DATA 558 Homework 5

Shrusti Ghela

6/3/2022

1. A note before you begin: In this problem, I will ask you to "write out an expression for a linear model." For instance, if I asked you to write out an expression for a linear model to predict an n-vector y using the columns of an $n \times p$ matrix X, then here's what I'd want to see:

$$y_i = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \epsilon_i$$

where ϵ_i is a mean-zero noise term.

Suppose we have an $n \times p$ data matrix X, and a continuous-valued response $y \in \mathbb{R}^n$. We saw in lecture that the mth principal component score vector is a linear combination of the p features, of the form

$$z_{im} = \phi_{1m} x_{i1} + \phi_{2m} x_{i2} + \ldots + \phi_{pm} x_{ip}$$

(e.g. see (12.2) and (12.4) in textbook). In principal components regression, we fit a linear model to predict y, but instead of using the columns of X as predictors, we use the first M principal component score vectors, where $M < p$.

a. Write out an expression for the linear model that we are fitting in principal components regression. Your answer should involve $y_i, z_{i1}, \ldots, z_{iM}$, a mean-zero noise term ϵ_i, and some coefficients.

$$y_i = \theta_0 + \theta_1 z_{i1} + \theta_2 z_{i2} + \ldots + \theta_m z_{im} + \epsilon_i; i = 1, \ldots n$$

b. Now plug in Equation 1 from this homework to your answer from (a), in order to express the principal components regression model in terms of $x_{i1}, \ldots, x_{ip}$

$$y_i = \theta_0 + \theta_1(\phi_{11} x_{i1} + \phi_{21} x_{i2} + \ldots + \phi_{p1} x_{ip}) + \theta_2(\phi_{12} x_{i1} + \phi_{22} x_{i2} + \ldots + \phi_{p2} x_{ip}) + \ldots +$$
$$\theta_m(\phi_{1m} x_{i1} + \phi_{2m} x_{i2} + \ldots + \phi_{pm} x_{ip}) + \epsilon_i$$

$$y_i = \theta_0 + (\theta_1 \phi_{11} + \theta_2 \phi_{12} + \ldots + \theta_m \phi_{1m}) x_{i1} + (\theta_1 \phi_{21} + \theta_2 \phi_{22} + \ldots + \theta_m \phi_{2m}) x_{i2} + \ldots +$$
$$(\theta_1 \phi_{p1} + \theta_2 \phi_{p2} + \ldots + \theta_m \phi_{pm}) x_{ip} + \epsilon_i$$

So,

$$y_i = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \epsilon_i$$

where,

$$\beta_0 = \theta_0$$

$$\beta_j = \theta_1 \phi_{j1} + \theta_2 \phi_{j2} + \ldots + \theta_m \phi_{jm} = \sum_{m=1}^{M} \theta_m \, \phi_{jm}; \forall j \in \{1, p\}$$

c. Use your answer from (b) to argue that the principal components regression model is linear in the columns of X.

PCR:

$$y_i = \theta_0 + \theta_1 z_{i1} + \theta_2 z_{i2} + \ldots + \theta_m z_{im} + \epsilon_i$$

could be re-written as:

$$y_i = \theta_0 + (\theta_1 \phi_{11} + \theta_2 \phi_{12} + \ldots + \theta_m \phi_{1m}) x_{i1} + (\theta_1 \phi_{21} + \theta_2 \phi_{22} + \ldots + \theta_m \phi_{2m}) x_{i2} + \ldots +$$
$$(\theta_1 \phi_{p1} + \theta_2 \phi_{p2} + \ldots + \theta_m \phi_{pm}) x_{ip} + \epsilon_i$$

The mean of the response variable is a linear combination of the parameters (regression coefficients) and the predictor variables (columns of X). Hence, the principal components regression model is linear in the columns of X.

d. In light of your answer to (c), is the following claim true? Explain your answer. Claim: Fitting a linear model to predict y using the first m principal components will yield the same fitted values as fitting a linear model to predict y using the columns of X.

Dimension reduction serves to constrain the estimated β_j coefficients, since now they must take the form $\beta_j = \sum_{m=1}^{M} \theta_m \, \phi_{jm}; \forall j \in \{1, p\}$. 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 a value of $M \ll p$ can significantly reduce the variance of the fitted coefficients. If $M = p$, and all the Z_m are linearly independent, then this poses no constraints. In this case, no dimension reduction occurs, and so fitting PCR is equivalent to performing least squares on the original p predictors.

2. We saw in class that K-means clustering minimizes the within-cluster sum of squares, given in (12.17) of the textbook. We can better understand the meaning of the within-cluster sum of squares by looking at (12.18) of the textbook. This shows us that the within-cluster sum of squares is (up to a scaling by a factor of two) the sum of squared distances from each observation to its cluster centroid.

a. Show computationally that (12.18) holds. You can do this by repeating this procedure a whole bunch of times:

- Simulate an $n \times p$ data matrix, as well as some clusters $C_1, \ldots, C_K$. (It doesn't matter whether there are any "true clusters" in your data, nor whether $C_1, \ldots, C_K$ correspond to these true clusters — (12.18) is a mathematical identity that should hold no matter what.)
- Compute the left-hand side of (12.18) on this data.
- Compute the right-hand side of (12.18) on this data.
- Verify that the left- and right-hand sides are equal. (If they aren't, then you have done something wrong!)

```
for (i in 1:10){
  n <- 1000
  p <- 5
  x <- matrix(rnorm(n * p), ncol = p)
  km <- kmeans(x, i, nstart = 20)
  a<-round(sum(km$withinss),3)
  b<-round(sum(rowSums((x-fitted(km))^2)),3)
  if (a!=b){
    print("LHS is not equalt to RHS")
  } else {
    print("LHS = RHS")
  }

}
```

```
## [1] "LHS = RHS"
## [1] "LHS = RHS"
## [1] "LHS = RHS"
## [1] "LHS = RHS"
## [1] "LHS = RHS"
## [1] "LHS = RHS"

## [1] "LHS = RHS"
## [1] "LHS = RHS"
## [1] "LHS = RHS"
## [1] "LHS = RHS"
```

Using simulation, we see that (12.18) holds.

b. *Extra-credit*: Show analytically that (12.18) holds. In other words, use algebra to prove (12.18).

(12.18) says,

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

This equation could be simply written as,

$$\sum_{i,j}||x_i - x_j||^2 = \sum_{i\neq j}||(x_i-\bar{x})-(x_j-\bar{x})||^2 = 2n\sum_{i=1}^{n}||x_i-\bar{x}||^2$$

where x_i and x_j are vectors from the same clusters.

But we can write,

$$||(x_i-\bar{x})-(x_j-\bar{x})||^2 = ||x_i-\bar{x}||^2 + ||x_j-\bar{x}||^2 - 2(x_i-\bar{x})^T(x_j-\bar{x})$$

Due to symmetry, we can evaluate the first two expressions of the above

$$\sum_{i\neq j}||x_i-\bar{x}||^2 = \sum_{i\neq j}||x_j-\bar{x}||^2 = (n-1)\sum_{i=1}^{n}||x_i-\bar{x}||^2$$

We know that,

$$\sum_{j\neq i}(x_j-\bar{x}) = \left[\sum_{j=1}^{n}(x_j-\bar{x})\right] - (x_i-\bar{x}) = n\bar{x}-n\bar{x}-(x_i-\bar{x}) = -(x_i-\bar{x})$$

Using this, the last expression is,

$$\sum_{i\neq j} -2(x_i-\bar{x})^T(x_j-\bar{x}) = -2\sum_{i=1}^{n}(x_i-\bar{x})^T\sum_{j\neq i}(x_j-\bar{x}) = 2\sum_{i=1}^{n}(x_i-\bar{x})^T(x_i-\bar{x})$$

$$= 2\sum_{i=1}^{n}||x_i-\bar{x}||^2$$

Finally, we have for one class,

$$(2(n-1)+2)\sum||x_i-\bar{x}||^2 = 2n\sum||x_i-\bar{x}||^2$$

So, for all the classes, we sum over all the classes and we get,

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

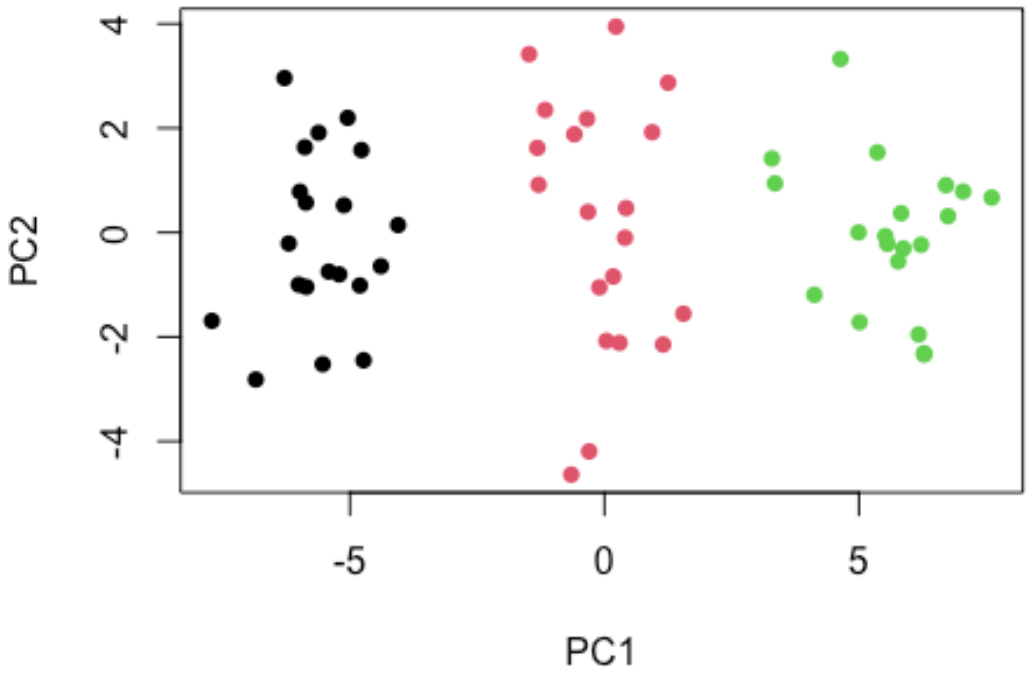
3. In this problem, you will generate simulated data, and then perform PCA and K-means clustering on the data.

a. Generate a simulated data set with 20 observations in each of three classes (i.e. 60 observations total), and 50 variables.

```
set.seed(132)
X <- rbind(matrix(rnorm(20*50, mean = 0), nrow = 20),
           matrix(rnorm(20*50, mean=0.8), nrow = 20),
           matrix(rnorm(20*50, mean=1.6), nrow = 20))
```

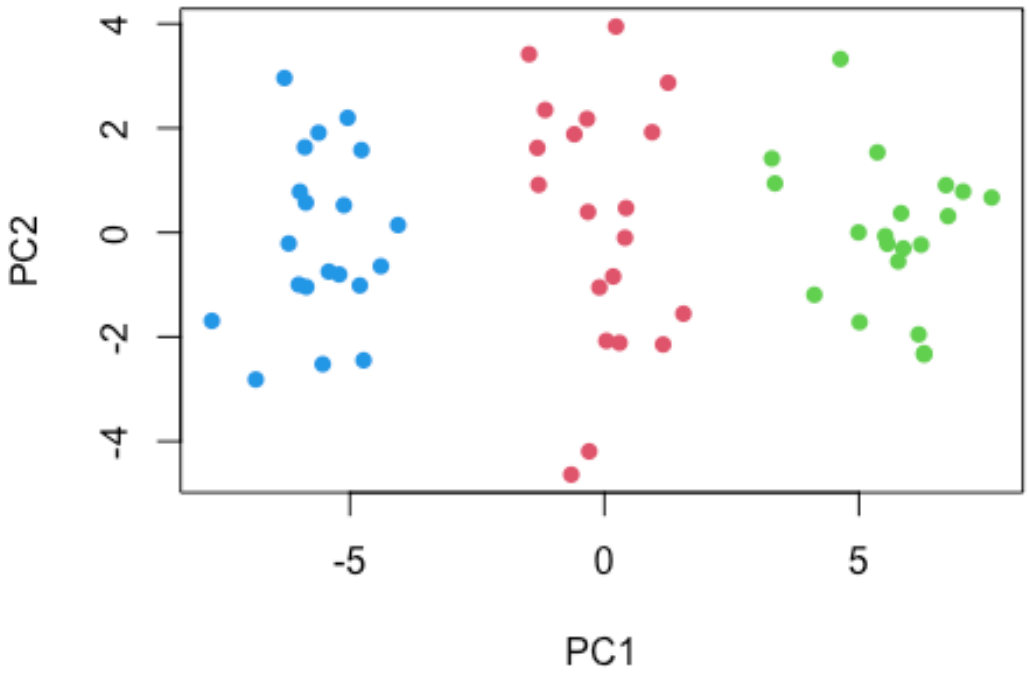
b. Perform PCA on the 60 observations and plot the first two principal component score vectors. Use a different color to indicate the observations in each of the three classes. If the three classes appear separated in this plot, then continue on to part (c). If not, then return to part (a) and modify the simulation so that there is greater separation between the three classes. Do not continue to part (c) until the three classes show at least some separation in the first two principal component score vectors.

```
X.pca = prcomp(X)$x
plot(X.pca[,1:2], col=c(rep(1,20), rep(2,20), rep(3,20)), pch=16)
```



c. Perform K-means clustering of the observations with K = 3. How well do the clusters that you obtained in K-means clustering compare to the true class labels? *Hint: You can use the table function in R to compare the true class labels to the class labels obtained by clustering. Be careful how you interpret the results: K-means clustering will arbitrarily number the clusters, so you cannot simply check whether the true class labels and clustering labels are the same.*

```
res = kmeans(X, centers = 3)
true_class = c(rep(1,20), rep(2,20), rep(3,20))
plot(X.pca[,1:2], col = (res$cluster + 1), pch=16)
```



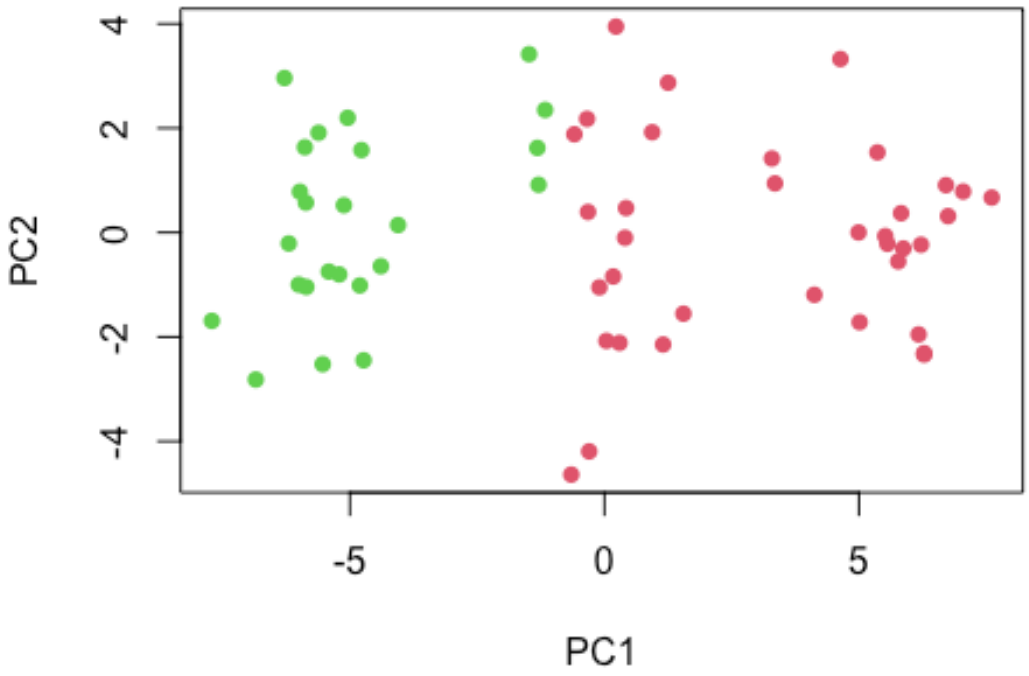

```
table(res$cluster, true_class)

##    true_class
##      1  2  3
##   1  0 20  0
##   2  0  0 20
##   3 20  0  0
```

All the points are perfectly clustered.

d. Perform K-means clustering with K = 2. Describe your results.

```
res = kmeans(X, centers = 2)
true = c(rep(1,20), rep(2,20), rep(3,20))
plot(X.pca[,1:2], col = (res$cluster + 1), pch=16)
```



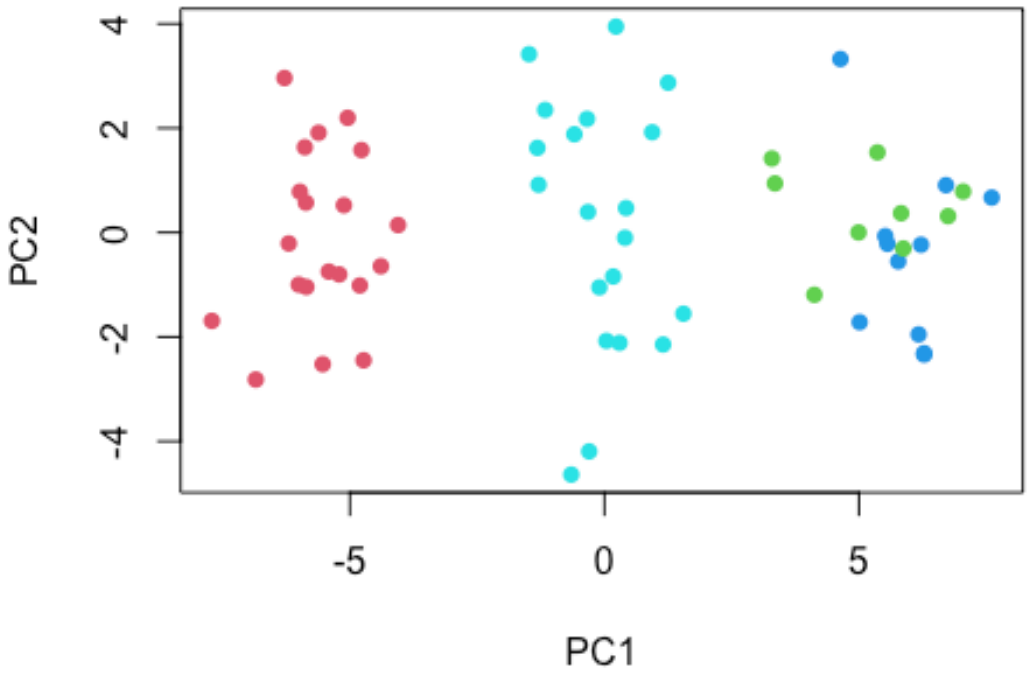

```
table(res$cluster, true_class)
```

```
##    true_class
##      1  2  3
##   1  0 16 20
##   2 20  4  0
```

The middle class is forced to be in either of the two clusters. The classes on the extreme ends are clustered correctly.

e. Now perform K-means clustering with K = 4, and describe your results.

```
res = kmeans(X, centers = 4)
true = c(rep(1,20), rep(2,20), rep(3,20))
plot(X.pca[,1:2], col = (res$cluster + 1), pch=16)
```



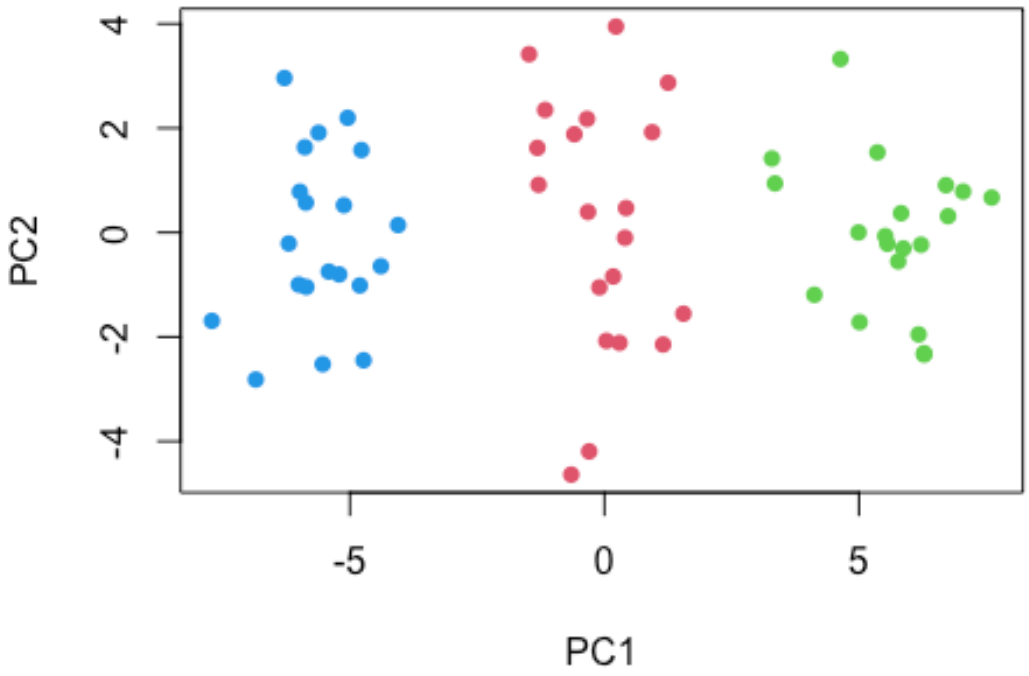

```
table(res$cluster, true_class)
```

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

One of the classes is forced to be split into two clusters.

f. Now perform K-means clustering with K = 3 on the first two principal component score vectors, rather than on the raw data. That is, perform K-means clustering on the 60 × 2 matrix of which the first column is the first principal component score vector, and the second column is the second principal component score vector. Comment on the results.

```
res = kmeans(X.pca[,1:2], centers = 3)
true = c(rep(1,20), rep(2,20), rep(3,20))
plot(X.pca[,1:2], col = (res$cluster + 1), pch=16)
```



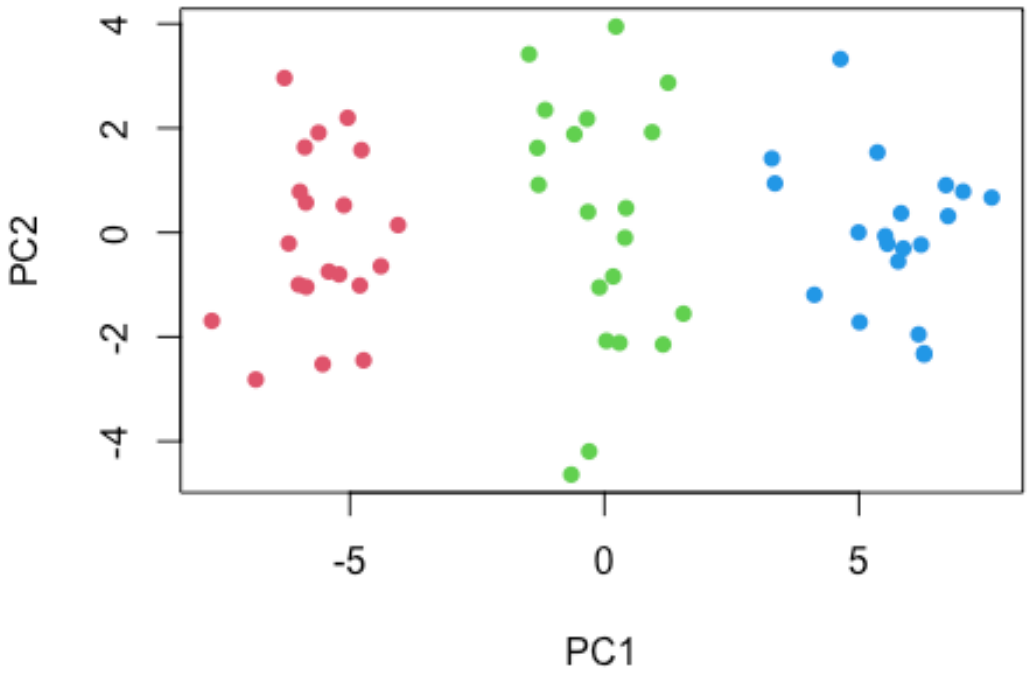

```
table(res$cluster, true_class)

##    true_class
##      1  2  3
##   1  0 20  0
##   2  0  0 20
##   3 20  0  0
```

This gives us a similar result as in (c.) This tells us that PCA carries adequate information.

g. Using the scale function, perform K-means clustering with K = 3 on thevdata after scaling each variable to have standard deviation one. How do these results compare to those obtained in (c)? Explain.

```
res = kmeans(scale(X), centers = 3)
true = c(rep(1,20), rep(2,20), rep(3,20))
plot(X.pca[,1:2], col = (res$cluster + 1), pch=16)
```




```
table(res$cluster, true_class)

##    true_class
##      1  2  3
##   1 20  0  0
##   2  0 20  0
##   3  0  0 20
```

This gives us a similar result as in (c.) and (g.) Scaling doesn't change the results.

4. This problem involves the OJ data set, which is part of the ISLR2 package.

```
library(ISLR2)
head(OJ)

##   Purchase WeekofPurchase StoreID PriceCH PriceMM DiscCH DiscMM SpecialCH
## 1       CH            237       1    1.75    1.99   0.00    0.0         0
## 2       CH            239       1    1.75    1.99   0.00    0.3         0
## 3       CH            245       1    1.86    2.09   0.17    0.0         0
## 4       MM            227       1    1.69    1.69   0.00    0.0         0
## 5       CH            228       7    1.69    1.69   0.00    0.0         0
## 6       CH            230       7    1.69    1.99   0.00    0.0         0
##   SpecialMM  LoyalCH SalePriceMM SalePriceCH PriceDiff Store7 PctDiscMM
## 1         0 0.500000        1.99        1.75      0.24     No  0.000000
## 2         1 0.600000        1.69        1.75     -0.06     No  0.150754
```

```
## 3         0 0.680000        2.09        1.69      0.40     No  0.000000
## 4         0 0.400000        1.69        1.69      0.00     No  0.000000
## 5         0 0.956535        1.69        1.69      0.00    Yes  0.000000
## 6         1 0.965228        1.99        1.69      0.30    Yes  0.000000
##   PctDiscCH ListPriceDiff STORE
## 1  0.000000          0.24     1
## 2  0.000000          0.24     1
## 3  0.091398          0.23     1
## 4  0.000000          0.00     1
## 5  0.000000          0.00     0
## 6  0.000000          0.30     0
```

a. Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```
dim(OJ)
```

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

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
set.seed(1000)
index = sample(1:nrow(OJ), 800)
train = OJ[index,]
test = OJ[-index,]
```

b. Fit a support vector classifier to the training data using cost=0.01, with Purchase as the response and the other variables as predictors. Use the summary() function to produce summary statistics, and describe the results obtained.

```
library(e1071)
svm.fit <- svm(Purchase ~., data = train, kernel = "linear", cost = 0.01)
summary(svm.fit)
```

```
##
## 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:  440
##
##  ( 219 221 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Number of support vectors is 440: 219 connected to class CH and 221 connected to class MM.

c. What are the training and test error rates?

```
svm.train.pred <- predict(svm.fit, train)
svm.test.pred <- predict(svm.fit, test)
train.error <- mean(svm.train.pred != train$Purchase)
test.error <- mean(svm.test.pred != test$Purchase)

train.error

## [1] 0.17625

test.error

## [1] 0.1555556
```

d. Use the tune() function to select an optimal cost. Consider values in the range 0.01 to 10.

```
svm.tune <- tune(svm, Purchase ~ ., data = train, kernel = "linear", ranges =
list(cost = c(0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10)))
summary(svm.tune)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.1725
##
## - Detailed performance results:
##    cost   error dispersion
## 1   0.01 0.18500 0.04281744
## 2   0.02 0.18250 0.04257347
## 3   0.05 0.18500 0.04073969
## 4   0.10 0.18375 0.03866254
```

```
## 5   0.20 0.18500 0.04031129
## 6   0.50 0.18250 0.04005205
## 7   1.00 0.17750 0.04322101
## 8   2.00 0.17750 0.04632314
## 9   5.00 0.17500 0.04823265
## 10 10.00 0.17250 0.04401704
```

```
svm.tune$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train, ranges = lis
t(cost = c(0.01,
##     0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  333
```

We get the best error when cost=1.

e. Compute the training and test error rates using this new value for cost.

```
svm.bestFit <- svm(Purchase ~., data = train, kernel = "linear", cost = 1)
svm.train.pred <- predict(svm.bestFit, train)
svm.test.pred <- predict(svm.bestFit, test)
train.error <- mean(svm.train.pred != train$Purchase)
test.error <- mean(svm.test.pred != test$Purchase)
```

```
train.error
```

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

```
test.error
```

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

The test error is 0.162963 and the train error is 0.1675

f. Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use the default value for gamma.

```
svm.fit <- svm(Purchase ~., data = train, kernel = "radial", cost = 0.01)
summary(svm.fit)
```

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

```
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##
## Number of Support Vectors:  620
##
##  ( 308 312 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Number of support vectors is 620: 308 connected to class CH and 312 connected to class MM.

```
svm.train.pred <- predict(svm.fit, train)
svm.test.pred <- predict(svm.fit, test)
train.error <- mean(svm.train.pred != train$Purchase)
test.error <- mean(svm.test.pred != test$Purchase)

train.error

## [1] 0.385

test.error

## [1] 0.4037037
```

The test error is 0.4037037 and the train error is 0.385

```
svm.tune <- tune(svm, Purchase ~ ., data = train, kernel = "radial", ranges =
list(cost = c(0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10)))
summary(svm.tune)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.5
##
## - best performance: 0.17625
##
## - Detailed performance results:
##    cost   error dispersion
## 1   0.01 0.38500 0.03106892
## 2   0.02 0.38500 0.03106892
```

```
## 3   0.05 0.23375 0.04678927
## 4   0.10 0.18375 0.03438447
## 5   0.20 0.18125 0.03240906
## 6   0.50 0.17625 0.03557562
## 7   1.00 0.17750 0.03763863
## 8   2.00 0.17875 0.03866254
## 9   5.00 0.18250 0.03827895
## 10 10.00 0.18125 0.04050463

svm.tune$best.model

##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train, ranges = lis
t(cost = c(0.01,
##     0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.5
##
## Number of Support Vectors:  408
```

We get the best error when cost=2

```
svm.bestFit <- svm(Purchase ~., data = train, kernel = "radial", cost = 2)
svm.train.pred <- predict(svm.bestFit, train)
svm.test.pred <- predict(svm.bestFit, test)
train.error <- mean(svm.train.pred != train$Purchase)
test.error <- mean(svm.test.pred != test$Purchase)

train.error

## [1] 0.14375

test.error

## [1] 0.1703704
```

The test error is 0.1703704 and the train error is 0.14375

g. Repeat parts (b) through (e) using a support vector machine with a polynomial kernel. Set degree=2.

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

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

```
##     cost = 0.01, degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  620
##
##  ( 308 312 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Number of support vectors is 620: 308 connected to class CH and 312 connected to class MM.

```
svm.train.pred <- predict(svm.fit, train)
svm.test.pred <- predict(svm.fit, test)
train.error <- mean(svm.train.pred != train$Purchase)
test.error <- mean(svm.test.pred != test$Purchase)

train.error

## [1] 0.385

test.error

## [1] 0.4037037
```

The test error is 0.4037037 and the train error is 0.385

```
svm.tune <- tune(svm, Purchase ~ ., data = train, kernel = "polynomial", rang
es = list(cost = c(0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10)), degree = 2
)
summary(svm.tune)

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

```
## - best performance: 0.1775
##
## - Detailed performance results:
##     cost   error dispersion
## 1   0.01 0.38500 0.04958158
## 2   0.02 0.37625 0.04581439
## 3   0.05 0.33375 0.04251225
## 4   0.10 0.31500 0.03374743
## 5   0.20 0.23875 0.03458584
## 6   0.50 0.20375 0.04641674
## 7   1.00 0.19250 0.05006940
## 8   2.00 0.18375 0.05104804
## 9   5.00 0.18125 0.04007372
## 10 10.00 0.17750 0.04241004
```

```
svm.tune$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train, ranges = lis
t(cost = c(0.01,
##     0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10)), kernel = "polynomial",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  348
```

We get the best error when cost=5

```
svm.bestFit <- svm(Purchase ~., data = train, kernel = "polynomial", cost = 5
)
svm.train.pred <- predict(svm.bestFit, train)
svm.test.pred <- predict(svm.bestFit, test)
train.error <- mean(svm.train.pred != train$Purchase)
test.error <- mean(svm.test.pred != test$Purchase)
```

```
train.error
```

```
## [1] 0.145
```

```
test.error
```

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

The test error is 0.1666667 and the train error is 0.145

h. Overall, which approach seems to give the best results on this data?

The linear kernel acctually seemed to do the best.