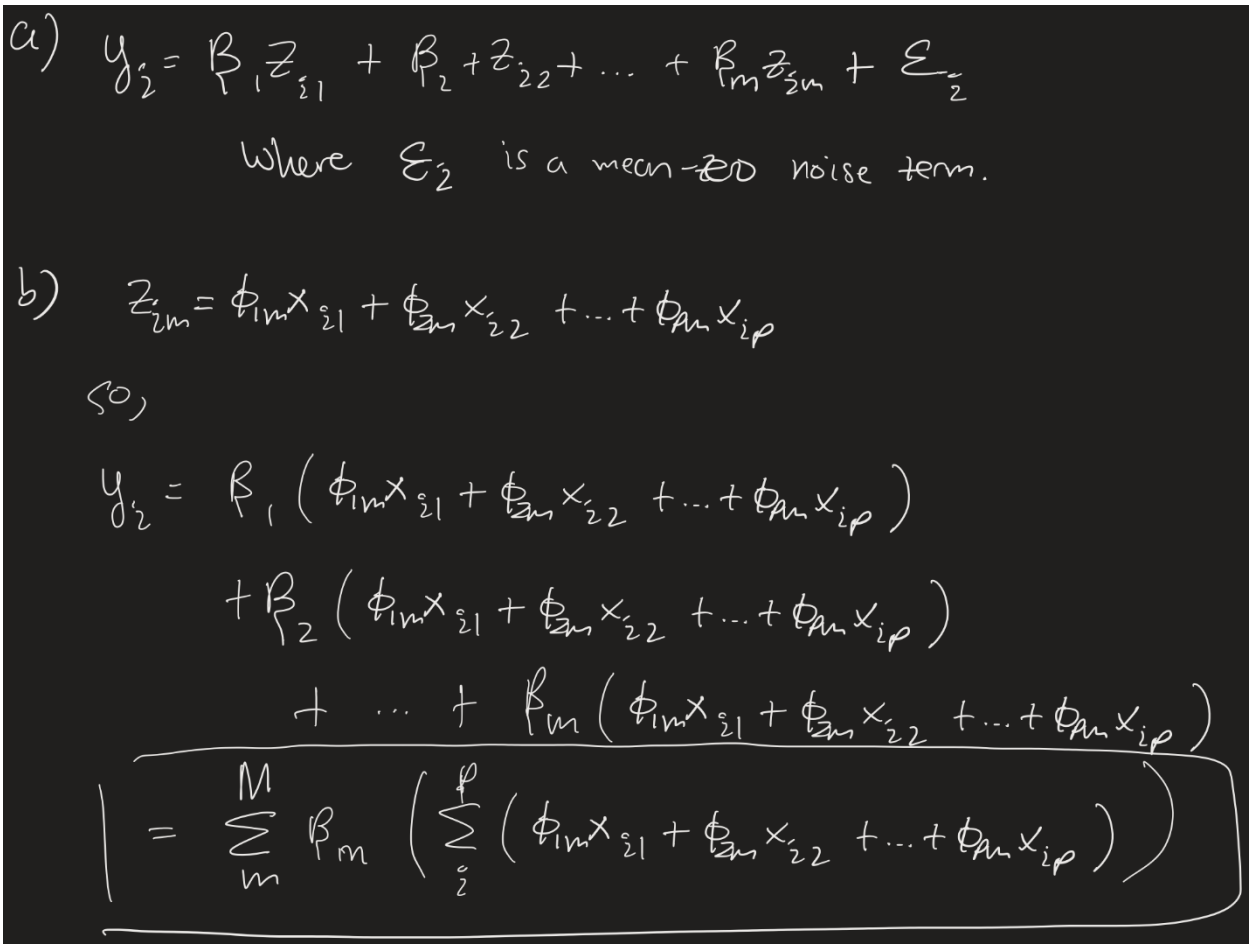
James Yang

DATA 558

Spring 2022

Homework #5

1.



c. There is a linear combination of components which are linear as well. Principal component regression is linear in columns of f(g(x)). G(x) in it of itself is linear. The summations are linear.

d. This isn't true because there the X columns doesn't represent the Beta m values. The phi values are also changing the X columns which mean that the principal components will yield a different fitted value as opposed to just a simple linear model.

2. Left side of data is just the mean value of kmeans$withinss = 1.219
Right side of data is the average distance between point and centroid squared = 1.219

```r
#Q2
library(factoextra)
nxpMatrix <- matrix(rnorm(36), nrow = 6) #nxp = 6x6
k <- 4
km.res <- kmeans(nxpMatrix, k, nstart = 36)
left <- sum(km.res$withinss) / length(km.res$withinss)

euclidDistance = 0
number_iterations = 0
for(i in 1:36) {
  for(j in 1:24) {
    number_iterations = number_iterations + 1
    euclidDistance = euclidDistance + sqrt(sum((km.res$centers[j] - nxpMatrix[i])^2))
  }

}

avg = euclidDistance / number_iterations
```

Data	
km.res	List of 9
nxpMatrix	num [1:6, 1:6] -0.592 0.337 0.777 1.445 0.626 ...
Values	
avg	1.21878478140722
dist	NULL
euclidDistance	1079.5751208405
i	36L
j	24L
k	4
left	1.21878478140722
number_iterations	864
Functions	
euc.dist	function (x1, x2)

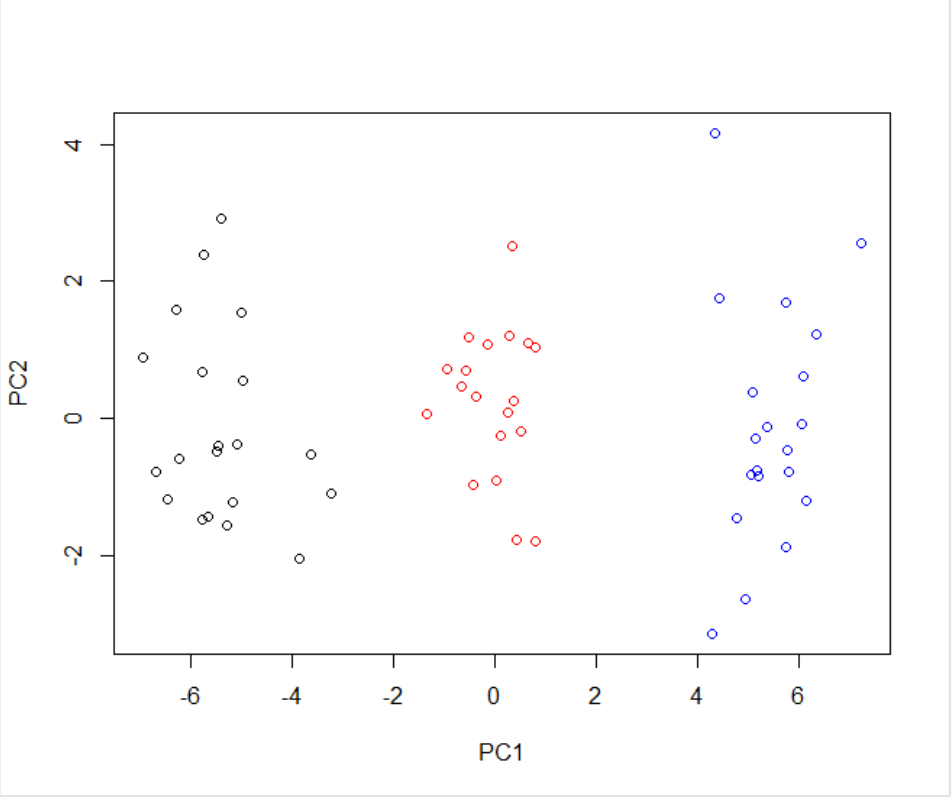
This iteration created an equal value which indeed supports the idea in the textbook that it holds equivalent.

3.

a. Producing the observations with different means so they can get spread out.

```r
#Q3
#a
x <- rbind(matrix(rnorm(1000, mean = 0), nrow = 20),
           matrix(rnorm(1000, mean = 1), nrow = 20),
           matrix(rnorm(1000, mean = 2), nrow = 20))
```

b. Performing PCA, I get the following plot with the following code. Since it is dimension reduction, we only want the $x values from the output.




```
#b (dimension reduction)
pca.x <- prcomp(x, scale = TRUE)$x
plot(pca.x[,1:2], col=c(rep("Black",20), rep("Red",20), rep("Blue",20)))
```

c. After doing K-means clustering = 3, we get all perfect labels.

```
#c
km.x <- kmeans(x, 3, nstart = 25)``
table(km.x$cluster, c(rep(1, 20), rep(2,20), rep(3,20)))
```

```
   1  2  3
1  0 20  0
2 20  0  0
3  0  0 20
```

d. After doing K = 2, we still get a perfect classification.

```
#d
km.x2 <- kmeans(x, 2, nstart = 25)
table(km.x2$cluster, c(rep(1, 20), rep(2,20), rep(3,20)))
```

```
   1  2  3
1  0  0 20
2 20 20  0
```

e. After doing K = 4, it appears the 3rd column gets a split classification between 3 and 4.

```
#e
km.x3 <- kmeans(x, 4, nstart = 25)
table(km.x3$cluster, c(rep(1, 20), rep(2,20), rep(3,20)))
```

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

f. After doing K = 3 with the PCA, it appears to perfectly classify.

```
#f
km.x <- kmeans(pca.x[,1:2], 3, nstart = 25)
table(km.x$cluster, c(rep(1, 20), rep(2,20), rep(3,20)))
```

```
   1  2  3
1  0  0 20
2  0 20  0
3 20  0  0
```

g. After doing the scale(x), it appears to also perfectly classify. The results appear to be the exact same as the ones found in part c except the classification flipped with column 2 and 3 finding 3 and 1 respectively. This could be due to randomization but standardizing the values because of their mean may alter their classification, but it does it on each set of values so they might just classify differently but still as a cluster.

```
#g
km.scale <- kmeans(scale(x), 3, nstart = 25)
table(km.scale$cluster, c(rep(1, 20), rep(2,20), rep(3,20)))
```

```
   1  2  3
1  0  0 20
2 20  0  0
3  0 20  0
```

4. .

a. The following code creates a training set of 800 observations in OJ.

```
#4a
library(ISLR2)
train <- sample(1:nrow(OJ), 800)
model_4a.train <- head(OJ, 800)
model_4a.test <- OJ[-train, ]
```

b. After calling svm, we get the following results.

```
#4b
library(e1071)
model_4b <- svm(Purchase ~ ., cost= 0.01, data = model_4a.train, kernel = "linear")
summary(model_4b)
```

```
Call:
svm(formula = Purchase ~ ., data = OJ, cost = 0.01, kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01

Number of Support Vectors:  560

 ( 279 281 )


Number of Classes:  2

Levels:
 CH MM
```

It appears there were over 424 support vectors being plotted to classify this binary svm. 212 in one and 212 in the other. The reason it's binary is because the number of classes being shown appears to be 2. It appears to be a linear kernel used with cost of 0.01.

c. Testing error appears to be 17.4%.
Training error appears to be 15.5%
With this code:

```
#4c
model_4c <- predict(model_4b, newdata = model_4a.test)
predict_vals <- table(predict = model_4c, truth = model_4a.test$Purchase)
testing_error <- 1 - (sum(diag(predict_vals)) / length(model_4a.test$Purchase))

model_4c <- predict(model_4b, newdata = model_4a.train)
predict_vals <- table(predict = model_4c, truth = model_4a.train$Purchase)
training_error <- 1 - (sum(diag(predict_vals)) / length(model_4a.train$Purchase))
```

d. When running the following code,

```
#4d
tune.out <- tune(svm, Purchase ~ ., data = model_4a.train, kernel = "linear", ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10)))
summary(tune.out)
```

It produces the following:

```
> summary(tune.out)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
 0.01

- best performance: 0.155

- Detailed performance results:
   cost   error dispersion
1 1e-03 0.35000 0.09409658
2 1e-02 0.15500 0.02898755
3 1e-01 0.15625 0.03397814
4 1e+00 0.15625 0.03076005
5 5e+00 0.15750 0.03395258
6 1e+01 0.15500 0.03641962
```

Indicating that the most optimal cost is at 0.01 or 1 with an error of 15.5%.

e. When using the following code,

```
#4e
model_4b <- svm(Purchase ~ ., cost= 1, data = model_4a.train, kernel = "linear")
model_4e <- predict(model_4b, newdata = model_4a.test)
predict_vals <- table(predict = model_4e, truth = model_4a.test$Purchase)
testing_error <- 1 - (sum(diag(predict_vals)) / length(model_4a.test$Purchase))

model_4e <- predict(model_4b, newdata = model_4a.train)
predict_vals <- table(predict = model_4e, truth = model_4a.train$Purchase)
training_error <- 1 - (sum(diag(predict_vals)) / length(model_4a.train$Purchase))
```

Training error: 14.875%

Testing error: 17.407%

f. When using radial as my kernel as opposed to linear with a cost of 0.01, it classifies in two sections with 305 and 303 (CH and MM) respectively.

```
Call:
svm(formula = Purchase ~ ., data = model_4a.train, cost = 0.01, kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  0.01

Number of Support Vectors:  608

 ( 305 303 )


Number of Classes:  2

Levels:
 CH MM
```

When producing radial errors, we get that **Testing Error: 41.9% and Training Error as 38.9%**

When getting the best tune model, the best cost was 1. When implementing the Training and Testing error for that value, we get **14.9% and 15.5% respectively.** The calculated values above were all using this code:

```
#4f
model_4f <- svm(Purchase ~ ., cost= 0.01, data = model_4a.train, kernel = "radial")
summary(model_4f)

model_4f_predict <- predict(model_4f, newdata = model_4a.test)
predict_vals <- table(predict = model_4f_predict, truth = model_4a.test$Purchase)
testing_error <- 1 - (sum(diag(predict_vals)) / length(model_4a.test$Purchase))

model_4f_predict <- predict(model_4f, newdata = model_4a.train)
predict_vals <- table(predict = model_4f_predict, truth = model_4a.train$Purchase)
training_error <- 1 - (sum(diag(predict_vals)) / length(model_4a.train$Purchase))

tune.out <- tune(svm, Purchase ~ ., data = model_4a.train, kernel = "radial", ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10)))
summary(tune.out)

model_4f <- svm(Purchase ~ ., cost= 1, data = model_4a.train, kernel = "radial")
summary(model_4f)

model_4f_predict <- predict(model_4f, newdata = model_4a.test)
predict_vals <- table(predict = model_4f_predict, truth = model_4a.test$Purchase)
testing_error <- 1 - (sum(diag(predict_vals)) / length(model_4a.test$Purchase))

model_4f_predict <- predict(model_4f, newdata = model_4a.train)
predict_vals <- table(predict = model_4f_predict, truth = model_4a.train$Purchase)
training_error <- 1 - (sum(diag(predict_vals)) / length(model_4a.train$Purchase))
```

g. When using polynomial as my kernel with a degree of 2, we get that it classifies 309 in CH and 303 in MM.

```
Call:
svm(formula = Purchase ~ ., data = model_4a.train, cost = 0.01, kernel = "polynomial", degree = 2)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  0.01
     degree:  2
     coef.0:  0

Number of Support Vectors:  612

 ( 309 303 )


Number of Classes:  2

Levels:
 CH MM
```

When calculating their errors, **we get 41.9% and 37.9% for testing and training errors respectively.**

When running tune, we get that the most optimal cost for low error is 10 with 17.9% error.

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
    5

- best performance: 0.17875

- Detailed performance results:
   cost   error dispersion
1 1e-03 0.37875 0.05894029
2 1e-02 0.38000 0.06015027
3 1e-01 0.32000 0.07269609
4 1e+00 0.19750 0.05163978
5 5e+00 0.17875 0.05205833
6 1e+01 0.17875 0.04332131
```

When running with optimal cost of 10, **we get that the training and testing error are 15.4% and 13.7% respectively.**
Using this code:

```
model_4g <- svm(Purchase ~ ., cost= 0.01, data = model_4a.train, kernel = "polynomial", degree = 2)
summary(model_4g)

model_4g_predict <- predict(model_4g, newdata = model_4a.test)
predict_vals <- table(predict = model_4g_predict, truth = model_4a.test$Purchase)
testing_error <- 1 - (sum(diag(predict_vals)) / length(model_4a.test$Purchase))

model_4g_predict <- predict(model_4g, newdata = model_4a.train)
predict_vals <- table(predict = model_4g_predict, truth = model_4a.train$Purchase)
training_error <- 1 - (sum(diag(predict_vals)) / length(model_4a.train$Purchase))

tune.out <- tune(svm, Purchase ~ ., data = model_4a.train, kernel = "polynomial", degree = 2, ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10)))
summary(tune.out)

model_4g <- svm(Purchase ~ ., cost= 10, data = model_4a.train, kernel = "polynomial", degree = 2)
summary(model_4g)

model_4g_predict <- predict(model_4g, newdata = model_4a.test)
predict_vals <- table(predict = model_4g_predict, truth = model_4a.test$Purchase)
testing_error <- 1 - (sum(diag(predict_vals)) / length(model_4a.test$Purchase))

model_4g_predict <- predict(model_4g, newdata = model_4a.train)
predict_vals <- table(predict = model_4g_predict, truth = model_4a.train$Purchase)
```

h. Overall, it looks like the polynomial with degree 2 fits the model better, but they are fairly similar and would most likely need to be validated to get a certain result. However, just from my experiment, it appears that the polynomial is the best model to fit this dataset.