STAT 435 HW5

Peiran Chen

06/03/2022

1.

(a).

To perform a principal components regression, we can then fit the linear regression model

$$\begin{aligned} y_i &= \theta_0 + \theta_1 z_{i1} + ... + \theta_M z_{iM} + \varepsilon_i \\ &= \theta_0 + \sum_{m=1}^{M} \theta_m z_{im} + \varepsilon_i, \quad i = 1, ..., n \end{aligned}$$

using least squares. And the regression coefficients are given by $\theta_0, \theta_1, ..., \theta_M$.

(b).

After plug in Equation 1, we have

$$y_i = \theta_0 + \theta_1(\phi_{11}x_{i1} + \phi_{21}x_{i2} + ... + \phi_{p1}x_{ip}) + ... + \theta_M(\phi_{1M}x_{i1} + \phi_{2M}x_{i2} + ... + \phi_{pM}x_{ip}) + \varepsilon_i$$

(c).

To see that the principal components regression model is linear in the columns of X. We can expand the formula we get from part (b).

$$\begin{aligned} y_i &= \theta_0 + \theta_1(\phi_{11}x_{i1} + \phi_{21}x_{i2} + ... + \phi_{p1}x_{ip}) + ... + \theta_M(\phi_{1M}x_{i1} + \phi_{2M}x_{i2} + ... + \phi_{pM}x_{ip}) + \varepsilon_i \\ &= \theta_0 + x_{i1}(\theta_1\phi_{11} + ... + \theta_M\phi_{1M}) + x_{i2}(\theta_1\phi_{21} + ... + \theta_M\phi_{2M}) + ... + x_{ip}(\theta_1\phi_{p1} + ... + \theta_M\phi_{pM}) \\ &= \theta_0 + x_{i1}\left(\sum_{m=1}^{M}\theta_m\phi_{1m}\right) + x_{i2}\left(\sum_{m=1}^{M}\theta_m\phi_{2m}\right) + ... + x_{ip}\left(\sum_{m=1}^{M}\theta_m\phi_{pm}\right) \\ &= \theta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} \end{aligned}$$

Where $\beta_1 = \sum_{m=1}^{M}\theta_m\phi_{1m}, ..., \beta_p = \sum_{m=1}^{M}\theta_m\phi_{pm}$ are just linear and not quadratic or etc, they are constants. Hence, we are able to see that it's just a linear combination of xs. Hence, it's linear in the columns of X.

(d).

The Claim is False. In a regression setting, suppose we have the below linear model

$$Y = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p + \varepsilon$$

Then, our derivation of the PCR is just a special case of the linear model. Where the dimension reduction in our PCR serves to constrain the estimated β_p coefficients, since now they must take the form $\beta_p = \sum_{m=1}^{M} \theta_m \phi_{pm}$. This constraint on the form of the coefficients has the potential to bias the coefficient estimates. However, in situations where p is large relative to n, selecting the value $M \ll p$ can significantly reduce the variance of the fitted coefficients. And the claim is true only when we are fitting a $m = p$ PCR, that way, our constrain of the β_p is no longer in effect and we would end up doing least squares solution.

2.

(a).

- Simulate an $n = 50 \times p = 2$ data matrix. And we shift the mean of the first 25 observations relative to the next 25 observations.

```
set.seed(435)

x <- matrix(rnorm(50 * 2), ncol = 2)

x[1: 25, 1] <- x[1:25, 1] + 3
x[1: 25, 2] <- x[1:25, 2] - 4

left <- 0
right <- 0

for (i in 1:25){
  for (j in 1:2){
    left <- left + mean((x[i, j] - x[1:25, j])^2)
    right <- right + (x[i, j] - mean(x[1:25, j]))^2
  }
}


kable(data.frame("Left" = left, "Right" = right*2), caption = "Cluster 1")
```

Table 1: Cluster 1

Left	Right
80.13463	80.13463

As we can see from above, left-hand side of (12.18) is equal to the right-hand side of (12.18). Hence, we have shown ***computationally*** that (12.18) holds.

(b).

Proof: If $\frac{1}{|C_k|}\sum_{i,i'\in C_k}\sum_{j=1}^{p}(x_{ij}-x_{i'j})^2$ holds, then

$$\begin{aligned}
\frac{1}{|C_k|}\sum_{i,i'\in C_k}\sum_{j=1}^{p}(x_{ij}-x_{i'j})^2 &= \frac{1}{|C_k|}\sum_{i,i'\in C_k}\sum_{j=1}^{p}(x_{ij}-\bar{x}_{kj}+\bar{x}_{kj}-x_{i'j})^2 \\
&= \frac{1}{|C_k|}\sum_{i,i'\in C_k}\sum_{j=1}^{p}((x_{ij}-\bar{x}_{kj})-(x_{i'j}-\bar{x}_{kj}))^2 \\
&= \frac{1}{|C_k|}\sum_{i,i'\in C_k}\sum_{j=1}^{p}((x_{ij}-\bar{x}_{kj})^2-2(x_{ij}-\bar{x}_{kj})(x_{i'j}-\bar{x}_{kj})+(x_{i'j}-\bar{x}_{kj})^2) \\
&= \frac{|C_k|}{|C_k|}\sum_{i\in C_k}\sum_{j=1}^{p}(x_{ij}-\bar{x}_{kj})^2-\frac{2}{|C_k|}\sum_{i,i'\in C_k}\sum_{j=1}^{p}(x_{ij}-\bar{x}_{kj})(x_{i'j}-\bar{x}_{kj})+\frac{|C_k|}{|C_k|}\sum_{i'\in C_k}\sum_{j=1}^{p}(x_{i'j}-\bar{x}_{kj})^2 \\
&= \sum_{i\in C_k}\sum_{j=1}^{p}(x_{ij}-\bar{x}_{kj})^2-0+\sum_{i'\in C_k}\sum_{j=1}^{p}(x_{i'j}-\bar{x}_{kj})^2 \\
&= 2\sum_{i\in C_k}\sum_{j=1}^{p}(x_{ij}-\bar{x}_{kj})^2
\end{aligned}$$

□

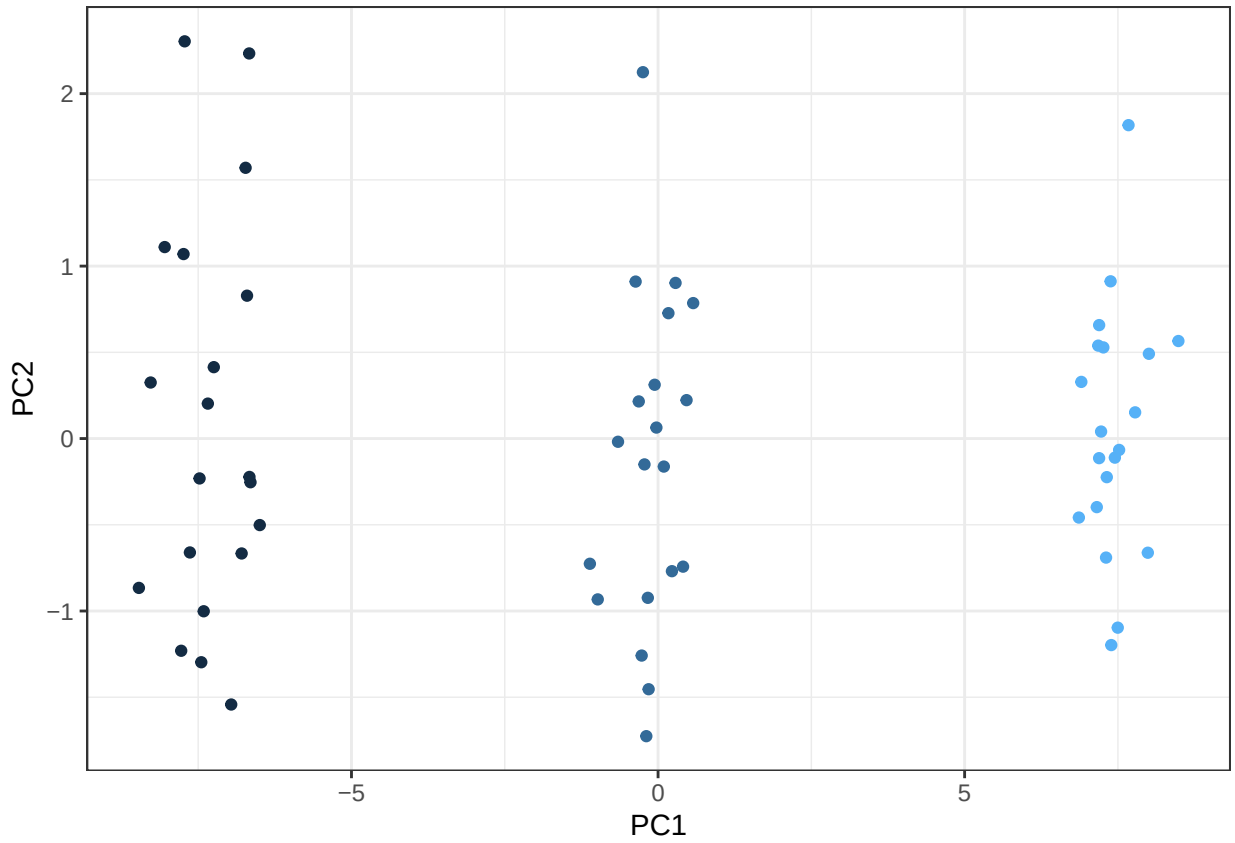
3.

(a).

```
set.seed(435)

obs_1 <- data.frame(replicate(50, rnorm(20, mean = 0, sd = 0.5))) %>%
  mutate(id = 1)
obs_2 <- data.frame(replicate(50, rnorm(20, mean = 1, sd = 0.5))) %>%
  mutate(id = 2)
obs_3 <- data.frame(replicate(50, rnorm(20, mean = 2, sd = 0.5))) %>%
  mutate(id = 3)

data <- rbind(obs_1, obs_2, obs_3)
```

(b).

```
pr.out <- prcomp(data %>% select(-id), scale = TRUE)


ggplot(data.frame(PC1 = pr.out$x[, 1], PC2 = pr.out$x[, 2], id = data$id), aes(x= PC1, y = PC2, col = i
  geom_point() +
  theme_bw() +
  theme(legend.position = "none")
```



(c).

```
km.out <- kmeans(data, 3, nstart = 20)
table(predict = km.out$cluster, true = data$id)
```

```
##         true
## predict  1  2  3
##       1 20  0  0
##       2  0  0 20
##       3  0 20  0
```

After performing K-means with K = 3, all of the data are perfectly clustered.

(d).

```
km.out <- kmeans(data, 2, nstart = 20)
table(predict = km.out$cluster, true = data$id)
```

```
##         true
## predict  1  2  3
##       1  0  0 20
##       2 20 20  0
```

After performing K-means with K = 2, all of the data are clustered into 2 clusters.

(e).

```
km.out <- kmeans(data, 4, nstart = 20)
table(predict = km.out$cluster, true = data$id)
```

```
##         true
## predict  1  2  3
##       1 20  0  0
##       2  0  0 11
##       3  0 20  0
##       4  0  0  9
```

This time, with K = 4, our cluster three got split into 2 clusters.

(f).

```
pr.out.2 <- kmeans(pr.out$x[,1:2], 3, nstart = 20)
table(predict = pr.out.2$cluster, true = data$id)
```

```
##         true
## predict  1  2  3
##       1  0 20  0
##       2 20  0  0
##       3  0  0 20
```

We essentially getting the same way of clustering these into 3 groups, hence the PCA carries enough information.

(g).

```
pr.out.3 <- kmeans(scale(data), 3, nstart = 20)
table(predict = pr.out.3$cluster, true = data$id)
```

```
##         true
## predict  1  2  3
##       1 20  0  0
##       2  0 20  0
##       3  0  0 20
```

It's the same as the results obtained in (b), and scaling does not change the results.

4.

(a).

```
data(OJ)

set.seed(435)

train <- sample(nrow(OJ), 800)
test <- -train

OJ_train <- OJ[train,]
OJ_test <- OJ[-train,]
```

(b).

```
svm.fit <- svm(Purchase ~., data = OJ_train, kernel = "linear",
               cost = 0.01, scale = FALSE)

summary(svm.fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ_train, kernel = "linear", cost = 0.01,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  628
##
##  ( 316 312 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

We see from the result that there were 628 suppoort vectors, 316 in one class and 312 in the other.

(c).

```
table(predict = predict(svm.fit,OJ_train), truth = OJ_train$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 469 203
##      MM  19 109
```

Thus, with this confusion matrix, we can calculate the Training Error Rate is

$$(203 + 19)/800 = 0.2775$$

```
table(predict = predict(svm.fit,OJ_test), truth = OJ_test$Purchase)
```

```
##         truth
## predict  CH  MM
##      CH 158  66
##      MM   7  39
```

With the above Confusion Matrix, we calculated the Testing Error Rate is

$$(66 + 7)/270 = 0.2703704$$

(d).

With the help of `tune()` function, we can perform ten-Fold Cross-Validation to help us select the optimal value for `cost`.

```
set.seed(435)
tune.out <- tune(svm, Purchase ~., data = OJ_train, kernel = "linear", ranges = list(cost = c(0.01, 0.1
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.16875
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.17750 0.04401704
## 2  0.10 0.16875 0.04050463
## 3  1.00 0.17625 0.03557562
## 4  5.00 0.17500 0.04166667
## 5 10.00 0.17625 0.04730589
```

We see that `cost = 0.1` results in the lowest Cross-Validation Error Rate.

(e).

```
bestmod <- tune.out$best.model

table(predict = predict(bestmod,OJ_train), truth = OJ_train$Purchase)
```

```
##         truth
## predict  CH  MM
##      CH 429  74
##      MM  59 238
```

With `cost = 0.1`, our new Training Error Rate is

$$(74 + 59)/800 = 0.16625$$

```
table(predict = predict(bestmod,OJ_test), truth = OJ_test$Purchase)
```

```
##         truth
## predict  CH  MM
##      CH 146  26
##      MM  19  79
```

With `cost = 0.1`, our new Testing Error Rate is

$$(26 + 19)/270 = 0.1666667$$

Which, both Training and Testing Error Rate decreased from the previous `cost = 0.01`.

(f).

```
# Default Gamma = 1
svm.radial.fit <- svm(Purchase ~., data = OJ_train, gamma = 1, kernel = "radial", cost = 0.01)
summary(svm.radial.fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ_train, gamma = 1, kernel = "radial",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##
## Number of Support Vectors:  654
##
##  ( 342 312 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

We see from the result that there were 654 support vectors, 342 in one class and 312 in the other.

```
table(predict = predict(svm.radial.fit,OJ_train), truth = OJ_train$Purchase)
```

```
##         truth
## predict  CH  MM
##      CH 488 312
##      MM   0   0
```

Thus, with this confusion matrix, we can calculate the Training Error Rate is

$$312/800 = 0.39$$

```
table(predict = predict(svm.radial.fit,OJ_test), truth = OJ_test$Purchase)
```

```
##         truth
## predict  CH  MM
##      CH 165 105
##      MM   0   0
```

With the above Confusion Matrix, we calculated the Testing Error Rate is

$$105/270 = 0.3888889$$

With the help of `tune()` function, we can perform ten-Fold Cross-Validation to help us select the optimal value for `cost`.

```
set.seed(435)
tune.out <- tune(svm, Purchase ~., data = OJ_train,
                 kernel = "radial",
                 ranges = list(cost = c(0.01, 0.1, 1, 5, 10)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.175
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.39000 0.05329426
## 2  0.10 0.19500 0.05407043
## 3  1.00 0.18125 0.04938862
## 4  5.00 0.17750 0.04440971
## 5 10.00 0.17500 0.04639804
```

We see that `cost = 10` results in the lowest Cross-Validation Error Rate.

```
bestmod <- tune.out$best.model

table(predict = predict(bestmod,OJ_train), truth = OJ_train$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 451  77
##      MM  37 235
```

With `cost = 10`, our new Training Error Rate is

$$(77 + 37)/800 = 0.1425$$

```
table(predict = predict(bestmod,OJ_test), truth = OJ_test$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 150  33
##      MM  15  72
```

With `cost = 10`, our new Testing Error Rate is

$$(33 + 15)/270 = 0.1777778$$

Which, both Training and Testing Error Rate decreased from the previous `cost = 0.01`.

(g).

```
# Set degree = 2
svm.poly.fit <- svm(Purchase ~., data = OJ_train, degree = 2, kernel = "polynomial", cost = 0.01)
summary(svm.poly.fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ_train, degree = 2, kernel = "polynomial",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##      coef.0:  0
##
```

```
## Number of Support Vectors:  632
##
##  ( 320 312 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

We see from the result that there were 632 support vectors, 320 in one class and 312 in the other.

```
table(predict = predict(svm.poly.fit,OJ_train), truth = OJ_train$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 486 295
##      MM   2  17
```

Thus, with this confusion matrix, we can calculate the Training Error Rate is

$$(295 + 2)/800 = 0.37125$$

```
table(predict = predict(svm.poly.fit,OJ_test), truth = OJ_test$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 165  97
##      MM   0   8
```

With the above Confusion Matrix, we calculated the Testing Error Rate is

$$97/270 = 0.3592593$$

With the help of `tune()` function, we can perform ten-Fold Cross-Validation to help us select the optimal value for `cost`.

```
set.seed(435)
tune.out <- tune(svm, Purchase ~., data = OJ_train,
                 kernel = "polynomial",
                 ranges = list(cost = c(0.01, 0.1, 1, 5, 10)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
```

```
##
## - best parameters:
##  cost
##     5
##
## - best performance: 0.18875
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.36875 0.05781015
## 2  0.10 0.30875 0.05104804
## 3  1.00 0.19625 0.05864500
## 4  5.00 0.18875 0.05252314
## 5 10.00 0.19000 0.04594683
```

We see that `cost = 5` results in the lowest Cross-Validation Error Rate.

```
bestmod <- tune.out$best.model

table(predict = predict(bestmod,OJ_train), truth = OJ_train$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 449  82
##      MM  39 230
```

With `cost = 5`, our new Training Error Rate is

$$(82 + 35)/800 = 0.14625$$

```
table(predict = predict(bestmod,OJ_test), truth = OJ_test$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 150  34
##      MM  15  71
```

With `cost = 10`, our new Testing Error Rate is

$$(34 + 15)/270 = 0.1814815$$

Which, both Training and Testing Error Rate decreased from the previous `cost = 0.01`.

(h).

```
kable(tibble(" " = c("Kernel","gamma","degree","Cost", "Test Error Rate"),
                  "SVC" = c("NA","NA","NA", 0.1, 0.1666667),
                  "SVM" = c("radial",1,"NA", 10, 0.1777778),
                  "SVM " = c("polynomial","NA",2, 5, 0.1814815)))
```

	SVC	SVM	SVM
Kernel	NA	radial	polynomial
gamma	NA	1	NA
degree	NA	NA	2
Cost	0.1	10	5
Test Error Rate	0.1666667	0.1777778	0.1814815

As we can see from above, the Support Vector Classifier performs best which has the lowest Test Error Rate on this data.