 2.92085 2.93015 2.93846
## 2 3.21584 3.23747 3.25889 3.27835 3.29384 3.30362 3.30681 3.30393 3.29700
## 3 2.73344 2.74327 2.75433 2.76642 2.77931 2.79272 2.80649 2.82064 2.83541
## 4 2.97159 2.98045 2.99090 3.00284 3.01611 3.03048 3.04579 3.06194 3.07889
## 5 3.08153 3.10078 3.12185 3.14371 3.16510 3.18470 3.20140 3.21477 3.22544
## 6 3.43974 3.46266 3.48663 3.51002 3.53087 3.54711 3.55699 3.55986 3.55656
##          X46         X47         X48         X49         X50         X51         X52         X53         X54
## 1 2.94771 2.96019 2.97831 3.00306 3.03506 3.07428 3.11963 3.16868 3.21771
## 2 3.28925 3.28409 3.28505 3.29326 3.30923 3.33267 3.36251 3.39661 3.43188
## 3 2.85121 2.86872 2.88905 2.91289 2.94088 2.97325 3.00946 3.04780 3.08554
## 4 3.09686 3.11629 3.13775 3.16217 3.19068 3.22376 3.26172 3.30379 3.34793
## 5 3.23505 3.24586 3.26027 3.28063 3.30889 3.34543 3.39019 3.44198 3.49800
## 6 3.54937 3.54169 3.53692 3.53823 3.54760 3.56512 3.59043 3.62229 3.65830
##          X55         X56         X57         X58         X59         X60         X61         X62         X63
## 1 3.26254 3.29988 3.32847 3.34899 3.36342 3.37379 3.38152 3.38741 3.39164
## 2 3.46492 3.49295 3.51458 3.53004 3.54067 3.54797 3.55306 3.55675 3.55921
## 3 3.11947 3.14696 3.16677 3.17938 3.18631 3.18924 3.18950 3.18801 3.18498
## 4 3.39093 3.42920 3.45998 3.48227 3.49687 3.50558 3.51026 3.51221 3.51215

```

```

## 5 3.55407 3.60534 3.64789 3.68011 3.70272 3.71815 3.72863 3.73574 3.74059
## 6 3.69515 3.72932 3.75803 3.78003 3.79560 3.80614 3.81313 3.81774 3.82079
## X64 X65 X66 X67 X68 X69 X70 X71 X72
## 1 3.39418 3.39490 3.39366 3.39045 3.38541 3.37869 3.37041 3.36073 3.34979
## 2 3.56045 3.56034 3.55876 3.55571 3.55132 3.54585 3.53950 3.53235 3.52442
## 3 3.18039 3.17411 3.16611 3.15641 3.14512 3.13241 3.11843 3.10329 3.08714
## 4 3.51036 3.50682 3.50140 3.49398 3.48457 3.47333 3.46041 3.44595 3.43005
## 5 3.74357 3.74453 3.74336 3.73991 3.73418 3.72638 3.71676 3.70553 3.69289
## 6 3.82258 3.82301 3.82206 3.81959 3.81557 3.81021 3.80375 3.79642 3.78835
## X73 X74 X75 X76 X77 X78 X79 X80 X81
## 1 3.33769 3.32443 3.31013 3.29487 3.27891 3.26232 3.24542 3.22828 3.21080
## 2 3.51583 3.50668 3.49700 3.48683 3.47626 3.46552 3.45501 3.44481 3.43477
## 3 3.07014 3.05237 3.03393 3.01504 2.99569 2.97612 2.95642 2.93660 2.91667
## 4 3.41285 3.39450 3.37511 3.35482 3.33376 3.31204 3.28986 3.26730 3.24442
## 5 3.67900 3.66396 3.64785 3.63085 3.61305 3.59463 3.57582 3.55695 3.53796
## 6 3.77958 3.77024 3.76040 3.75005 3.73929 3.72831 3.71738 3.70681 3.69664
## X82 X83 X84 X85 X86 X87 X88 X89 X90
## 1 3.19287 3.17433 3.15503 3.13475 3.11339 3.09116 3.06850 3.04596 3.02393
## 2 3.42465 3.41419 3.40303 3.39082 3.37731 3.36265 3.34745 3.33245 3.31818
## 3 2.89655 2.87622 2.85563 2.83474 2.81361 2.79235 2.77113 2.75015 2.72956
## 4 3.22117 3.19757 3.17357 3.14915 3.12429 3.09908 3.07366 3.04825 3.02308
## 5 3.51880 3.49936 3.47938 3.45869 3.43711 3.41458 3.39129 3.36772 3.34450
## 6 3.68659 3.67649 3.66611 3.65503 3.64283 3.62938 3.61483 3.59990 3.58535
## X91 X92 X93 X94 X95 X96 X97 X98 X99
## 1 3.00247 2.98145 2.96072 2.94013 2.91978 2.89966 2.87964 2.85960 2.83940
## 2 3.30473 3.29186 3.27921 3.26655 3.25369 3.24045 3.22659 3.21181 3.19600
## 3 2.70934 2.68951 2.67009 2.65112 2.63262 2.61461 2.59718 2.58034 2.56404
## 4 2.99820 2.97367 2.94951 2.92576 2.90251 2.87988 2.85794 2.83672 2.81617
## 5 3.32201 3.30025 3.27907 3.25831 3.23784 3.21765 3.19766 3.17770 3.15770
## 6 3.57163 3.55877 3.54651 3.53442 3.52221 3.50972 3.49682 3.48325 3.46870
## X100 y
## 1 2.81920 22.5
## 2 3.17942 40.1
## 3 2.54816 8.4
## 4 2.79622 5.9
## 5 3.13753 25.5
## 6 3.45307 42.7

```

1. PC Regression

```

#using the PLS package to perform pc regression
set.seed(18)
pqr_fit <- pqr(y ~ ., data= tec_train, scale=TRUE, validation="CV")

summary(pqr_fit)

```

```

## Data:      X dimension: 172 100
## Y dimension: 172 1
## Fit method: svdpc
## Number of components considered: 100
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.

```

```

##          (Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps
## CV         12.72    11.33   11.08   7.638   4.363   3.403   3.172
## adjCV     12.72    11.32   11.07   7.626   4.355   3.393   3.163
##          7 comps 8 comps 9 comps 10 comps 11 comps 12 comps 13 comps
## CV         3.140    3.146   3.034   3.066   2.815   2.862   2.883
## adjCV     3.129    3.134   3.020   3.050   2.798   2.844   2.864
##          14 comps 15 comps 16 comps 17 comps 18 comps 19 comps 20 comps
## CV         2.856    2.817   2.686   2.666   2.604   2.653   2.597
## adjCV     2.838    2.792   2.646   2.620   2.595   2.616   2.550
##          21 comps 22 comps 23 comps 24 comps 25 comps 26 comps 27 comps
## CV         2.628    2.692   2.670   2.802   2.826   2.758   2.708
## adjCV     2.582    2.649   2.617   2.747   2.772   2.694   2.670
##          28 comps 29 comps 30 comps 31 comps 32 comps 33 comps 34 comps
## CV         2.648    2.617   2.578   2.604   2.753   2.770   2.840
## adjCV     2.602    2.559   2.525   2.549   2.678   2.698   2.765
##          35 comps 36 comps 37 comps 38 comps 39 comps 40 comps 41 comps
## CV         2.802    3.119   3.131   3.086   3.184   3.263   3.619
## adjCV     2.732    3.032   3.051   2.982   3.089   3.152   3.496
##          42 comps 43 comps 44 comps 45 comps 46 comps 47 comps 48 comps
## CV         3.267    3.422   3.599   3.458   3.574   3.900   3.527
## adjCV     3.153    3.304   3.473   3.331   3.429   3.747   3.387
##          49 comps 50 comps 51 comps 52 comps 53 comps 54 comps 55 comps
## CV         3.460    3.390   4.152   4.234   3.937   3.766   3.82
## adjCV     3.325    3.259   3.975   4.055   3.778   3.617   3.67
##          56 comps 57 comps 58 comps 59 comps 60 comps 61 comps 62 comps
## CV         3.834    3.858   4.064   3.793   3.536   3.209   3.205
## adjCV     3.670    3.695   3.886   3.629   3.389   3.078   3.070
##          63 comps 64 comps 65 comps 66 comps 67 comps 68 comps 69 comps
## CV         3.262    3.258   3.290   3.271   3.182   3.112   2.884
## adjCV     3.124    3.122   3.152   3.136   3.056   2.983   2.770
##          70 comps 71 comps 72 comps 73 comps 74 comps 75 comps 76 comps
## CV         2.860    2.811   2.817   2.755   2.733   2.717   2.728
## adjCV     2.748    2.704   2.700   2.644   2.625   2.613   2.622
##          77 comps 78 comps 79 comps 80 comps 81 comps 82 comps 83 comps
## CV         2.679    2.658   2.667   2.723   2.833   2.858   2.979
## adjCV     2.577    2.555   2.565   2.621   2.725   2.749   2.863
##          84 comps 85 comps 86 comps 87 comps 88 comps 89 comps 90 comps
## CV         3.081    3.128   3.000   3.401   3.593   3.725   3.546
## adjCV     2.958    3.006   2.873   3.251   3.430   3.555   3.385
##          91 comps 92 comps 93 comps 94 comps 95 comps 96 comps 97 comps
## CV         3.594    3.924   3.936   3.892   4.084   4.031   4.089
## adjCV     3.430    3.743   3.754   3.712   3.894   3.844   3.901
##          98 comps 99 comps 100 comps
## CV         3.965    4.096   3.847
## adjCV     3.780    3.904   3.668
##
## TRAINING: % variance explained
##          1 comps 2 comps 3 comps 4 comps 5 comps 6 comps 7 comps 8 comps
## X         98.50   99.59   99.88   99.99   100.00  100.00  100.00  100.00
## y        22.32   26.16   65.31   88.91   93.51   94.38   94.58   94.67
##          9 comps 10 comps 11 comps 12 comps 13 comps 14 comps 15 comps
## X        100.00  100.00   100    100.00  100.00  100.00  100.00
## y        95.19   95.29    96    96.01   96.01   96.13   96.5
##          16 comps 17 comps 18 comps 19 comps 20 comps 21 comps 22 comps

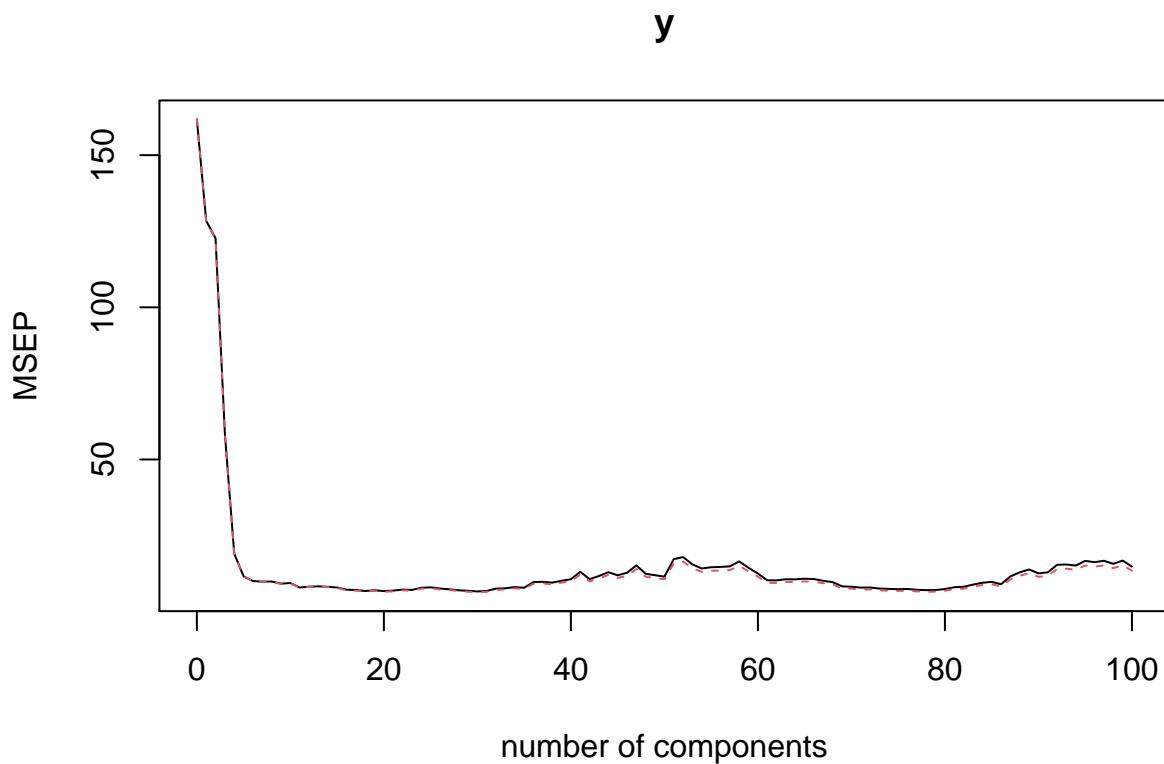
```

```

## X    100.00      100    100.00    100.00    100.00    100.00    100.00
## y    96.83       97    97.04     97.42     97.59     97.62     97.66
## 23 comps  24 comps  25 comps  26 comps  27 comps  28 comps  29 comps
## X    100.00      100.00   100.0    100.0     100.00    100.00    100.00
## y    97.75       97.76   97.8     97.9      97.91     98.09     98.17
## 30 comps  31 comps  32 comps  33 comps  34 comps  35 comps  36 comps
## X    100.00      100.00   100.00   100.00    100.00    100.00    100.0
## y    98.19       98.22   98.27    98.28     98.29     98.29     98.3
## 37 comps  38 comps  39 comps  40 comps  41 comps  42 comps  43 comps
## X    100.00      100.00   100.00   100.00    100.00    100.0     100.0
## y    98.31       98.46   98.48    98.57     98.57     98.7      98.7
## 44 comps  45 comps  46 comps  47 comps  48 comps  49 comps  50 comps
## X    100.00      100.00   100.00   100.00    100.00    100.00    100.00
## y    98.74       98.83   98.91    98.91     98.98     98.98     98.98
## 51 comps  52 comps  53 comps  54 comps  55 comps  56 comps  57 comps
## X    100.00      100.00   100.00   100.00    100.00    100.00    100.00
## y    99.01       99.02   99.02    99.04     99.07     99.14     99.14
## 58 comps  59 comps  60 comps  61 comps  62 comps  63 comps  64 comps
## X    100.00      100.00   100.00   100.0     100.00    100.00    100.00
## y    99.21       99.25   99.28    99.3      99.33     99.34     99.34
## 65 comps  66 comps  67 comps  68 comps  69 comps  70 comps  71 comps
## X    100.00      100.00   100.00   100.00    100.00    100.0     100.0
## y    99.34       99.34   99.34    99.38     99.38     99.4      99.4
## 72 comps  73 comps  74 comps  75 comps  76 comps  77 comps  78 comps
## X    100.00      100.00   100.00   100.00    100.00    100.00    100.00
## y    99.44       99.44   99.45    99.45     99.45     99.45     99.46
## 79 comps  80 comps  81 comps  82 comps  83 comps  84 comps  85 comps
## X    100.00      100.00   100.00   100.00    100.00    100.00    100.00
## y    99.46       99.46   99.47    99.47     99.48     99.49     99.49
## 86 comps  87 comps  88 comps  89 comps  90 comps  91 comps  92 comps
## X    100.00      100.00   100.00   100.00    100.00    100.00    100.00
## y    99.58       99.59   99.61    99.62     99.63     99.63     99.64
## 93 comps  94 comps  95 comps  96 comps  97 comps  98 comps  99 comps
## X    100.00      100.00   100.00   100.00    100.00    100.00    100.00
## y    99.65       99.65   99.65    99.66     99.66     99.68     99.69
## 100 comps
## X    100.0
## y    99.7

```

```
validationplot(pcr_fit, val.type = "MSEP")
```



```
#Number of components to be used should be 8 according to the validation plot
#this has been calculated using the validation plot shown below as well as the variance explained by nu
```

```
#root mean squared error
rmse = function(x,y) {sqrt(mean((x-y)^2))}

#calcualte r^2
ss_res = function(yi,y_hat) {sum((yi-y_hat)^2)}
ss_tot = function(yi) {sum((yi - mean(yi))^2)}
r2 = function(ss_res, ss_tot) {1 - (ss_res/ss_tot)}
rmseCV = RMSEP(pcr_fit, estimate ='CV');
rmseCV
```

| ## (Intercept) | 1 comps | 2 comps | 3 comps | 4 comps | 5 comps |
|----------------|----------|----------|----------|----------|----------|
| ## 12.719 | 11.333 | 11.076 | 7.638 | 4.363 | 3.403 |
| ## 6 comps | 7 comps | 8 comps | 9 comps | 10 comps | 11 comps |
| ## 3.172 | 3.140 | 3.146 | 3.034 | 3.066 | 2.815 |
| ## 12 comps | 13 comps | 14 comps | 15 comps | 16 comps | 17 comps |
| ## 2.862 | 2.883 | 2.856 | 2.817 | 2.686 | 2.666 |
| ## 18 comps | 19 comps | 20 comps | 21 comps | 22 comps | 23 comps |
| ## 2.604 | 2.653 | 2.597 | 2.628 | 2.692 | 2.670 |
| ## 24 comps | 25 comps | 26 comps | 27 comps | 28 comps | 29 comps |
| ## 2.802 | 2.826 | 2.758 | 2.708 | 2.648 | 2.617 |
| ## 30 comps | 31 comps | 32 comps | 33 comps | 34 comps | 35 comps |
| ## 2.578 | 2.604 | 2.753 | 2.770 | 2.840 | 2.802 |

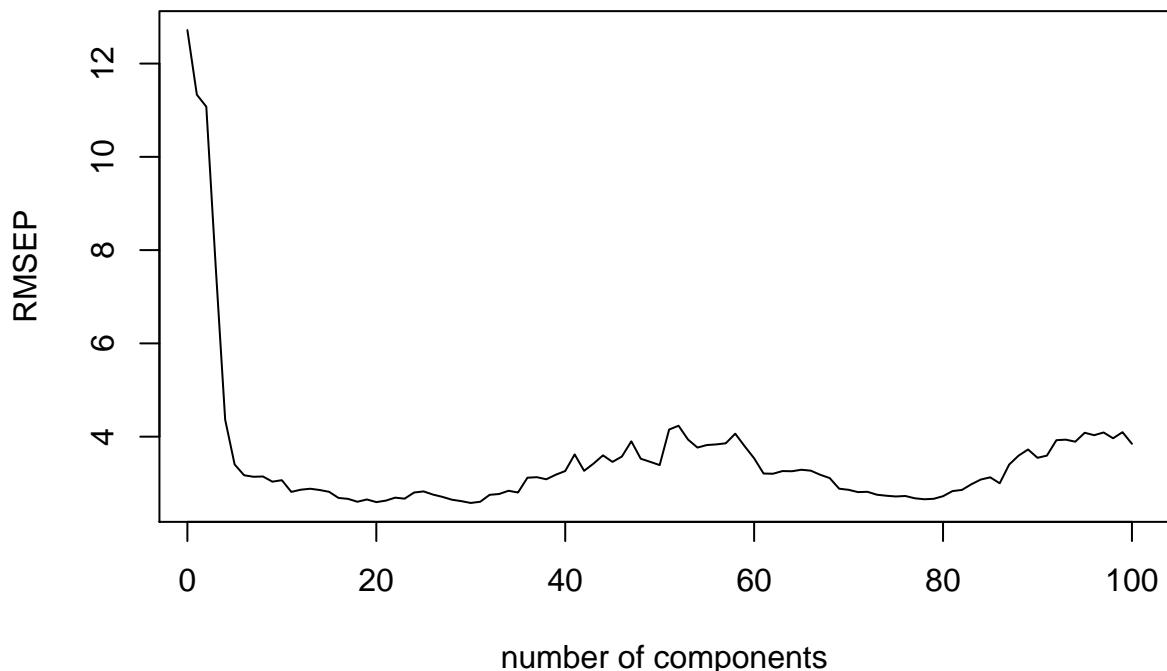
```

##   36 comps    37 comps    38 comps    39 comps    40 comps    41 comps
## 3.119        3.131       3.086       3.184       3.263       3.619
##   42 comps    43 comps    44 comps    45 comps    46 comps    47 comps
## 3.267        3.422       3.599       3.458       3.574       3.900
##   48 comps    49 comps    50 comps    51 comps    52 comps    53 comps
## 3.527        3.460       3.390       4.152       4.234       3.937
##   54 comps    55 comps    56 comps    57 comps    58 comps    59 comps
## 3.766        3.820       3.834       3.858       4.064       3.793
##   60 comps    61 comps    62 comps    63 comps    64 comps    65 comps
## 3.536        3.209       3.205       3.262       3.258       3.290
##   66 comps    67 comps    68 comps    69 comps    70 comps    71 comps
## 3.271        3.182       3.112       2.884       2.860       2.811
##   72 comps    73 comps    74 comps    75 comps    76 comps    77 comps
## 2.817        2.755       2.733       2.717       2.728       2.679
##   78 comps    79 comps    80 comps    81 comps    82 comps    83 comps
## 2.658        2.667       2.723       2.833       2.858       2.979
##   84 comps    85 comps    86 comps    87 comps    88 comps    89 comps
## 3.081        3.128       3.000       3.401       3.593       3.725
##   90 comps    91 comps    92 comps    93 comps    94 comps    95 comps
## 3.546        3.594       3.924       3.936       3.892       4.084
##   96 comps    97 comps    98 comps    99 comps    100 comps
## 4.031       4.089       3.965       4.096       3.847

```

```
plot(rmseCV, main='Predicted RMSE by number of components', xlim=c(1,100), xlab = "number of components")
```

Predicted RMSE by number of components



```
#predicting the values using 8 principal components
yhat = predict(pcr_fit, tec_test, ncomp=8)
```

```
# calculating the mse
mse_pred <- rmse(tec_test$y, yhat)^2
ss_res_pcr = ss_res(tec_test$y, yhat)
ss_tot_pcr = ss_tot(yhat)
r2_pcr = r2(ss_res_pcr, ss_tot_pcr)
r2_pcr
```

```
## [1] 0.9506282
```

```
rmse(tec_test$y, yhat)
```

```
## [1] 2.833257
```

2. Ridge Regression

```
#installing the packages
install.packages("glmnet")
```

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.0.4
```

```
## Loaded glmnet 4.1-1
```

```
#using the pre-split data from training and testing tec_train and tec_test
train_X = tec_train[,1:100]
test_X = tec_test[,1:100]
```

```
#scaling training and testing X
```

```
train_X = scale(train_X, center = T, scale = T)
```

```
test_X = scale(test_X, center = T, scale = T)
```

```
#convert into matrix
```

```
temp_train_X = as.matrix(train_X)
```

```
temp_test_X = as.matrix(test_X)
```

```
train_Y = tec_train[,101]
```

```
test_Y = tec_test[,101]
```

```
#fitting the ridge regression model
```

```
lambdas = 10^seq(2, -3, by = -.1)
```

```
ridge_reg = glmnet(temp_train_X, train_Y, nlambda = 25, alpha = 0, family = 'gaussian', lambda = lambdas)
```

```
summary(ridge_reg)
```

```
##          Length Class      Mode
## a0            51   -none-    numeric
## beta         5100  dgCMatrix S4
## df             51   -none-    numeric
```

```

## dim      2 -none- numeric
## lambda  51 -none- numeric
## dev.ratio 51 -none- numeric
## nulldev   1 -none- numeric
## npasses    1 -none- numeric
## jerr       1 -none- numeric
## offset     1 -none- logical
## call       7 -none- call
## nobs      1 -none- numeric

#performing cross validation
ridge_crossval <- cv.glmnet(temp_train_X, train_Y, alpha = 0, lambda = lambdas)

#thus we used the functions provided by the glmnet package to figure out the most optimal lambda
optimal_lambda <- ridge_crossval$lambda.min
optimal_lambda

```

```
## [1] 0.001
```

We were able to calculate the optimal lambda using cross-validation

```

#use the optimal lambda to predict the y_hat values and calculate MSE
rmse = function(x,y) {sqrt(mean((x-y)^2))}
mse = function(x,y) {mean((x-y)^2)}

y_hat_ridge = predict(ridge_reg, s = optimal_lambda, newx = temp_test_X)
mse_pred_ridge = mse(y_hat_ridge, test_Y)

ss_res_ridge = ss_res(test_Y, y_hat_ridge)
ss_tot_ridge = ss_tot(y_hat_ridge)
r2_ridge = r2(ss_res = ss_res_ridge, ss_tot = ss_tot_ridge)

mse_pred_ridge

```

```
## [1] 10.87534
```

```
r2_ridge
```

```
## [1] 0.9317082
```

3. Lasso Regression

```

lambdas <- 10^seq(2, -3, by = -.1)

#alpha=1 for lasso regression
lasso_reg = cv.glmnet(temp_train_X, train_Y, alpha = 1, lambda = lambdas, standardize=T)

optimal_lambda_lasso = lasso_reg$lambda.min
optimal_lambda_lasso

```

```

## [1] 0.001

#using the optimal lambda to predict the values of y_fat
y_hat_lasso = predict(lasso_reg, s = optimal_lambda_lasso, newx = temp_test_X)

#calculating the r2 for this method
ss_res_lasso = ss_res(test_Y, y_hat_lasso)
ss_tot_lasso = ss_tot(y_hat_lasso)
r2_lasso = r2(ss_res = ss_res_lasso, ss_tot = ss_tot_lasso)

#mean squared error being calcualted over the testing dataset
mse_pred_lasso = mse(y_hat_lasso, test_Y)
mse_pred_lasso

```

```
## [1] 11.02909
```

```
r2_lasso
```

```
## [1] 0.9315608
```

The best method so far has been the PCA Regression technique that had the lowest value of mean squared error on the same testing dataset.

Q5.

```

install.packages("pls")
install.packages("faraway")
install.packages("spls")
install.packages("caret")

library("pls")

## Warning: package 'pls' was built under R version 4.0.4

## Be aware that pls 0.0.1 contains experimental and partly untested code.
## Use cautiously.

##
## Attaching package: 'pls'

## The following object is masked from 'package:pls':
## 
##     loadings

## The following object is masked from 'package:stats':
## 
##     loadings

```

```

library("spls")

## Warning: package 'spls' was built under R version 4.0.4

## Sparse Partial Least Squares (SPLS) Regression and
## Classification (version 2.2-3)

library("tidyverse")

## -- Attaching packages ----- tidyverse 1.3.0 --

## v tibble 3.0.3      v dplyr   1.0.2
## v tidyverse 1.1.2    v stringr 1.4.0
## v readr   1.3.1      v forcats 0.5.0
## v purrr   0.3.4

## Warning: package 'dplyr' was built under R version 4.0.3

## Warning: package 'stringr' was built under R version 4.0.3

## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::collapse() masks nlme::collapse()
## x tidyverse::complete() masks RCurl::complete()
## x tidyverse::expand() masks Matrix::expand()
## x dplyr::filter() masks stats::filter()
## x dplyr::lag() masks stats::lag()
## x tidyverse::pack() masks Matrix::pack()
## x dplyr::select() masks MASS::select()
## x tidyverse::unpack() masks Matrix::unpack()

library("caret")

## Warning: package 'caret' was built under R version 4.0.4

## Loading required package: lattice

##
## Attaching package: 'lattice'

## The following object is masked from 'package:fda':
## 
##     melanoma

## Registered S3 methods overwritten by 'caret':
##   method      from
##   predict.splsda spls
##   print.splsda  spls

##
## Attaching package: 'caret'

```



```
## [1] 0.03158691

#using the optimal value of lambda to build the classifier
slgr_model = glmnet(pros_trainX, pros_trainY, alpha = 1, family = "binomial",
                     lambda = optimal_lambda_slgr)
```

Now that we have built the model, we can use it to predict the values for the test set as well as the training set

Part2. Building another classifier - Logistic Regression Model with L2 regularization

```
## [1] 0.9375
```

Comparing the 2 models We can say that based on the accuracy of the 2 models, the SLR with L1 regularization is more accurate. However, we can take a look at the ROC curve to further understand which model has a more accurate prediction.

```
install.packages("ROCR")
```

```
library(ROCR)
```

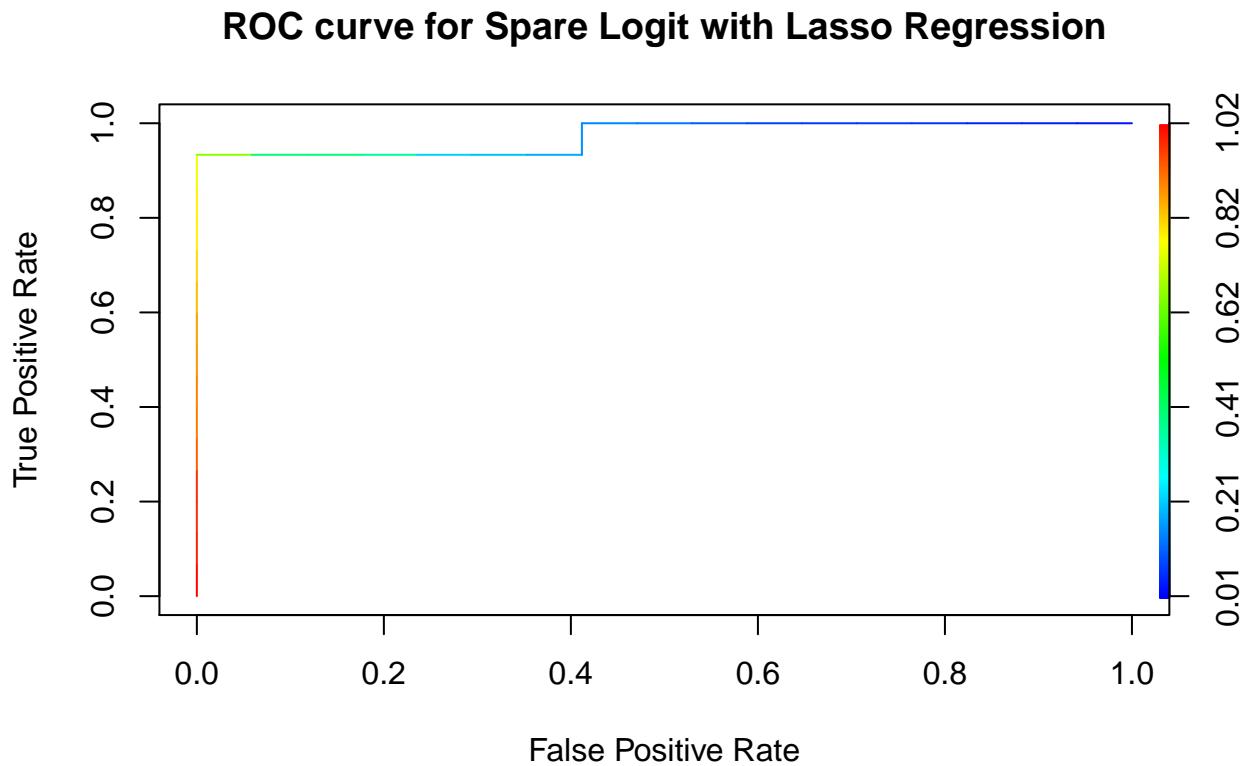
```
## Warning: package 'ROCR' was built under R version 4.0.4
```

```
thesePredictions<-yhat_slgr_prob  
theseLabels<-pros_testY
```

```
pred <- prediction(thesePredictions, theseLabels)
```

```
perf <- performance(pred, "tpr", "fpr")
```

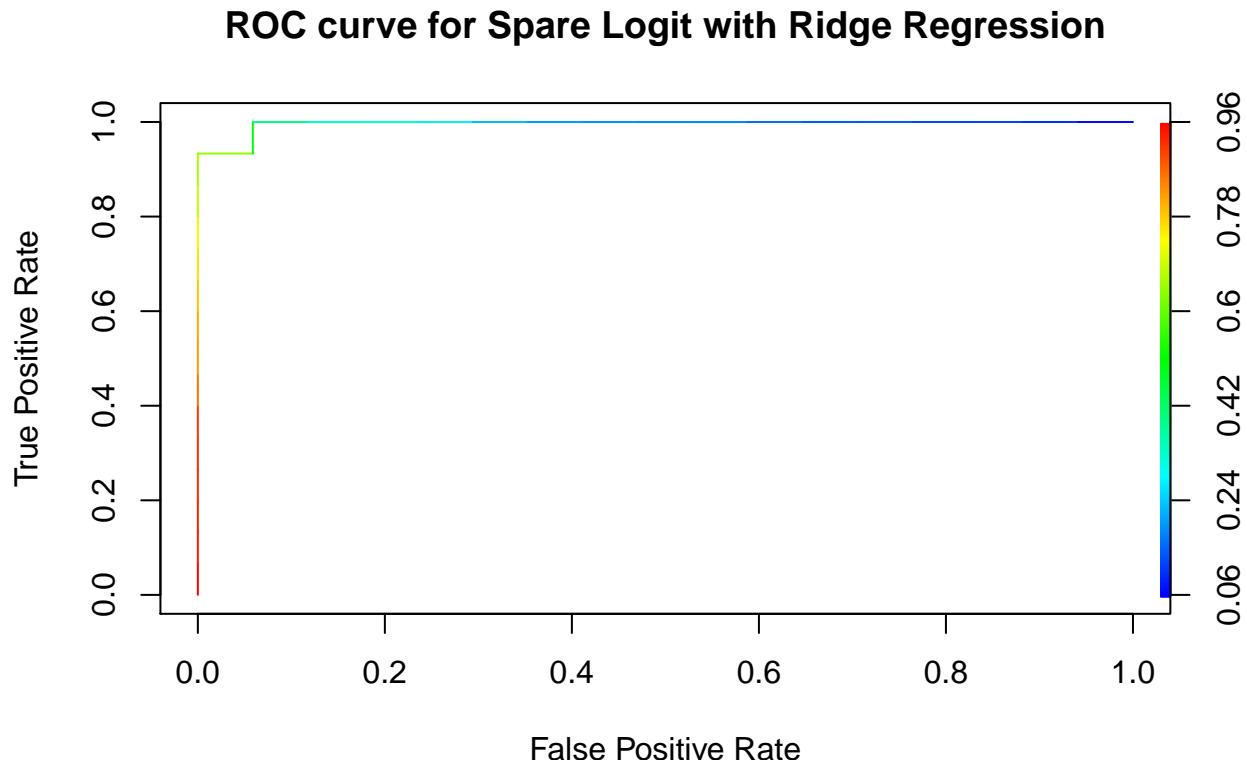
```
plot(perf, colorize=TRUE, main="ROC curve for Spare Logit with Lasso Regression", xlab="False Positive Rate", ylab="True Positive Rate")
```



```
#doing the same for L2  
thesePredictions2<-yhat_lgr_prob  
theseLabels2<-pros_testY
```

```
pred2 <- prediction(thesePredictions2, theseLabels2)
```

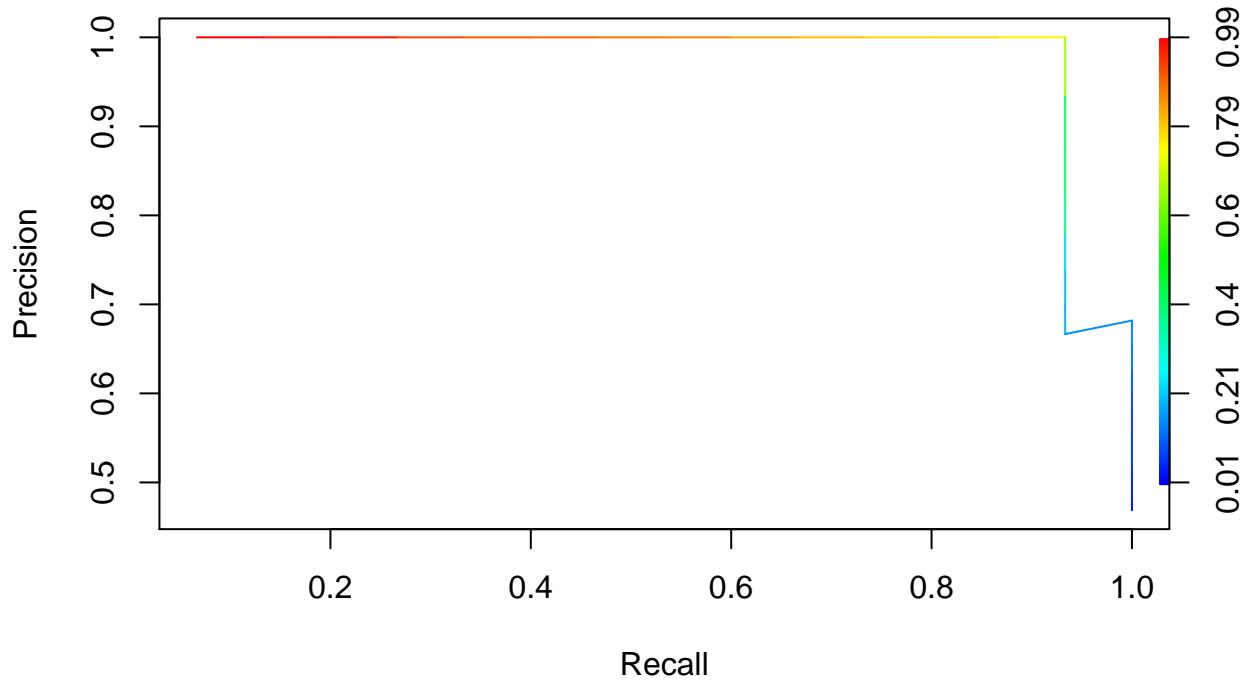
```
perf2 <- performance(pred2,"tpr","fpr")
plot(perf2,colorize=TRUE, main="ROC curve for Spare Logit with Ridge Regression", xlab="False Positive Rate")
```



Also plotting the Precision vs. recall curves for both regressions

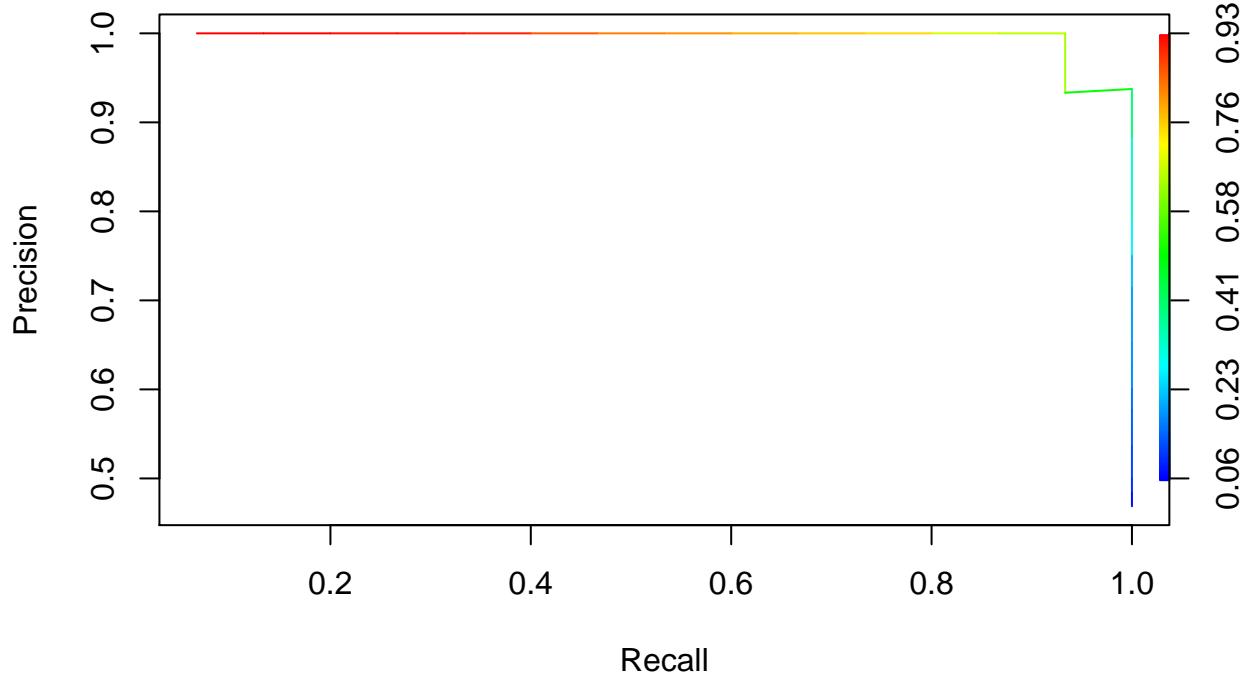
```
perfPR <- performance(pred, "prec", "rec")
plot(perfPR,colorize=TRUE,main="Precision-Recall curve for Spare Logit with Lasso Regression", xlab="Recall")
```

Precision–Recall curve for Spare Logit with Lasso Regression



```
#for l2
perfPR2 <- performance(pred2, "prec", "rec")
plot(perfPR2,colorize=TRUE,main="Precision-Recall curve for Spare Logit with Ridge Regression", xlab="R
```

Precision–Recall curve for Spare Logit with Ridge Regression



Based on both, the ROC curve and the precision-recall curve, we can confirm that the Lasso Regularization yields more accurate results compared to Ridge Regularization. In the ridge, the coefficients of the linear transformation are normal distributed and in the lasso they are Laplace distributed. In the lasso, this makes it easier for the coefficients to be zero and therefore easier to eliminate some of your input variable as not contributing to the output. Since we are using all the variables, this makes the model extremely specific and reduces the ability of the model to accurately predict on an unknown dataset. It increases variance and bias.