DATA 558: HW Assignment 5

Hriday Baghar

Jun 9, 2022

Instructions:

You may discuss the homework problems in small groups, but you must write up the final solutions and code yourself. Please turn in your code for the problems that involve coding. However, code without written answers will receive no credit. To receive credit, you must explain your answers and show your work. All plots should be appropriately labeled and legible, with axis labels, legends, etc., as needed.

On this assignment, some of the problems involve random number generation. Be sure to set a random seed (using the command `set.seed()`*) before you begin.*

```
library(ISLR2)
library(MASS)
library(ggplot2)
library(e1071)
library(dplyr)
```

1. 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

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

(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 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: $y_i = \beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip} + \epsilon_i$, where ϵ_i is a mean-zero noise term.

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

$$y_i = \beta_0 + \beta_1 z_{i1} + \beta_2 z_{i2} + ... + \beta_M z_{iM} + \varepsilon_i, \quad i = 1, ..., n$$

where $\beta_0, ..., \beta_m$ are the regression coefficients fit using least squares and M is the number of principal components and $z_{i1}, ..., z_{iM}$ are the principal components for observation i.

(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 $x_{i1}, ..., x_{ip}$.

$$\begin{aligned} y_i =& \beta_0 + \beta_1(\phi_{11}x_{i1} + \phi_{21}x_{i2} + ... + \phi_{p1}x_{ip}) + ... \\ &+ \beta_M(\phi_{1M}x_{i1} + \phi_{2M}x_{i2} + ... + \phi_{pM}x_{ip}) + \varepsilon_i, \quad i = 1, ..., n \end{aligned} \tag{1}$$

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

We see that the above expression is still a linear combination of β, which is what we mean when we say that the model is linear. For example for x_{i1} we have a coefficient $\beta_1\phi_{11} + \beta_2\phi_{12} + ... + \phi_{1M}\beta_M$. This is still a linear combination of β, hence it 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.*

The given claim is not always true. In cases where the number of principal components is less than the number of features i.e. $M < p$ the results will not be the same. This is because, using lesser principal components will change the expression of the coefficient for each x_{ip} as we can see from the expression in (c). This makes sense because each principal component is able to explain a certain percentage of variance in the data, and reducing the number of principal components in principal components regression will be fit on a dataset with reduced dimensions, giving different coefficient estimates.

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 $C_1, ..., C_K$. (It doesn't matter whether there are any "true clusters" in your data, nor whether $C_1, ..., C_K$ 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!)

We define functions to repeatedly create the data matrix, calculate LHS and RHS.

```
set.seed(42)
sim.data <- function(p,k,n) {
  mu <- sample(seq(-10:10), p, replace = TRUE)
  A <- matrix(runif(p^2)*2-1, ncol=p)
  var <- t(A) %*% A
  cluster <- sample(seq(1,k), n, replace = TRUE)

  data <- cbind(mvrnorm(n = n, mu = mu, Sigma = var), cluster)

  return(data)
}
```

Now let us compute the LHS of the given equality,

```
get.cluster.lhs <- function(c,data){
  sum <- 0
  x <- data[data[,ncol(data)]==c,-c(ncol(data))]

  for(i in 1:nrow(x)){
    for(i1 in 1:nrow(x)){
      sum <- sum + sum((x[i,]-x[i1,])^2)
    }
  }
  return(sum*1/nrow(x))
}
```

And now RHS,

```
get.cluster.rhs <- function(c,data){
  sum <- 0
  x <- data[data[,ncol(data)]==c,-c(ncol(data))]
  for(i in 1:nrow(x)){
      for(j in 1:ncol(x)){
        sum <- sum + (x[i,j] - mean(x[,j]))^2
      }
  }
  return(sum*2)
}
```

Using the above 3 functions we repeat the experiment 1000 times.

```
perform.experiment <- function(){
  n <- sample(50:100, 1)
  p <- sample(5:10, 1)
  k <- sample(2:5, 1)

  data <- sim.data(p,k,n)

  lhs <- sapply(1:k, function(x) get.cluster.lhs(x,data))
  rhs <- as.vector(sapply(1:k, function(x) get.cluster.rhs(x,data)))

  return(all.equal(lhs,rhs))
}
results <- replicate(1000, perform.experiment())
#If below result is true then we know the equality holds
all.equal(results, rep(TRUE,1000))
```

```
## [1] TRUE
```

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

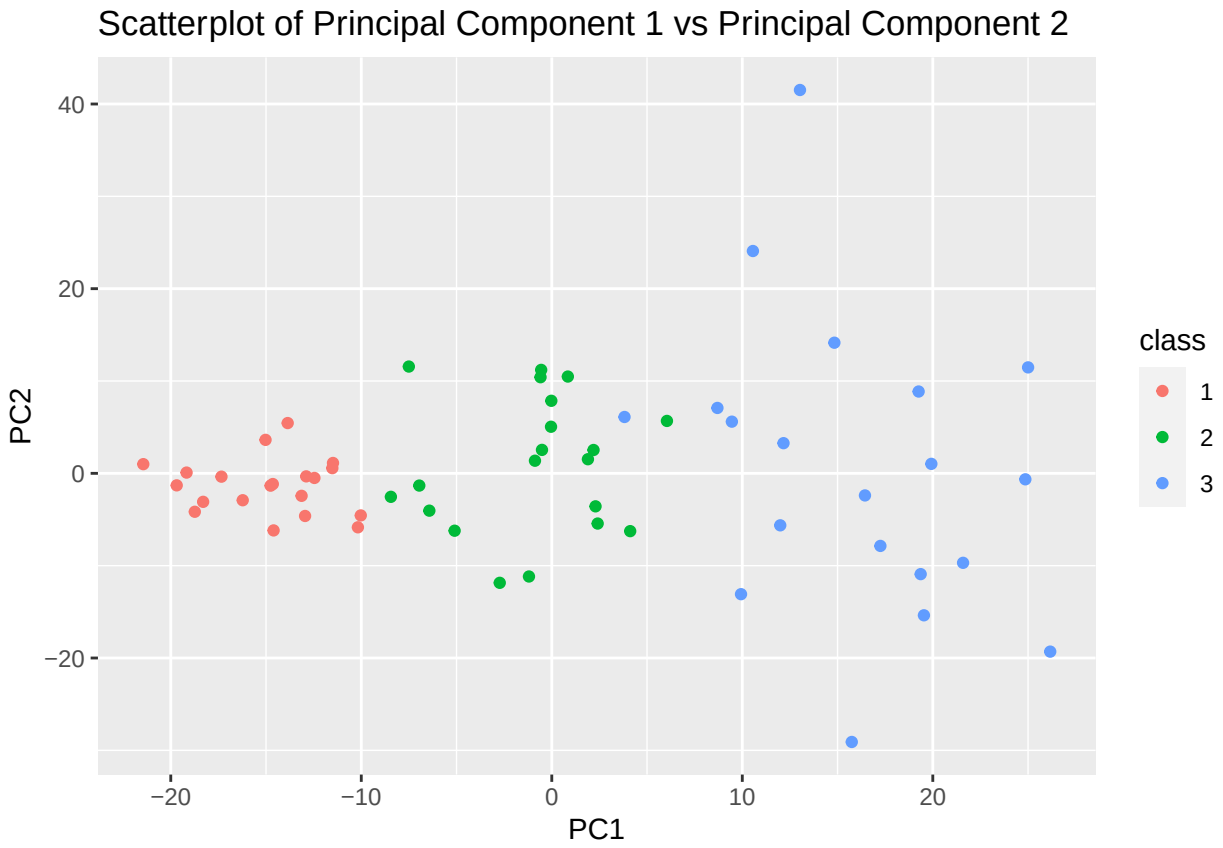
3. In this problem, you will generate simulated data, and then perform PCA and K-means clustering on the data.

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

```
set.seed(42)
mu <- c(0,2,4)
var <- matrix(c(10, 0.5, 0.7, 0.5, 20, 0.6, 0.7, 0.6, 40), 3,3)

raw.data <- mvrnorm(n = 1000, mu = mu, Sigma = var)

data <- data.frame(rbind(
  matrix(raw.data[,1], nrow=20, ncol=50),
  matrix(raw.data[,2], nrow=20, ncol=50),
  matrix(raw.data[,3], nrow=20, ncol=50)
  ),
  class = cbind(c(rep(1,20), rep(2,20), rep(3,20)))
)
```

(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.

```
pca <- prcomp(data[,-c(51)])

pca.data <- data.frame(PC1 = pca$x[,"PC1"], PC2 = pca$x[,"PC2"],
                       class = as.factor(data$class))

ggplot(data = pca.data, aes(x=PC1, y=PC2, color=class))+
  geom_point()+
  labs(title="Scatterplot of Principal Component 1 vs Principal Component 2")
```



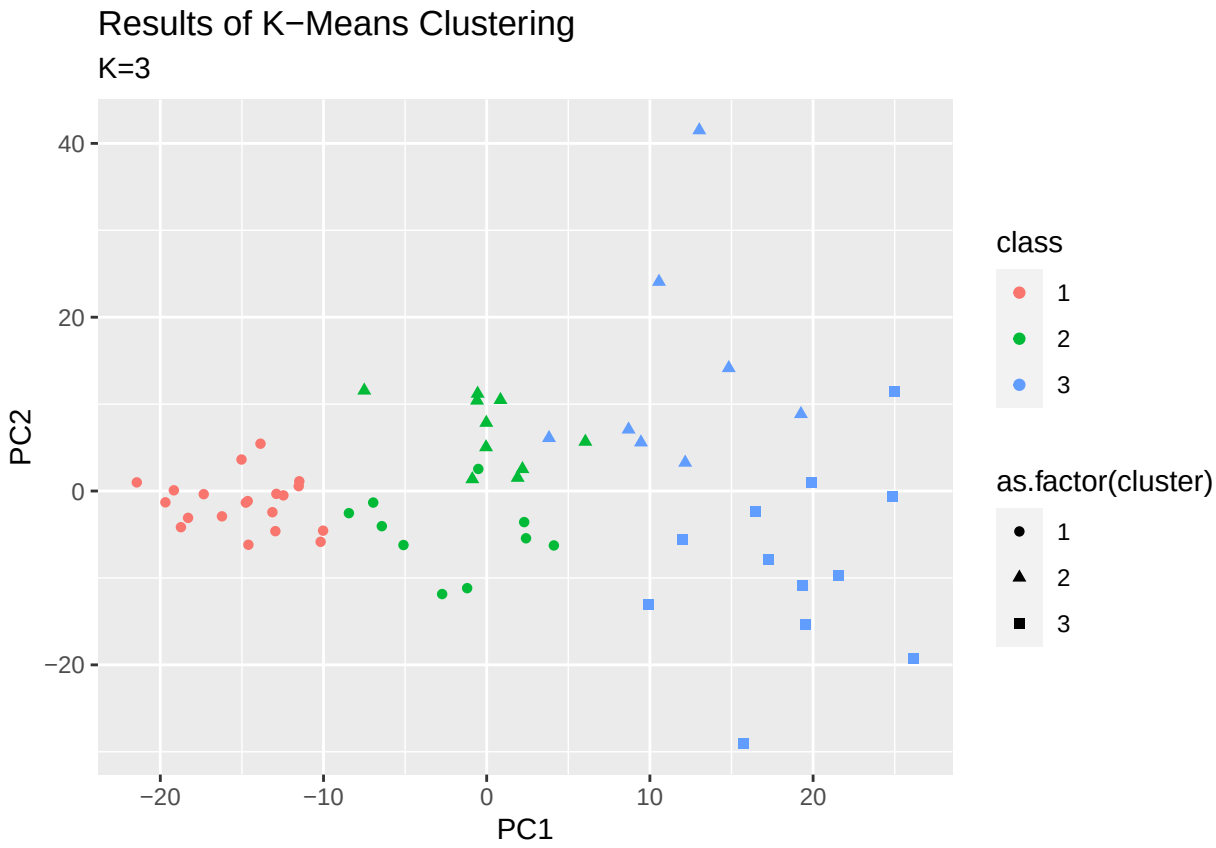
(c) Perform K-means clustering of the observations with $K = 3$. How well do the clusters that you obtained in K-means clustering compare to the true class labels?

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.*

We define a function to display the result table and plot the first 2 principal components with the true cluster and assigned cluster to visualize the tabulated results. The colors denote the original cluster from the data generation step and the shapes denote the assigned cluster.

```
get.cluster.plot <- function(K=3){
  km <- kmeans(data[,-c(51)], K, nstart = 20)
  df <- data.frame(pca.data, cluster = as.factor(km$cluster))
  print(table(km$cluster, data[,"class"]))
  ggplot(df, aes(x=PC1, y=PC2, shape = as.factor(cluster), color = class))+
    geom_point()+
    labs(title="Results of K-Means Clustering", subtitle = paste("K",K,sep="="))
}
get.cluster.plot()
```

```
##
##       1  2  3
##   1 20 10  0
##   2  0 10  8
##   3  0  0 12
```

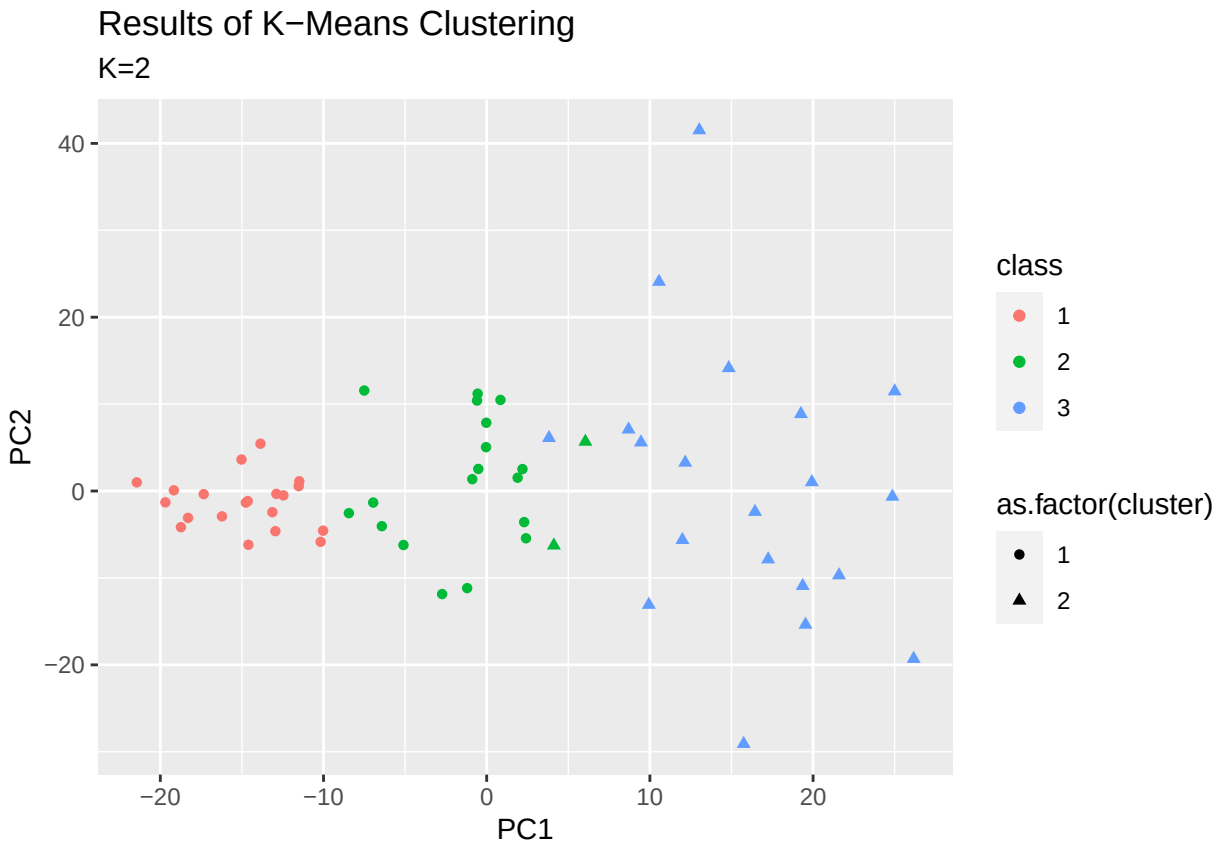


We see that one cluster has all 20 observations correctly classified. The other 2 clusters have 10 and 8 misclassifications respectively. The algorithm does not do a good job of identifying the green cluster. The blue cluster is also not captured very well.

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

```
get.cluster.plot(2)
```

```
##
##       1  2  3
##   1 20 18  0
##   2  0  2 20
```

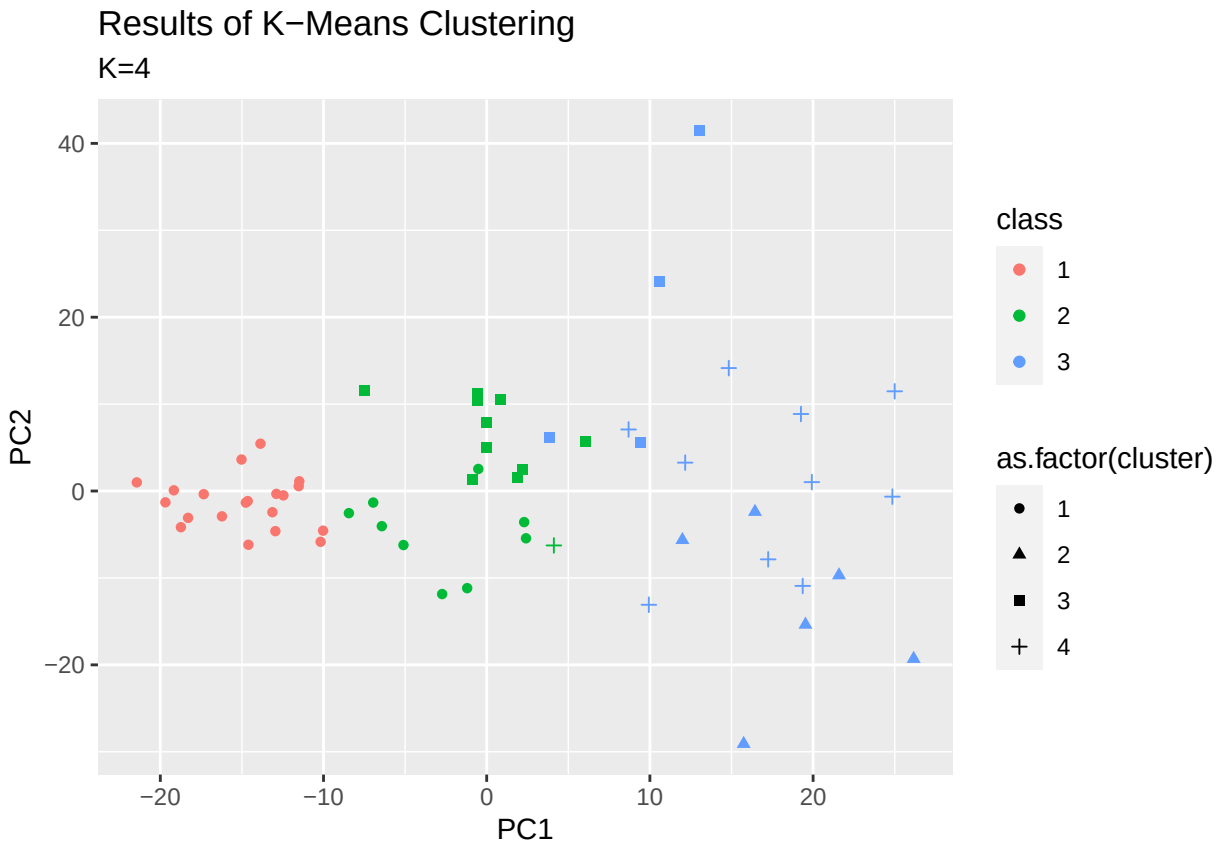


The algorithm counts most green cluster observations in the red cluster (18 of them) and the remaining 2 go in the blue cluster.

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

```
get.cluster.plot(4)
```

```
##
##       1  2  3
##   1 20  9  0
##   2  0  0  6
##   3  0 10  4
##   4  0  1 10
```



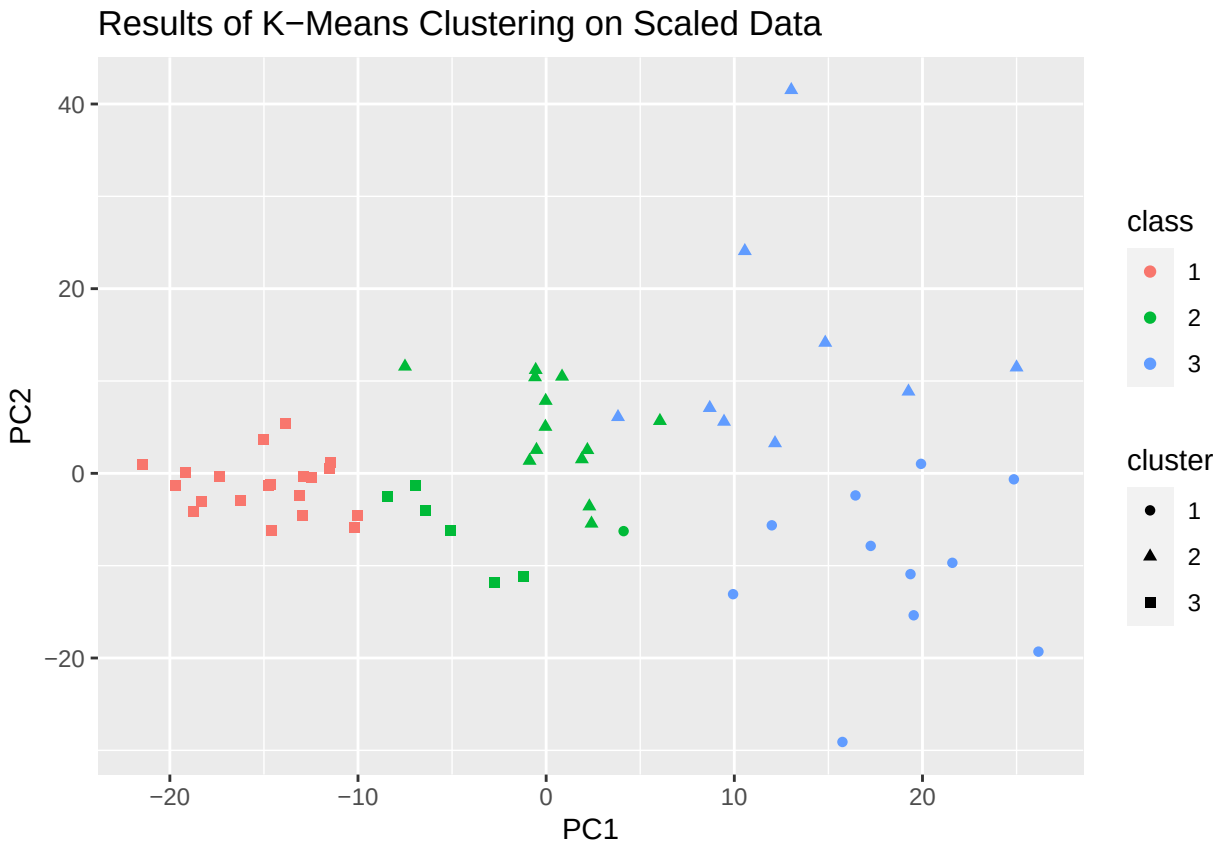
Again, the red cluster (with lower variance) is well captured. The extra cluster that the algorithm determines has only 6 observations. The green and blue clusters are once again only partially identified. The blue cluster also appears to have a strange separation which appears to be non-linear in the 2D PCA plot (note the blue triangles and squares).

(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.

```
km.2f <- kmeans(pca.data[,-c(3)], 3, nstart = 20)
table(km.2f$cluster, data[,"class"])
```

```
##
##      1  2  3
##   1  0  1 11
##   2  0 13  9
##   3 20  6  0
```

```
pca.data <- data.frame(pca.data, cluster = as.factor(km.2f$cluster))

ggplot(data = pca.data, aes(x=PC1, y=PC2, color=class, shape=cluster))+
  geom_point()+
  labs(title="Results of K-Means Clustering on Scaled Data")
```



Using the first two Principal components we see that there is an improvement in identifying the green cluster.

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

```
km.2g <- kmeans(scale(data[,-c(51)]), 3, nstart = 20)
table(km.2g$cluster, data[,"class"])
```

```
##
##      1  2  3
##   1 20  6  0
##   2  0  3 14
##   3  0 11  6
```

```
pca <- prcomp(data.frame(scale(data[,-c(51)])))


pca.data <- data.frame(PC1 = pca$x[,"PC1"], PC2 = pca$x[,"PC2"],
                       class = as.factor(data$class),
                       cluster = as.factor(km.2g$cluster))

ggplot(data = pca.data, aes(x=PC1, y=PC2, color=class, shape=cluster))+
  geom_point()+
  labs(title="Results of K-Means Clustering on Scaled Data")
```

Applying K-means on the scaled data does a better job of identifying the blue cluster and perhaps also the green cluster. It is worth noting that the blue cluster is the one with the largest mean and variance and this result is consistent with the idea of what scaling tries to achieve.

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

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

```
set.seed(42)
train <- sample(nrow(OJ), size = 800, replace = FALSE)
```

(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.

```
svm.4b <- svm(Purchase ~ ., data = OJ, subset = train, cost = 0.01, kernel = "linear")
summary(svm.4b)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ, cost = 0.01, kernel = "linear",
##     subset = train)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  432
##
##  ( 215 217 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

There were 432 support vectors in this model out of which 215 were assigned to CH and 217 assigned to MM.

(c) What are the training and test error rates?

```
#Defining function to reuse
get.svm.metrics <- function(model){
  train.preds <- table(predicted = predict(model, OJ[train,]),
                       true = OJ[train, "Purchase"])
  print("Confusion matrix: Training data")
  print(train.preds)
  train.error <- round(1 - sum(diag(train.preds))/sum(train.preds),2)

  test.preds <- table(predict(model, OJ[-train,]), OJ[-train, "Purchase"])
  print("Confusion matrix: Test data")
  print(test.preds)
  test.error <- round(1 - sum(diag(test.preds))/sum(test.preds),2)

  return(c(train.error, test.error))
```

```
}

error <- get.svm.metrics(svm.4b)
```

```
## [1] "Confusion matrix: Training data"
##           true
## predicted  CH  MM
##        CH 432  77
##        MM  60 231
## [1] "Confusion matrix: Test data"
##
##       CH  MM
##   CH 142  25
##   MM  19  84
```

Training error rate = 0.17 Test error rate = 0.16

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

```
tune.out <- tune(svm, Purchase ~ ., data = OJ[train,], kernel = "linear",
                 ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 5, 10)))

summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.1775
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.18250 0.04133199
## 2  0.05 0.18250 0.04794383
## 3  0.10 0.18000 0.04901814
## 4  0.50 0.18125 0.04050463
## 5  1.00 0.17750 0.04031129
## 6  5.00 0.17875 0.03821086
## 7 10.00 0.18375 0.03438447
```

```
tune.out$best.parameters$cost
```

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

The best cost to use is 1.

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

```
svm.4e <- svm(Purchase ~ ., data = OJ, kernel = "linear",
              cost = tune.out$best.parameters$cost, subset = train)
```

```
summary(svm.4e)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ, kernel = "linear", cost = tune.out$best.parameters$cost,
##     subset = train)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  331
##
##  ( 166 165 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
error <- get.svm.metrics(svm.4e)
```

```
## [1] "Confusion matrix: Training data"
##           true
## predicted  CH  MM
##        CH 434  76
##        MM  58 232
## [1] "Confusion matrix: Test data"
##
##       CH  MM
##   CH 140  23
##   MM  21  86
```

The model has 331 support vectors - 166 belong to CH and 165 belong to MM.

Training error rate = 0.17 Test error rate = 0.16

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

Using radial kernel and cost = 0.01:

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

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ, cost = 0.01, kernel = "radial",
##     subset = train)
##
##
## Parameters:
##    SVM-Type:  C-classification
```

```
##  SVM-Kernel:  radial
##        cost:  0.01
##
## Number of Support Vectors:  621
##
##  ( 308 313 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
error <- get.svm.metrics(svm.4f)
```

```
## [1] "Confusion matrix: Training data"
##          true
## predicted  CH  MM
##        CH 492 308
##        MM   0   0
## [1] "Confusion matrix: Test data"
##
##       CH  MM
##   CH 161 109
##   MM   0   0
```

Training error rate = 0.38 Test error rate = 0.4

Finding optimal value of cost:

```
tune.out <- tune(svm, Purchase ~ ., data = OJ[train,], kernel = "radial",
                 ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 5, 10)))

summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.5
##
## - best performance: 0.17375
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.38500 0.06942222
## 2  0.05 0.21500 0.04556741
## 3  0.10 0.18750 0.05335937
## 4  0.50 0.17375 0.05050096
## 5  1.00 0.18000 0.04972145
## 6  5.00 0.18625 0.05252314
## 7 10.00 0.19250 0.05309844
```

Cost = 0.5 is the optimal value. Let us fit a model using this value of cost.

```
svm.4f.best <- svm(Purchase ~ ., data = OJ, subset = train,
                   cost = tune.out$best.parameters$cost, kernel = "radial")
summary(svm.4f.best)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ, cost = tune.out$best.parameters$cost,
##     kernel = "radial", subset = train)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.5
##
## Number of Support Vectors:  404
##
##  ( 202 202 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
error <- get.svm.metrics(svm.4f.best)
```

```
## [1] "Confusion matrix: Training data"
##           true
## predicted  CH  MM
##        CH 449  79
##        MM  43 229
## [1] "Confusion matrix: Test data"
##
##       CH  MM
##   CH 145  27
##   MM  16  82
```

Training error rate = 0.15 Test error rate = 0.16

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

```
svm.4g <- svm(Purchase ~ ., data = OJ, subset = train, cost = 0.01,
              kernel = "polynomial", degree = 2)
summary(svm.4g)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ, cost = 0.01, kernel = "polynomial",
##     degree = 2, subset = train)
##
##
## Parameters:
##    SVM-Type:  C-classification
```

```
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  621
##
##  ( 308 313 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
error <- get.svm.metrics(svm.4g)
```

```
## [1] "Confusion matrix: Training data"
##           true
## predicted  CH  MM
##        CH 492 308
##        MM   0   0
## [1] "Confusion matrix: Test data"
##
##       CH  MM
##   CH 161 109
##   MM   0   0
```

Training error rate = 0.38 Test error rate = 0.4

Let us find the optimal value of cost.

```
tune.out <- tune(svm, Purchase ~ ., data = OJ[train,],
                 kernel = "polynomial", degree = 2,
                 ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 5, 10)))

summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     5
##
## - best performance: 0.185
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.38500 0.06146363
## 2  0.05 0.32500 0.05951190
## 3  0.10 0.31125 0.05447030
## 4  0.50 0.20375 0.04450733
## 5  1.00 0.19875 0.05152197
```

```
## 6  5.00 0.18500 0.04362084
## 7 10.00 0.18625 0.04226652
```

The optimal value for cost = 5. Let us fit a model using this value.

```
svm.4g.best <- svm(Purchase ~ ., data = OJ, subset = train,
              cost = tune.out$best.parameters$cost, kernel = "polynomial", degree = 2)
summary(svm.4g.best)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ, cost = tune.out$best.parameters$cost,
##     kernel = "polynomial", degree = 2, subset = train)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  5
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  365
##
##  ( 177 188 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
error <- get.svm.metrics(svm.4g.best)
```

```
## [1] "Confusion matrix: Training data"
##           true
## predicted  CH  MM
##        CH 459  85
##        MM  33 223
## [1] "Confusion matrix: Test data"
##
##       CH  MM
##   CH 146  30
##   MM  15  79
```

Training error rate = 0.15 Test error rate = 0.17

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

The SVM with a radial kernel with the best value of cost identified through cross validation gives the best result on both the training and test sets.