Stat435_HW5

Su Huang

2022-06-04

Problem 1

a).

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

b).

$z_{im} = \phi_{1m}x_{i1}+\phi_{2m}x_{i2}+...+\phi_{pm}x_{ip}$Plug this equation to part a:$y_i = \beta_0+\beta_1(\phi_{11}x_{i1}+\phi_{21}x_{i2}+...+\phi_{p1}x_{ip})+\beta_2(\phi_{12}x_{i1}+\phi_{22}x_{i2}+$

c).

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

4). It's False. Since we only use the first M PC instead of the whole columns of X, if we choose the better(more related) PC, which may give us more accurate prediction than using the columns of X.

Problem 2

```
matrix = array(rnorm(320), dim=c(20,16))

matrix[1:10, 16] <- 1
matrix[11:20, 16] <- 2
#matrix
```

```
# cluster 1
# left hand side:
sum = 0
for (i in 1:10) {
  for (i2 in 1:10) {
    for (j in 1:15) {
      sum = sum + (matrix[i, j] - matrix[i2, j])^2
    }
  }
}
c1_left <- sum/10

# right hand side:
sum2 <- 0
for (i in 1:10) {
  for (j in 1:15) {
      sum2 = sum2 + (matrix[i, j] - mean(matrix[1:10, j]))^2
  }
```

```
}
c1_right <- sum2 * 2

c1_right
```

a).

```
## [1] 322.9701
```

```
c1_left
```

```
## [1] 322.9701
```

```
c1_right - c1_left
```

```
## [1] 2.273737e-13
```

```
# cluster 2
# left hand side:
sum = 0
for (i in 11:20) {
  for (i2 in 11:20) {
    for (j in 1:15) {
      sum = sum + (matrix[i, j] - matrix[i2, j])^2
    }
  }
}
c1_left <- sum/10

# right hand side:
sum2 <- 0
for (i in 11:20) {
  for (j in 1:15) {
      sum2 = sum2 + (matrix[i, j] - mean(matrix[11:20, j]))^2
  }
}
c1_right <- sum2 * 2

c1_right
```

```
## [1] 225.1242
```

```
c1_left
```

```
## [1] 225.1242
```

```
c1_right - c1_left
```

```
## [1] 3.126388e-13
```

The left side is equal to the right side for both clusters.

b).

Problem 3

```
set.seed(1)
df <- data.frame(replicate(50, rnorm(20, mean = 1,  sd = 1))) %>%
```

```
    rbind(data.frame(replicate(50, rnorm(20, mean = 2, sd = 1)))) %>%
    rbind(data.frame(replicate(50, rnorm(20, mean = 3, sd = 1)))) %>%
    as.tibble %>%
    mutate(id = row_number(),
           class = ifelse(id <= 20, 'a',
                          ifelse(id <= 40, 'b',
                                 'c'))) %>%
    select(-id)
```

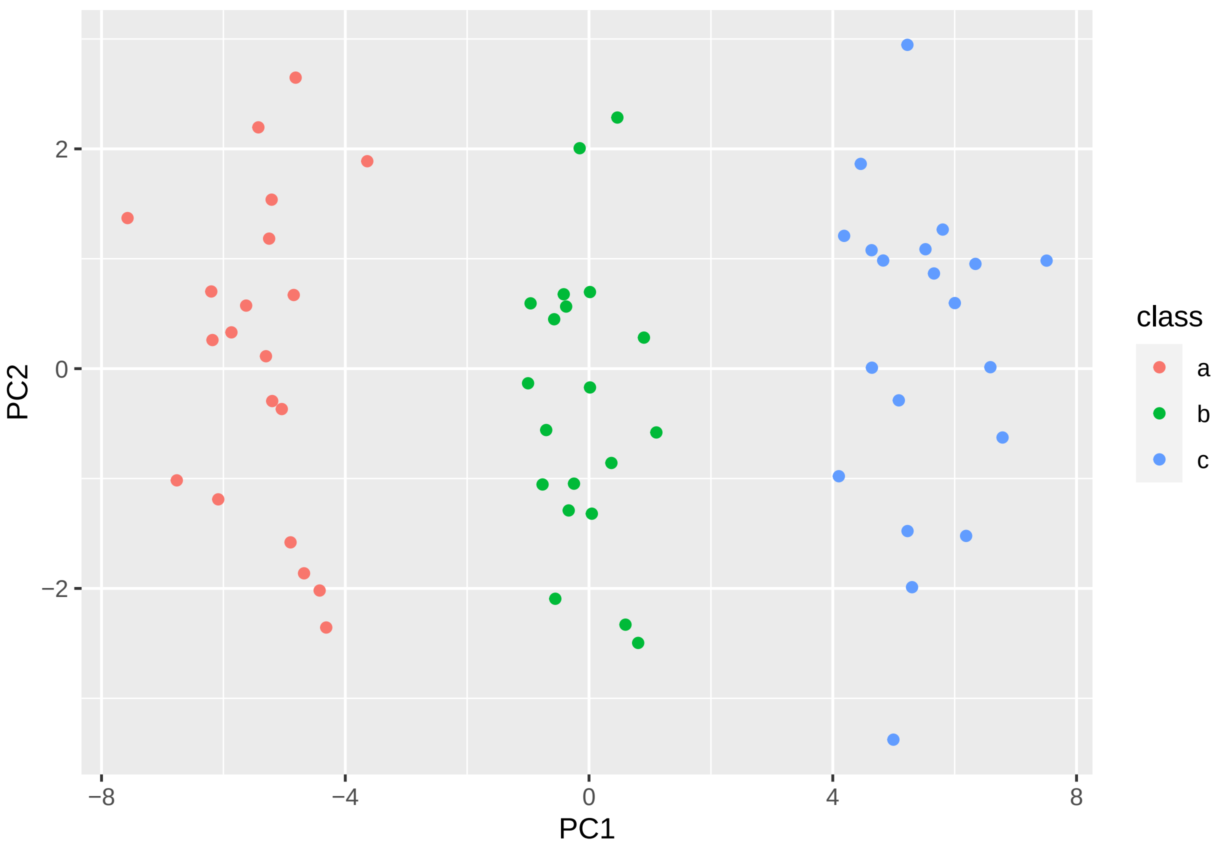
a).

```
## Warning: `as.tibble()` was deprecated in tibble 2.0.0.
## Please use `as_tibble()` instead.
## The signature and semantics have changed, see `?as_tibble`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

```
df
```

```
## # A tibble: 60 x 51
##       X1     X2    X3     X4     X5     X6    X7     X8     X9     X10    X11
##    <dbl>  <dbl> <dbl>  <dbl>  <dbl>  <dbl> <dbl>  <dbl>  <dbl>   <dbl>  <dbl>
##  1 0.374  1.92  0.835  3.40   0.431 0.380  0.494 -0.914  1.43  -0.231   1.41
##  2 1.18   1.78  0.747  0.961  0.865 1.04   2.34   2.18   0.761  1.98    2.69
##  3 0.164  1.07  1.70   1.69   2.18  0.0891 0.785 -0.665  2.06   1.22    2.59
##  4 2.60  -0.989 1.56   1.03  -0.524 1.16   0.820  0.536  1.89  -0.467   0.669
##  5 1.33   1.62  0.311  0.257  1.59  0.345  0.900 -0.116  0.381  1.52   -1.29
##  6 0.180  0.944 0.293  1.19   1.33  2.77   1.71   0.249  3.21   0.841   3.50
##  7 1.49   0.844 1.36  -0.805  2.06  1.72   0.926  3.09   0.745  2.46    1.67
##  8 1.74  -0.471 1.77   2.47   0.696 1.91   0.962  1.02  -0.424  0.234   1.54
##  9 1.58   0.522 0.888  1.15   1.37  1.38   0.318 -0.286  0.856  0.570   0.987
## 10 0.695  1.42  1.88   3.17   1.27  2.68   0.676 -0.641  1.21   0.0739  1.51
## # ... with 50 more rows, and 40 more variables: X12 <dbl>, X13 <dbl>,
## #   X14 <dbl>, X15 <dbl>, X16 <dbl>, X17 <dbl>, X18 <dbl>, X19 <dbl>,
## #   X20 <dbl>, X21 <dbl>, X22 <dbl>, X23 <dbl>, X24 <dbl>, X25 <dbl>,
## #   X26 <dbl>, X27 <dbl>, X28 <dbl>, X29 <dbl>, X30 <dbl>, X31 <dbl>,
## #   X32 <dbl>, X33 <dbl>, X34 <dbl>, X35 <dbl>, X36 <dbl>, X37 <dbl>,
## #   X38 <dbl>, X39 <dbl>, X40 <dbl>, X41 <dbl>, X42 <dbl>, X43 <dbl>,
## #   X44 <dbl>, X45 <dbl>, X46 <dbl>, X47 <dbl>, X48 <dbl>, X49 <dbl>, ...
```

```
pr.out <- prcomp(df %>% select(-class), scale = TRUE)
ggplot(data.frame(PC1 = pr.out$x[,1], PC2 = pr.out$x[,2], class = df$class),
       aes(x = PC1, y = PC2, col = class)) + geom_point()
```



b).

```
km.out <- kmeans(df %>% select(-class), 3, nstart = 20)
table(df$class, km.out$cluster)
```

c).

```
##
##      1  2  3
##   a  0  0 20
##   b 20  0  0
##   c  0 20  0
```

The k-means clusters perform perfect on the observations.

```
km.out2 <- kmeans(df %>% select(-class), 2, nstart = 20)
table(df$class, km.out2$cluster)
```

d).

```
##
##      1  2
##   a  0 20
##   b  0 20
##   c 20  0
```

The observations a and b are included into one cluster (cluster 2)

```
km.out4 <- kmeans(df %>% select(-class), 4, nstart = 20)
table(df$class, km.out4$cluster)
```

e).

```
##
##      1  2  3  4
##   a  9 11  0  0
##   b  0  0  0 20
##   c  0  0 20  0
```

The observation a is separated into the cluster 1 and 2.

```
km.outpca <- kmeans(pr.out$x[,1:2], 3, nstart = 20)
table(df$class, km.outpca$cluster)
```

f).

```
##
##      1  2  3
##   a  0 20  0
##   b 20  0  0
##   c  0  0 20
```

The k-means clusters perform perfect on the observations.

```
km.outscale <- kmeans(scale(df %>% select(-class)), 3, nstart = 20)
table(df$class, km.outscale$cluster)
```

g).

```
##
##      1  2  3
##   a  0  0 20
##   b 20  0  0
##   c  0 20  0
```

It performs as perfect as part c

Problem 4

```
library(ISLR2)
library(e1071)
#head(OJ)
dim(OJ)
```

```
## [1] 1070   18
```

```
set.seed(1)
is.train <- sample(dim(OJ)[1],800)
OJ.train <- OJ[is.train, ]
OJ.test <- OJ[-is.train, ]
```

a).

```
svmfit <- svm(Purchase ~ ., data = OJ.train, kernel = "linear", cost = 0.01, scale = FALSE)
summary(svmfit)
```

b).

```
##
## 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:  615
##
##  ( 309 306 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The number of support vectors is 615 which is a considerable number since we only have 800 data in training set. The number of classes is 2 with level of CH and MM.

```
pred_train <- predict(svmfit, OJ.train)
table(predict = pred_train, truth = OJ.train$Purchase)
```

c).

```
##        truth
## predict  CH  MM
##      CH 420 105
##      MM  65 210
```

```
print(paste("The training error for train is ", (65+105) /800))
```

```
## [1] "The training error for train is  0.2125"
```

```
pred_test <- predict(svmfit, OJ.test)
table(predict = pred_test, truth = OJ.test$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 148  43
##      MM  20  59
```

```
print(paste("The test error is ", (20+43) / 270))
```

```
## [1] "The test error is  0.233333333333333"
```

```r
tune.out.linear <- tune(svm, Purchase ~., data = OJ.train, kernel = "linear",
                 ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10)))
summary(tune.out.linear)
```

d).

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.17125
##
## - Detailed performance results:
##    cost   error dispersion
## 1 1e-03 0.31500 0.05329426
## 2 1e-02 0.17375 0.03884174
## 3 1e-01 0.17875 0.03064696
## 4 1e+00 0.17500 0.03061862
## 5 5e+00 0.17250 0.03322900
## 6 1e+01 0.17125 0.03488573
```

The optimal cost is 10 with error 0.17125 and dispersion 0.03488573

```r
pred_train_e <- predict(tune.out.linear$best.model, OJ.train)
table(predict = pred_train_e, truth = OJ.train$Purchase)
```

e).

```
##        truth
## predict  CH  MM
##      CH 423  69
##      MM  62 246
```

```r
print(paste("The training error for tune with cost = 10 is ", (62 + 69) /800))
```

```
## [1] "The training error for tune with cost = 10 is  0.16375"
```

```r
pred_train_etest <- predict(tune.out.linear$best.model, OJ.test)
table(predict = pred_train_etest, truth = OJ.test$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 156  28
##      MM  12  74
```

```r
print(paste("The testing error  for tune with cost = 10 is ", (12 + 28) /270))
```

```
## [1] "The testing error  for tune with cost = 10 is  0.148148148148148"
```

```
svmrad <- svm(Purchase ~ ., data = OJ.train, kernel = "radial", cost = 0.01, scale = FALSE)
summary(svmrad)
```

f).

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "radial", cost = 0.01,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##
## Number of Support Vectors:  642
##
##  ( 327 315 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The number of support vectors is 642 (327, 315). The number of classes is 2 (CH, MM)

```
pred_train_rad <- predict(svmrad, OJ.train)
table(predict = pred_train_rad, truth = OJ.train$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 485 315
##      MM   0   0
```

```
pred_test_rad <- predict(svmrad, OJ.test)
table(predict = pred_test_rad, truth = OJ.test$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 168 102
##      MM   0   0
```

```
print(paste("The training error is ", 315 /800))
```

```
## [1] "The training error is  0.39375"
```

```
print(paste("The test error test is ", 102 / 270))
```

```
## [1] "The test error test is  0.377777777777778"
```

```
tune.out.rad <- tune(svm, Purchase ~., data = OJ.train, kernel = "radial",
                ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10)))
summary(tune.out.rad)
```

```
##
```

```
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.17625
##
## - Detailed performance results:
##    cost   error dispersion
## 1 1e-03 0.39375 0.06568284
## 2 1e-02 0.39375 0.06568284
## 3 1e-01 0.18250 0.05470883
## 4 1e+00 0.17625 0.03793727
## 5 5e+00 0.18125 0.04299952
## 6 1e+01 0.18125 0.04340139
```

The optimal cost is 1 with error = 0.17625.

```
pred_train_f <- predict(tune.out.rad$best.model, OJ.train)
table(predict = pred_train_f, truth = OJ.train$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 441  77
##      MM  44 238
```

```
pred_train_ftest <- predict(tune.out.rad$best.model, OJ.test)
table(predict = pred_train_ftest, truth = OJ.test$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 151  33
##      MM  17  69
```

```
print(paste("The training error for tune with cost = 5 is ", (44 + 77) /800))
```

```
## [1] "The training error for tune with cost = 5 is  0.15125"
```

```
print(paste("The testing error for tune with cost = 5 is ", (17 + 33) /270))
```

```
## [1] "The testing error for tune with cost = 5 is  0.185185185185185"
```

```
svmpoly <- svm(Purchase ~ ., data = OJ.train, kernel = "polynomial", cost = 0.01, scale = FALSE, degree
summary(svmpoly)
```

g

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "polynomial",
##     cost = 0.01, degree = 2, scale = FALSE)
##
##
## Parameters:
```

```
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  333
##
##  ( 166 167 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The number of support vectors is 333 (166, 167). The number of classes is 2 (CH, MM)

```r
pred_train_poly <- predict(svmpoly, OJ.train)
table(predict = pred_train_poly, truth = OJ.train$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 423  70
##      MM  62 245
```

```r
pred_test_poly <- predict(svmpoly, OJ.test)
table(predict = pred_test_poly, truth = OJ.test$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 154  29
##      MM  14  73
```

```r
print(paste("The training error for train is ", (62+70) /800))
```

```
## [1] "The training error for train is  0.165"
```

```r
print(paste("The test error for test is ", (14+29) / 270))
```

```
## [1] "The test error for test is  0.159259259259259"
```

```r
tune.out.poly <- tune(svm, Purchase ~., data = OJ.train, kernel = "polynomial",
                ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10)))
summary(tune.out.poly)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.19125
##
## - Detailed performance results:
```

```
##    cost   error dispersion
## 1 1e-03 0.39375 0.08191501
## 2 1e-02 0.37125 0.07337357
## 3 1e-01 0.29000 0.07139483
## 4 1e+00 0.19375 0.04903584
## 5 5e+00 0.19250 0.05041494
## 6 1e+01 0.19125 0.05622685
```

The optimal cost is 10 with error = 0.19125 and dispersion = 0.05204165

```r
pred_train_g <- predict(tune.out.poly$best.model, OJ.train)
table(predict = pred_train_g, truth = OJ.train$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 446  75
##      MM  39 240
```

```r
pred_train_gtest <- predict(tune.out.poly$best.model, OJ.test)
table(predict = pred_train_gtest, truth = OJ.test$Purchase)
```

```
##        truth
## predict  CH  MM
##      CH 155  42
##      MM  13  60
```

```r
print(paste("The training error for tune with cost = 10 is ", (39 + 75) /800))
```

```
## [1] "The training error for tune with cost = 10 is  0.1425"
```

```r
print(paste("The testing error for tune with cost = 10 is ", (13 + 42) /270))
```

```
## [1] "The testing error for tune with cost = 10 is  0.203703703703704"
```

h). The polynomial model with cost = 10 on tune giving training error with 0.1425 on training data is the best for training. The linear model with cost = 10 on tune giving training error with 0.1481 on testing data is the best results.