HW5_435

1. $z_{im} = \phi_{1m} x_{i1} + \phi_{2m} x_{i2} + \dots + \phi_{pm} x_{ip}$ (1)

a) $y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip} + \epsilon_i$

$y_i = \beta_0 + \beta_1 z_{i1} + \beta_2 z_{i2} + \dots + \beta_M z_{iM} + \epsilon_i$

b) $y_i = \beta_0 + \beta_1(\phi_{11} x_{i1} + \phi_{21} x_{i2} + \dots + \phi_{p1} x_{ip}) + \beta_2(\phi_{12} x_{i1} + \phi_{22} x_{i2} + \dots \phi_{p2} x_{ip}) + \dots + \beta_M(\phi_{1M} x_{i1} + \phi_{2M} x_{i2} + \dots \phi_{pM} x_{ip}) + \epsilon_i$

c) $y_i = \beta_0 + (\beta_1 \phi_{11} + \beta_2 \phi_{12} + \dots + \beta_M \phi_{1M}) x_{i1} + (\beta_1 \phi_{21} + \beta_2 \phi_{22} + \dots + \beta_M \phi_{2M}) x_{i2} + \dots + (\beta_1 \phi_{p1} + \beta_2 \phi_{p2} + \dots + \beta_M \phi_{pM}) x_{ip}$

↳ this means that the principal components regression model is linear in the colums of X because all of the coefs for $x_{i1}, \dots, x_{ip}$ are constants

d) No, it's False. This is because we have too many restriction on the principle components like mimizing the variance

```r
# Part 2a
set.seed(435)
x <- matrix(rnorm(50 * 2), ncol = 2)
cluster1 <- x[1:25,]
cluster2 <- x[26:50,]

sum1 <- 0
sum2 <- 0
sum3 <- 0
sum4 <- 0
for (i in 1:25){
  for (j in 1:25){
    sum1 = sum1 + (cluster1[i,1] - cluster1[j,1])^2
    sum2 = sum2 + (cluster1[i,2] - cluster1[j,2])^2
    sum3 = sum3 + (cluster2[i,1] - cluster2[j,1])^2
    sum4 = sum4 + (cluster2[i,2] - cluster2[j,2])^2
  }
}
```

```r
sum_cluster1 <- (sum1 + sum2)/25
sum_cluster2 <- (sum3 + sum4)/25
(result1 <- c(sum_cluster1, sum_cluster2))
```

```
## [1] 80.13463 85.32348
```

```r
diff1 <- scale(cluster1, center=TRUE, scale=FALSE)
right1 <- 2*sum(diff1^2)

diff2 <- scale(cluster2, center=TRUE, scale=FALSE)
right2 <- 2*sum(diff2^2)

(result2 <- c(right1, right2))
```

```
## [1] 80.13463 85.32348
```

We can see that the result on the left hand side is equal to the result on the right hand side. The result is 80.13463 for the first cluster and 85.32348 for the second cluster.

2b. Prove $\frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 = 2 \sum_{i \in C_k} \sum_{j=1}^{p} (x_{ij} - \bar{x}_{kj})^2$

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

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

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

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

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

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

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

$$2 \sum_{i \in C_k} \sum_{j=1}^{p} (x_{ij} - \bar{x}_{kj})^2 - 0 =$$

$$2 \sum_{i \in C_k} \sum_{j=1}^{p} (x_{ij} - \bar{x}_{kj})^2$$

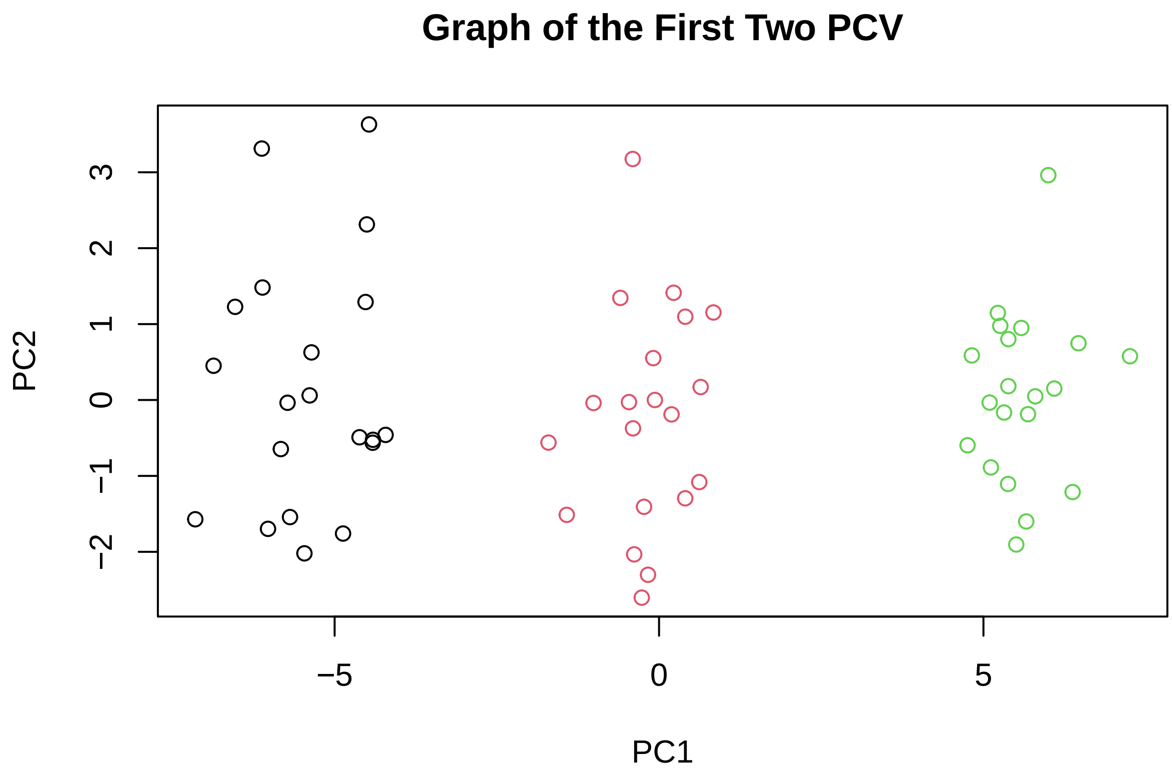
// equals to RHS.

∴ Proven that

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

```r
# Part 3a
set.seed(435)
x <- rbind(matrix(rnorm(20*50, mean=0), nrow = 20),
           matrix(rnorm(20*50, mean=1), nrow = 20),
           matrix(rnorm(20*50, mean=2), nrow = 20))
```

```r
# Part 3b
pr.out <- prcomp(x, scale = TRUE)$x
plot(pr.out[,1:2], col=c(rep(1,20), rep(2,20), rep(3,20)),
     main="Graph of the First Two PCV")
```




```
# Part 3c
set.seed(435)
class <- c(rep(1,20), rep(2,20), rep(3,20))
table(kmeans(x, centers = 3)$cluster, class)
```

```
##    class
##      1  2  3
##   1  0 20  0
##   2  0  0 20
##   3 20  0  0
```

They are well clustered and pretty accurate compared to the true class labels.

```
# Part 3d
set.seed(435)
table(kmeans(x, centers = 2)$cluster, class)
```

```
##    class
##      1  2  3
##   1 20 20  0
##   2  0  0 20
```

Some are forced to a wrong class since it combined two classes, which increases the inaccuracy.

```r
# Part 3e
set.seed(435)
table(kmeans(x, centers = 4)$cluster, class)
```

```
##    class
##      1  2  3
##   1  0 20  0
##   2  0  0 11
##   3 20  0  0
##   4  0  0  9
```

As we can see, one of the three classes are split into two, which increases the inaccuracy.

```r
# Part 3f
set.seed(435)
table(kmeans(pr.out[,1:2], centers = 3)$cluster, class)
```

```
##    class
##      1  2  3
##   1  0 20  0
##   2  0  0 20
##   3 20  0  0
```

We get the same results as 3b, in which they are well clustered and correctly classifies all three clusters.

```r
# Part 3g
set.seed(435)
table(kmeans(scale(x), centers = 3)$cluster, class)
```

```
##    class
##      1  2  3
##   1  0 20  0
##   2  0  0 20
##   3 20  0  0
```

We can see that we get the same results as 3b, in which they are well clustered. By scaling down the standard deviation to 1, we can decrease the overlap between the classes.

```r
# Part 4a
set.seed(435)
sample <- sample(nrow(OJ), 800)
train <- OJ[sample,]
test <- OJ[-sample,]
```

```r
# Part 4b
svmfit <- svm(Purchase ~ ., data = train, kernel = "linear", cost = 0.01)
summary(svmfit)
```

```
##
## Call:
```

```
## svm(formula = Purchase ~ ., data = train, kernel = "linear", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  446
##
##  ( 224 222 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
# Part 4c
ypred <- predict(svmfit, train)
(training_error <- mean(train$Purchase != ypred))
```

```
## [1] 0.1675
```

```
ypred_test <- predict(svmfit, test)
(test_error <- mean(test$Purchase != ypred_test))
```

```
## [1] 0.1666667
```

The training error is 0.1675, and the test error is 0.1666667.

```
# Part 4d
set.seed(435)
tune.out <- tune(svm, Purchase ~., data = train, kernel = "linear",
                ranges = list(cost = c( 0.01, 0.05, 0.1, 1, 5, 10)))
tune.out
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.16875
```

The optimal cost we select is 0.1.

```r
# Part 4e
svmfit1 <- svm(Purchase ~ ., data = train, kernel = "linear", cost = 0.1)

ypred <- predict(svmfit1, train)
(training_error <- mean(train$Purchase != ypred))
```

```
## [1] 0.16625
```

```r
ypred_test <- predict(svmfit1, test)
(test_error <- mean(test$Purchase != ypred_test))
```

```
## [1] 0.1666667
```

The training error is 0.16625, and the test error is 0.1666667.

```r
# Part 4f
set.seed(435)
svm_radial <- svm(Purchase ~ ., data = train, method = "radial", cost = 0.01)
summary(svm_radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, method = "radial", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##
## Number of Support Vectors:  627
##
##  ( 315 312 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
ypred <- predict(svm_radial, train)
(training_error <- mean(train$Purchase != ypred))
```

```
## [1] 0.39
```

```r
ypred_test <- predict(svm_radial, test)
(test_error <- mean(test$Purchase != ypred_test))
```

```
## [1] 0.3888889
```

```r
tune.out <- tune(svm, Purchase ~., data = train, kernel = "radial",
                 ranges = list(cost = c(0.01, 0.1, 1, 5, 10)))
tune.out
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.175
```

```r
# Using cost 10
svmfit2 <- svm(Purchase ~ ., data = train, kernel = "radial", cost = 10)

ypred <- predict(svmfit2, train)
(training_error <- mean(train$Purchase != ypred))
```

```
## [1] 0.1425
```

```r
ypred_test <- predict(svmfit2, test)
(test_error <- mean(test$Purchase != ypred_test))
```

```
## [1] 0.1777778
```

The training error with the cost 0.01 is 0.39, and the test error with the cost 0.01 is 0.3888889. On the other hand, the training error with the optimal cost 10 is 0.1425, and the test error with the optimal cost 10 is 0.1777778.

```r
# Part 4g
set.seed(435)

svmfit3 <- svm(Purchase ~ ., data = train, kernel = "polynomial", degree = 2,
               cost = 0.01)
summary(svmfit3)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "polynomial",
##     degree = 2, 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
```

```
ypred <- predict(svmfit3, train)
(training_error <- mean(train$Purchase != ypred))
```

```
## [1] 0.37125
```

```
ypred_test <- predict(svmfit3, test)
(test_error <- mean(test$Purchase != ypred_test))
```

```
## [1] 0.3592593
```

```
tune.out <- tune(svm, Purchase ~ ., data = train, kernel = "polynomial",
                 degree = 2, ranges = list(cost = c(0.01, 0.1, .5, 1, 5, 10)))
tune.out
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.17875
```

```
# Using cost 10
svmfit4 <- svm(Purchase ~ ., data = train, kernel = "polynomial", degree = 2,
               cost = 10)

ypred <- predict(svmfit4, train)
(training_error <- mean(train$Purchase != ypred))
```

```
## [1] 0.15
```

```
ypred_test <- predict(svmfit4, test)
(test_error <- mean(test$Purchase != ypred_test))
```

```
## [1] 0.162963
```

The training error with the cost 0.01 is 0.37125, and the test error with the cost 0.01 is 0.3592593. On the other hand, the training error with the optimal cost 10 is 0.15, and the test error with the optimal cost 10 is 0.162963.

4h. I believe that the polynomial degree 2 kernel gives us the best results because it has a relatively low training error and it has the lowest test error.