SDS Homework 7

Matthew Bradley, Ayanna Fisher, Hayley Zorkic, Ben Howell

4/29/2022

Question 7:

a:

```
library(ISLR)
library(e1071)
data(Auto)

Auto$gasMedian <- ifelse(Auto$mpg > median(Auto$mpg), 1, 0)
```

b:

```
# set seed so markdown output is the same each time
set.seed(1)
Auto$gasMedian = as.factor(Auto$gasMedian)
newAuto = Auto[2:10]
tune_out = tune(svm, gasMedian ~ ., kernel = "linear", data = newAuto, ranges = list(cost = c(.001, .01
summary(tune_out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.08673077
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-03 0.13525641 0.05661708
## 2 1e-02 0.08923077 0.04698309
## 3 1e-01 0.08673077 0.04040897
## 4 1e+00 0.09961538 0.04923181
## 5 5e+00 0.11230769 0.05826857
## 6 1e+01 0.11237179 0.05701890
## 7 1e+02 0.11750000 0.06208951
## 8 1e+03 0.13525641 0.06613861
```

The tune function uses 10-fold cross validation to find the linear svm with the best cost. The best linear svm

has a cost of 0.01, which has the lowest error at 0.0867. For all of the fitted costs, the errors were below 0.14. Dispersions are all similar, between 0.05 and 0.06.

c:

```
set.seed(1)
tune_outRadial = tune(svm, gasMedian ~ ., data = newAuto, kernel = "radial", ranges = list(cost = c(.00

tune_outPoly = tune(svm, gasMedian ~ ., data = newAuto, kernel = "polynomial", ranges = list(cost = c(.

summary(tune_outRadial)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10     1
##
## - best performance: 0.07897436
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-03  0.01 0.55115385 0.04366593
## 2  1e-02  0.01 0.55115385 0.04366593
## 3  1e-01  0.01 0.11224359 0.03836937
## 4  1e+00  0.01 0.08673077 0.04551036
## 5  5e+00  0.01 0.08673077 0.04040897
## 6  1e+01  0.01 0.08673077 0.03855882
## 7  1e+02  0.01 0.09692308 0.05742483
## 8  1e-03  0.10 0.55115385 0.04366593
## 9  1e-02  0.10 0.26564103 0.10022083
## 10 1e-01  0.10 0.08666667 0.04193895
## 11 1e+00  0.10 0.08923077 0.04376306
## 12 5e+00  0.10 0.08423077 0.04689205
## 13 1e+01  0.10 0.08416667 0.05256241
## 14 1e+02  0.10 0.10211538 0.04535762
## 15 1e-03  1.00 0.55115385 0.04366593
## 16 1e-02  1.00 0.55115385 0.04366593
## 17 1e-01  1.00 0.55115385 0.04366593
## 18 1e+00  1.00 0.07903846 0.04891067
## 19 5e+00  1.00 0.08147436 0.04910668
## 20 1e+01  1.00 0.07897436 0.04869339
## 21 1e+02  1.00 0.07897436 0.04869339
## 22 1e-03  5.00 0.55115385 0.04366593
## 23 1e-02  5.00 0.55115385 0.04366593
## 24 1e-01  5.00 0.55115385 0.04366593
## 25 1e+00  5.00 0.48967949 0.05080301
## 26 5e+00  5.00 0.48211538 0.05914633
## 27 1e+01  5.00 0.48211538 0.05914633
## 28 1e+02  5.00 0.48211538 0.05914633
## 29 1e-03 10.00 0.55115385 0.04366593
```

```
## 30 1e-02 10.00 0.55115385 0.04366593
## 31 1e-01 10.00 0.55115385 0.04366593
## 32 1e+00 10.00 0.51794872 0.04766442
## 33 5e+00 10.00 0.51794872 0.04766442
## 34 1e+01 10.00 0.51794872 0.04766442
## 35 1e+02 10.00 0.51794872 0.04766442
```

```
summary(tune_outPoly)
```

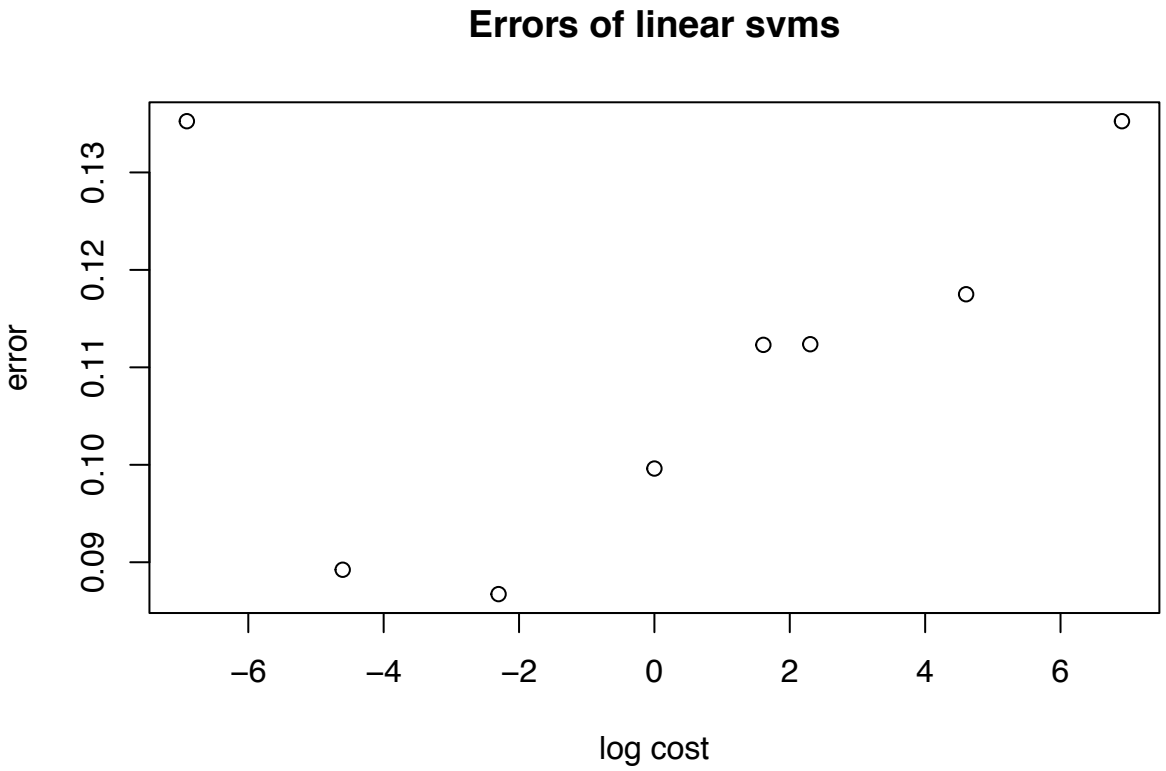
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##   100      1
##
## - best performance: 0.08403846
##
## - Detailed performance results:
##     cost degree      error dispersion
## 1  1e-03      1 0.60192308 0.06346118
## 2  1e-02      1 0.60192308 0.06346118
## 3  1e-01      1 0.36461538 0.09994785
## 4  1e+00      1 0.10705128 0.07510919
## 5  5e+00      1 0.08910256 0.05253109
## 6  1e+01      1 0.08653846 0.04956160
## 7  1e+02      1 0.08403846 0.06254843
## 8  1e+03      1 0.09698718 0.07021827
## 9  1e-03      2 0.60192308 0.06346118
## 10 1e-02      2 0.60192308 0.06346118
## 11 1e-01      2 0.60192308 0.06346118
## 12 1e+00      2 0.60192308 0.06346118
## 13 5e+00      2 0.60192308 0.06346118
## 14 1e+01      2 0.58416667 0.07806609
## 15 1e+02      2 0.31634615 0.07262899
## 16 1e+03      2 0.29346154 0.07790084
## 17 1e-03      3 0.60192308 0.06346118
## 18 1e-02      3 0.60192308 0.06346118
## 19 1e-01      3 0.60192308 0.06346118
## 20 1e+00      3 0.60192308 0.06346118
## 21 5e+00      3 0.60192308 0.06346118
## 22 1e+01      3 0.60192308 0.06346118
## 23 1e+02      3 0.44852564 0.13181806
## 24 1e+03      3 0.25750000 0.08857504
## 25 1e-03      4 0.60192308 0.06346118
## 26 1e-02      4 0.60192308 0.06346118
## 27 1e-01      4 0.60192308 0.06346118
## 28 1e+00      4 0.60192308 0.06346118
## 29 5e+00      4 0.60192308 0.06346118
## 30 1e+01      4 0.60192308 0.06346118
## 31 1e+02      4 0.60192308 0.06346118
## 32 1e+03      4 0.60192308 0.06346118
## 33 1e-03      5 0.60192308 0.06346118
```

```
## 34 1e-02      5 0.60192308 0.06346118
## 35 1e-01      5 0.60192308 0.06346118
## 36 1e+00      5 0.60192308 0.06346118
## 37 5e+00      5 0.60192308 0.06346118
## 38 1e+01      5 0.60192308 0.06346118
## 39 1e+02      5 0.60192308 0.06346118
## 40 1e+03      5 0.60192308 0.06346118
```

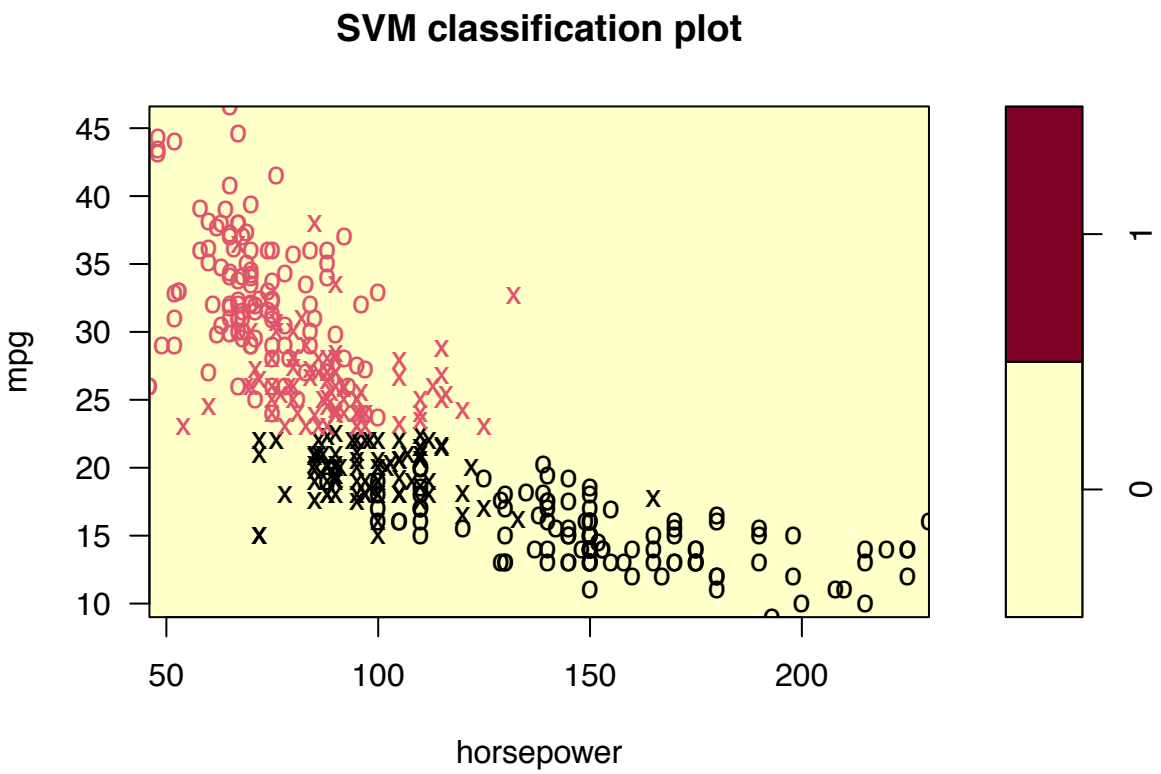
Here we used the tune function again, which is the same as we used in part b. For the radial svm model, it found the best values of cost and gamma to be 1 for each. For the polynomial model it found the best cost and degree to be 10 and 1 respectively. The lowest errors for the best radial and polynomial models are 0.07897 and 0.084038 respectively.

d:

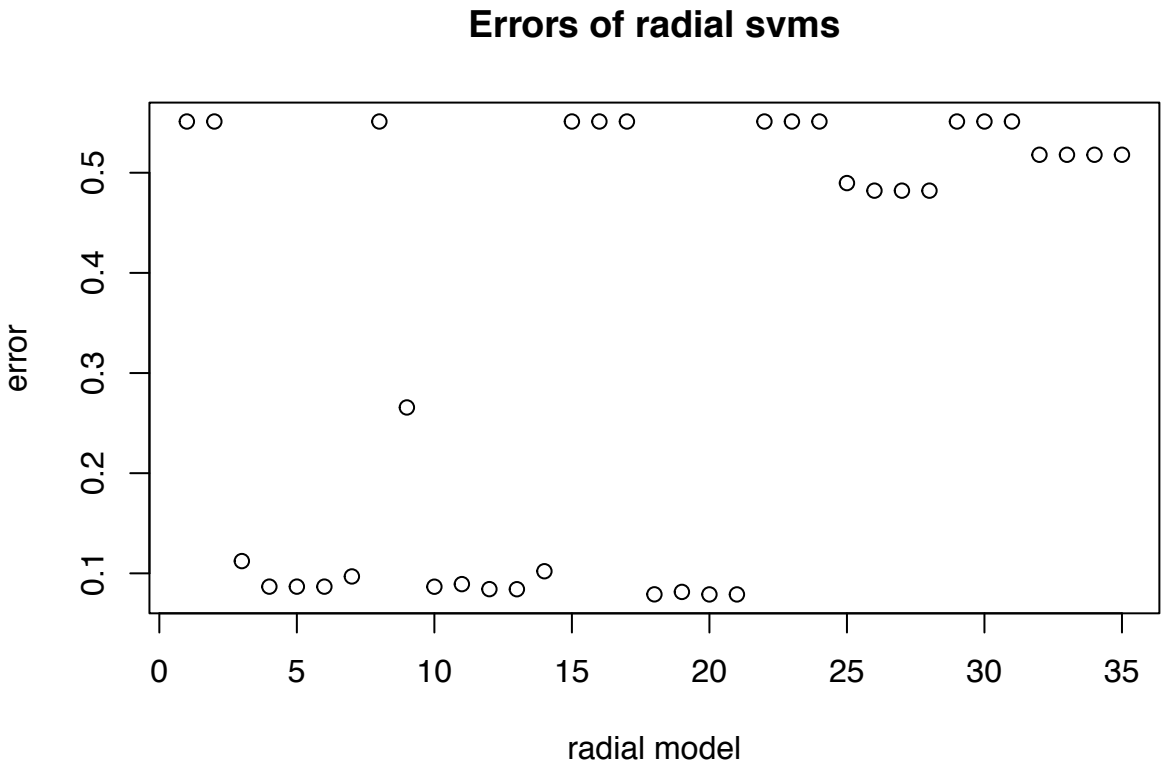
```
plot(y = tune_out$performances$error, x = log(tune_out$performances$cost), ylab = "error", xlab = "log c
      main = "Errors of linear svms")
```



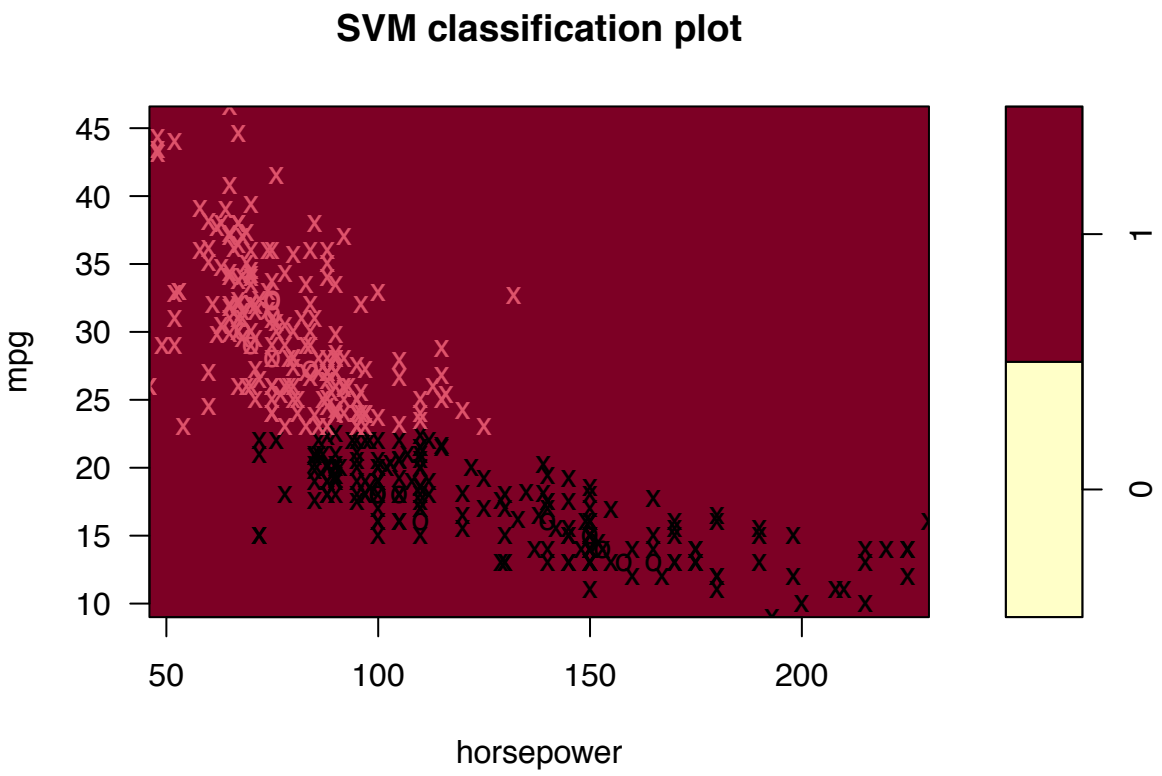

```
svmLinear <- svm(gasMedian ~ ., data = newAuto, kernel = "linear", cost = 0.01)
plot(svmLinear, Auto, mpg ~ horsepower)
```



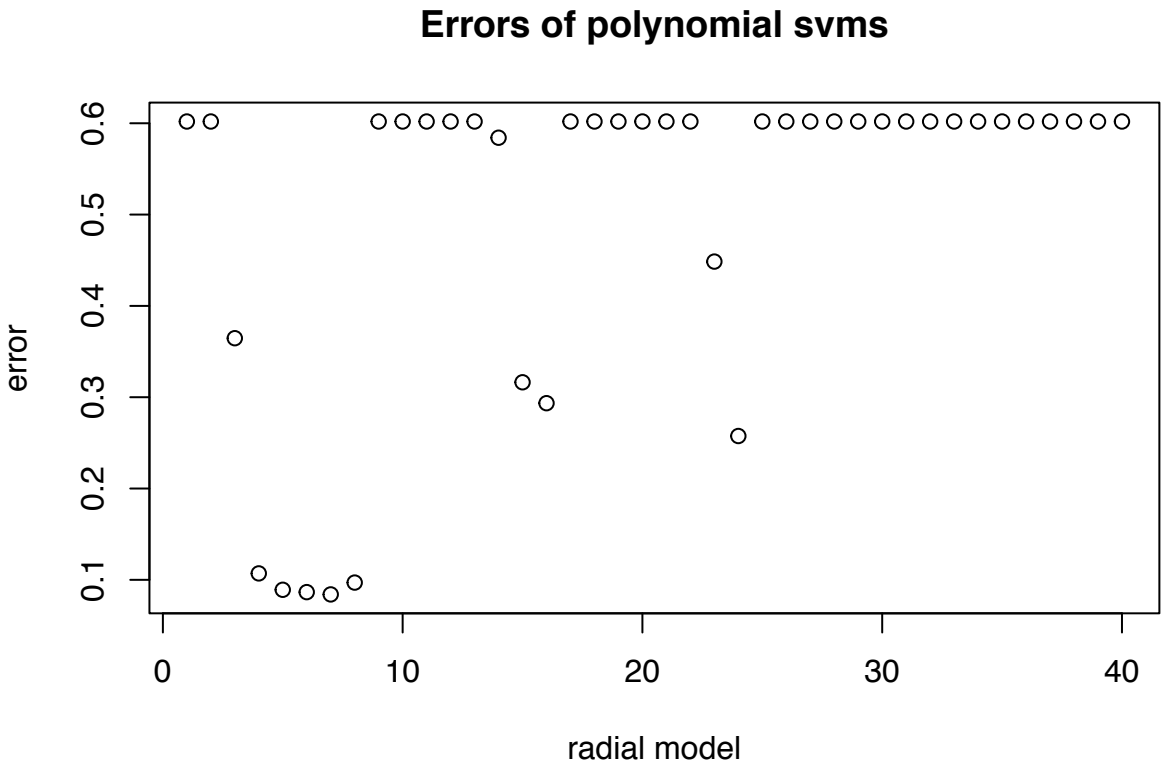

```
plot(y =  tune_outRadial$performances$error, x = 1:length(tune_outRadial$performances$error), ylab = "e
```



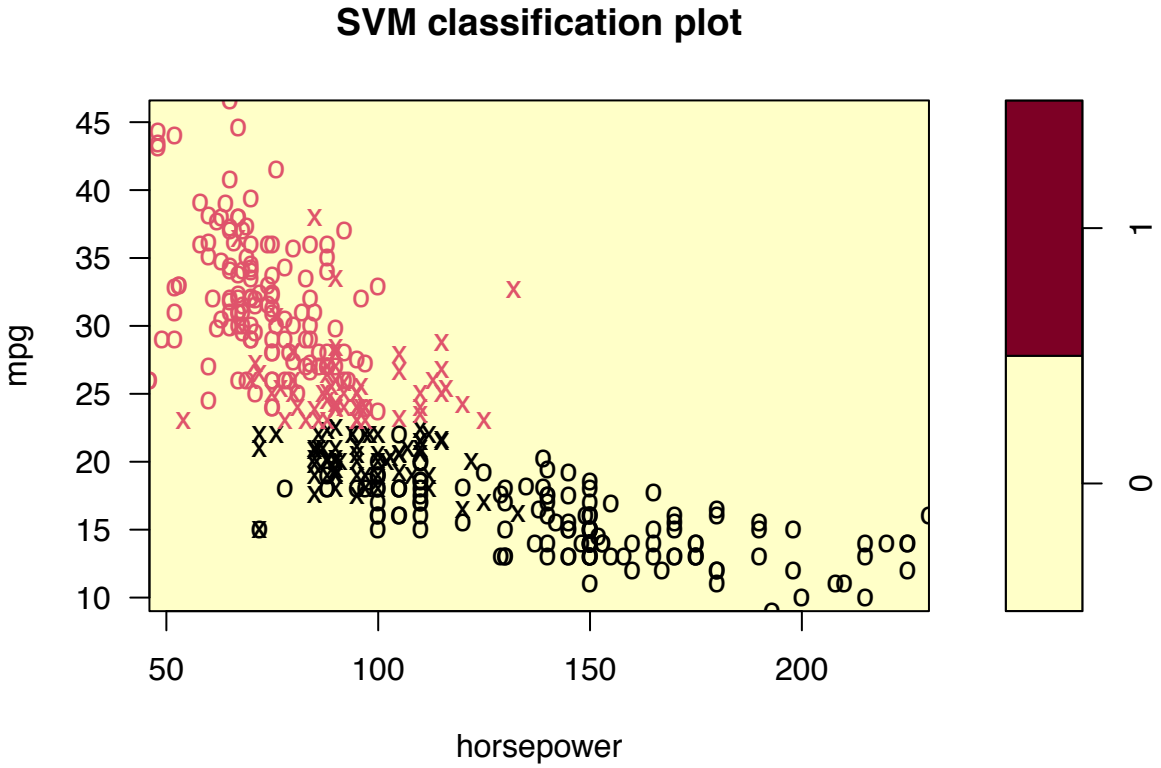

```
svmRadial <- svm(gasMedian~., data = newAuto, kernel = "radial", cost = 1, gamma = 1)
plot(svmRadial, Auto, mpg ~ horsepower)
```




```
plot(y =  tune_outPoly$performances$error, x = 1:length(tune_outPoly$performances$error), ylab = "error
```




```
svmPoly = svm(gasMedian~., data = newAuto, kernel = "polynomial", cost = 10, degree = 1)
plot(svmPoly, Auto, mpg~horsepower)
```


SVM classification plot
mpg
horsepower
45
40
35
30
25
20
15
10
50
100
150
200
1
0

HW7_Q2

2022-05-04

8. 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.

```
library(ISLR2)
n_train = 800

n <- dim(OJ)[1]

train_idx = sample(1:n, n_train)

test_idx = (1:n)[-train_idx]

n_test = length(test_idx)
```

(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 = OJ, kernel = "linear", cost = 0.01)
print(summary(svm.fit))
```

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

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

```
# Train
OJ_train = OJ[train_idx,]
y_hat = predict(svm.fit, newdata = OJ_train)
print(table(predicted = y_hat, truth = OJ_train$Purchase))
```

```
##          truth
## predicted  CH  MM
##        CH 418  77
##        MM  64 241
```

```
train_error = 1 - sum(y_hat == OJ_train$Purchase)/n_test
print(paste("Train Error:", train_error, sep=" "))
```

```
## [1] "Train Error: -1.44074074074074"
```

```
# Test
OJ_test = OJ[test_idx,]
y_hat = predict(svm.fit, newdata = OJ_test)
print(table(predicted = y_hat, truth = OJ_test$Purchase))
```

```
##          truth
## predicted  CH  MM
##        CH 158  23
##        MM  13  76
```

```
test_error = 1 - sum(y_hat == OJ_test$Purchase)/n_test
print(paste("Test Error:", test_error, sep=" "))
```

```
## [1] "Test Error: 0.133333333333333"
```

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

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

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     5
##
## - best performance: 0.164486
##
## - Detailed performance results:
##    cost     error dispersion
## 1  0.01 0.1738318 0.04624883
## 2  0.10 0.1757009 0.04679124
## 3  1.00 0.1672897 0.03877222
## 4  5.00 0.1644860 0.03969977
## 5 10.00 0.1672897 0.03724012
```

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

```
# Train
OJ_train = OJ[train_idx,]

y_hat = predict(tune$best.model, newdata = OJ_train)

print(table(predicted = y_hat, truth = OJ_train$Purchase))
```

```
##          truth
## predicted  CH  MM
##        CH 420  75
##        MM  62 243
```

```
train_error = 1 - sum(y_hat == OJ_train$Purchase)/n_test

print(paste("Train Error:", train_error, sep=" "))
```

```
## [1] "Train Error: -1.45555555555556"
```

```
# Test
OJ_test = OJ[test_idx,]

y_hat = predict(tune$best.model, newdata = OJ_test)

print(table(predicted = y_hat, truth = OJ_test$Purchase))
```

```
##          truth
## predicted  CH  MM
##        CH 158  23
##        MM  13  76
```

```
test_error = 1 - sum(y_hat == OJ_test$Purchase)/n_test

print(paste("Test Error:", test_error, sep=" "))
```

```
## [1] "Test Error: 0.133333333333333"
```

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

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

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ, kernel = "radial", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##
## Number of Support Vectors:  837
##
##  ( 420 417 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
# Train
OJ_train = OJ[train_idx,]

y_hat = predict(svm.fit, newdata = OJ_train)

print(table(predicted = y_hat, truth = OJ_train$Purchase))
```

```
##          truth
## predicted  CH  MM
##        CH 482 318
##        MM   0   0
```

```
train_error = 1 - sum(y_hat == OJ_train$Purchase)/n_test

print(paste("Train Error:", train_error, sep=" "))
```

```
## [1] "Train Error: -0.785185185185185"
```

```r
# Test
OJ_test = OJ[test_idx,]

y_hat = predict(svm.fit, newdata = OJ_test)

print(table(predicted = y_hat, truth = OJ_test$Purchase))
```

```
##          truth
## predicted  CH  MM
##        CH 171  99
##        MM   0   0
```

```r
test_error = 1 - sum(y_hat == OJ_test$Purchase)/n_test

print(paste("Test Error:", test_error, sep=" "))
```

```
## [1] "Test Error: 0.366666666666667"
```

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

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.1682243
##
## - Detailed performance results:
##    cost     error dispersion
## 1  0.01 0.3897196 0.04985397
## 2  0.10 0.1766355 0.03775773
## 3  1.00 0.1682243 0.03496876
## 4  5.00 0.1738318 0.04042649
## 5 10.00 0.1822430 0.04231488
```

```r
# Train
OJ_train = OJ[train_idx,]

y_hat = predict(tune$best.model, newdata = OJ_train)

print(table(predicted = y_hat, truth = OJ_train$Purchase))
```

```
##          truth
## predicted  CH  MM
##        CH 435  85
##        MM  47 233
```

```
train_error = 1 - sum(y_hat == OJ_train$Purchase)/n_test

print(paste("Train Error:", train_error, sep=" "))
```

```
## [1] "Train Error: -1.47407407407407"
```

```
# Test
OJ_test = OJ[test_idx,]

y_hat = predict(tune$best.model, newdata = OJ_test)

print(table(predicted = y_hat, truth = OJ_test$Purchase))
```

```
##          truth
## predicted  CH  MM
##        CH 162  23
##        MM   9  76
```

```
test_error = 1 - sum(y_hat == OJ_test$Purchase)/n_test

print(paste("Test Error:", test_error, sep=" "))
```

```
## [1] "Test Error: 0.118518518518518"
```

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

```
library(e1071)
svm.fit <- svm(Purchase ~ ., data = OJ, kernel = "polynomial", cost = 0.01, degree = 2)
print(summary(svm.fit))
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ, 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:  839
##
##  ( 422 417 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
# Train
OJ_train = OJ[train_idx,]

y_hat = predict(svm.fit, newdata = OJ_train)

print(table(predicted = y_hat, truth = OJ_train$Purchase))
```

```
##          truth
## predicted  CH  MM
##        CH 476 297
##        MM   6  21
```

```
train_error = 1 - sum(y_hat == OJ_train$Purchase)/n_test

print(paste("Train Error:", train_error, sep=" "))
```

```
## [1] "Train Error: -0.840740740740741"
```

```
# Test
OJ_test = OJ[test_idx,]

y_hat = predict(svm.fit, newdata = OJ_test)

print(table(predicted = y_hat, truth = OJ_test$Purchase))
```

```
##          truth
## predicted  CH  MM
##        CH 170  91
##        MM   1   8
```

```
test_error = 1 - sum(y_hat == OJ_test$Purchase)/n_test

print(paste("Test Error:", test_error, sep=" "))
```

```
## [1] "Test Error: 0.340740740740741"
```

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

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     5
##
## - best performance: 0.1794393
##
## - Detailed performance results:
##    cost     error dispersion
## 1  0.01 0.3691589 0.05594481
## 2  0.10 0.2990654 0.04061808
## 3  1.00 0.1925234 0.03121489
## 4  5.00 0.1794393 0.03013928
## 5 10.00 0.1813084 0.02994546
```

```
# Train
OJ_train = OJ[train_idx,]

y_hat = predict(tune$best.model, newdata = OJ_train)

print(table(predicted = y_hat, truth = OJ_train$Purchase))
```

```
##          truth
## predicted  CH  MM
##        CH 437  90
##        MM  45 228
```

```
train_error = 1 - sum(y_hat == OJ_train$Purchase)/n_test

print(paste("Train Error:", train_error, sep=" "))
```

```
## [1] "Train Error: -1.46296296296296"
```

```
# Test
OJ_test = OJ[test_idx,]

y_hat = predict(tune$best.model, newdata = OJ_test)

print(table(predicted = y_hat, truth = OJ_test$Purchase))
```

```
##          truth
## predicted  CH  MM
##        CH 162  22
##        MM   9  77
```

```
test_error = 1 - sum(y_hat == OJ_test$Purchase)/n_test

print(paste("Test Error:", test_error, sep=" "))
```

```
## [1] "Test Error: 0.114814814814815"
```

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

For the linear cost-optimized model: [1] "Train Error: -1.45925925925926" [1] "Test Error: 0.133333333333333"

For the radial cost-optimized model: [1] "Train Error: -1.49259259259259" [1] "Test Error: 0.137037037037037"

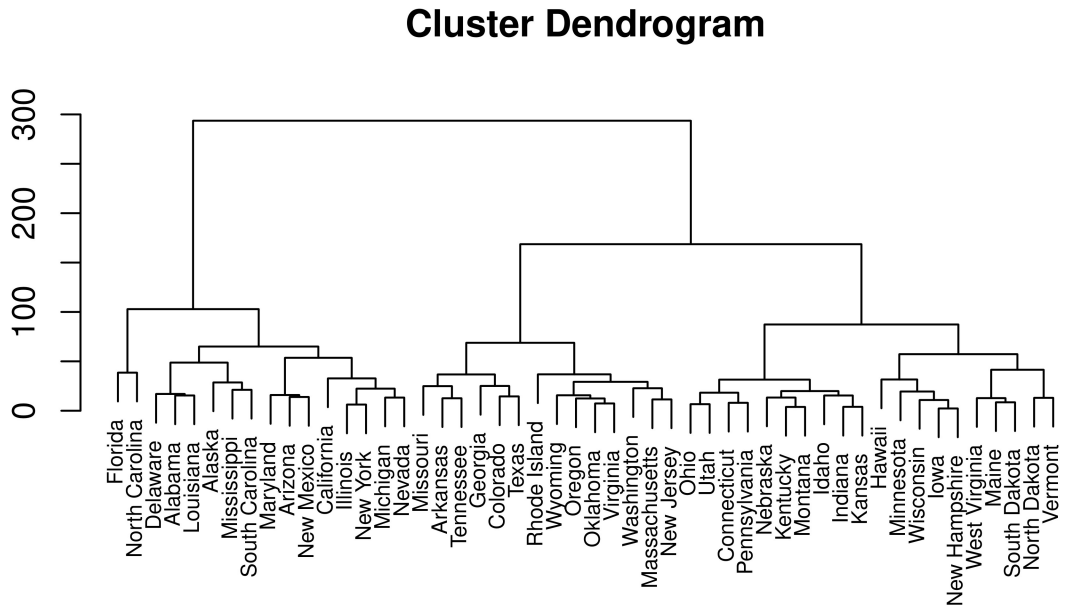
For the polynomial cost-optimized model: [1] "Train Error: -1.4962962962963" [1] "Test Error: 0.148148148148148"

From these errors, we can assume that the linear model performed the best becuase the test error was the lowest.

HW7_Q9

9a. Consider the USArrests data. We will now perform hierarchical clustering on the states.

```
data = USArrests
set.seed(1)

cluster = hclust(dist(data, method = "euclidean"), method = "complete")
plot(cluster, cex = 0.65, xlab = "", ylab = "", sub="")
```



9b. Cut the dendrogram at a height that results in three distinct clusters. Which states belong to which clusters?

```
clusters3 = cutree(cluster, 3)
clusters3
```

```
##        Alabama         Alaska        Arizona       Arkansas     California
```

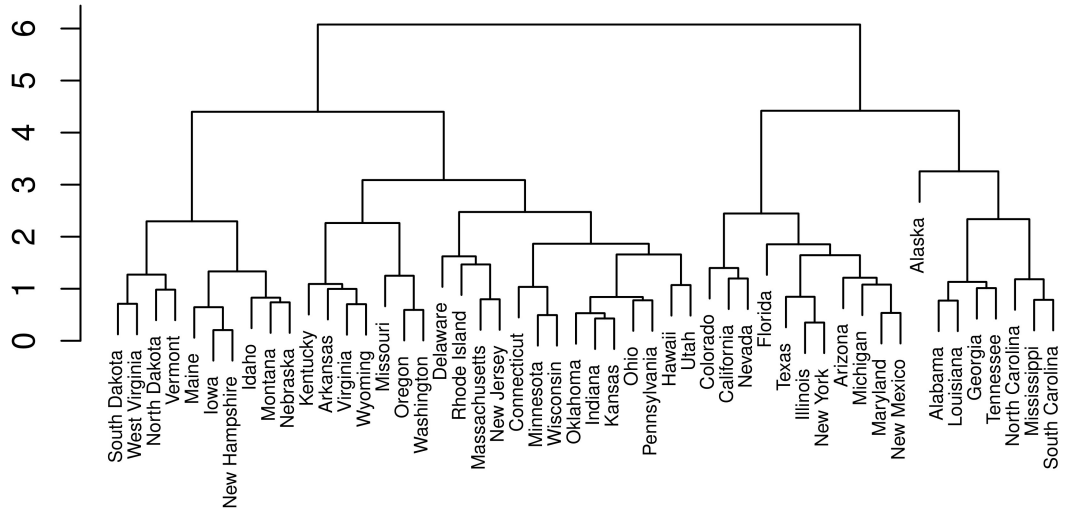
```
##              1              1              1              2              1
##       Colorado    Connecticut       Delaware        Florida        Georgia
##              2              3              1              1              2
##         Hawaii          Idaho       Illinois        Indiana           Iowa
##              3              3              1              3              3
##         Kansas       Kentucky      Louisiana          Maine       Maryland
##              3              3              1              3              1
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri
##              2              1              3              1              2
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey
##              3              3              1              3              2
##     New Mexico       New York North Carolina   North Dakota           Ohio
##              1              1              1              3              3
##       Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina
##              2              2              3              2              1
##   South Dakota      Tennessee          Texas           Utah        Vermont
##              3              2              2              3              3
##       Virginia     Washington  West Virginia      Wisconsin        Wyoming
##              2              2              3              3              2
```

```
table(clusters3)
```

```
## clusters3
##  1  2  3
## 16 14 20
```

9c. Hierarchically cluster the states using complete linkage and Euclidean distance, after scaling the variables to have standard deviation one.

```
scaled = scale(USArrests)
set.seed(1)

cluster_scaled = hclust(dist(scaled, method = "euclidean"), method = "complete")
plot(cluster_scaled, cex = 0.6, xlab = "", ylab = "", sub="")
```




```
clusters3_scaled = cutree(cluster_scaled, 3)
clusters3_scaled
```

```
##        Alabama         Alaska        Arizona       Arkansas     California
##              1              1              2              3              2
##       Colorado    Connecticut       Delaware        Florida        Georgia
##              2              3              3              2              1
##         Hawaii          Idaho       Illinois        Indiana           Iowa
##              3              3              2              3              3
##         Kansas       Kentucky      Louisiana          Maine       Maryland
##              3              3              1              3              2
##  Massachusetts       Michigan      Minnesota    Mississippi       Missouri
##              3              2              3              1              3
##        Montana       Nebraska         Nevada  New Hampshire     New Jersey
##              3              3              2              3              3
##     New Mexico       New York North Carolina   North Dakota           Ohio
##              2              2              1              3              3
##       Oklahoma         Oregon   Pennsylvania   Rhode Island South Carolina
##              3              3              3              3              1
##   South Dakota      Tennessee          Texas           Utah        Vermont
##              3              1              2              3              3
##       Virginia     Washington  West Virginia      Wisconsin        Wyoming
##              3              3              3              3              3
```

```
table(clusters3_scaled)
```

```
## clusters3_scaled
##  1  2  3
##  8 11 31
```

9d. What effect does scaling the variables have on the hierarchical clustering obtained? In your opinion, should the variables be scaled before the inter-observation dissimilarities are computed? Provide a justification for your answer.

Scaling the variables so that the standard deviation = 1, caused a shift in which states were assigned to which cluster. Previously, the states were separated 16, 14, 20 in clusters 1, 2, 3 respectively, whereas now it is at 8, 11, 31. For this dataset, there was increase in dissimilarity.

HW 7 Q4

Ben Howell

3/24/2022

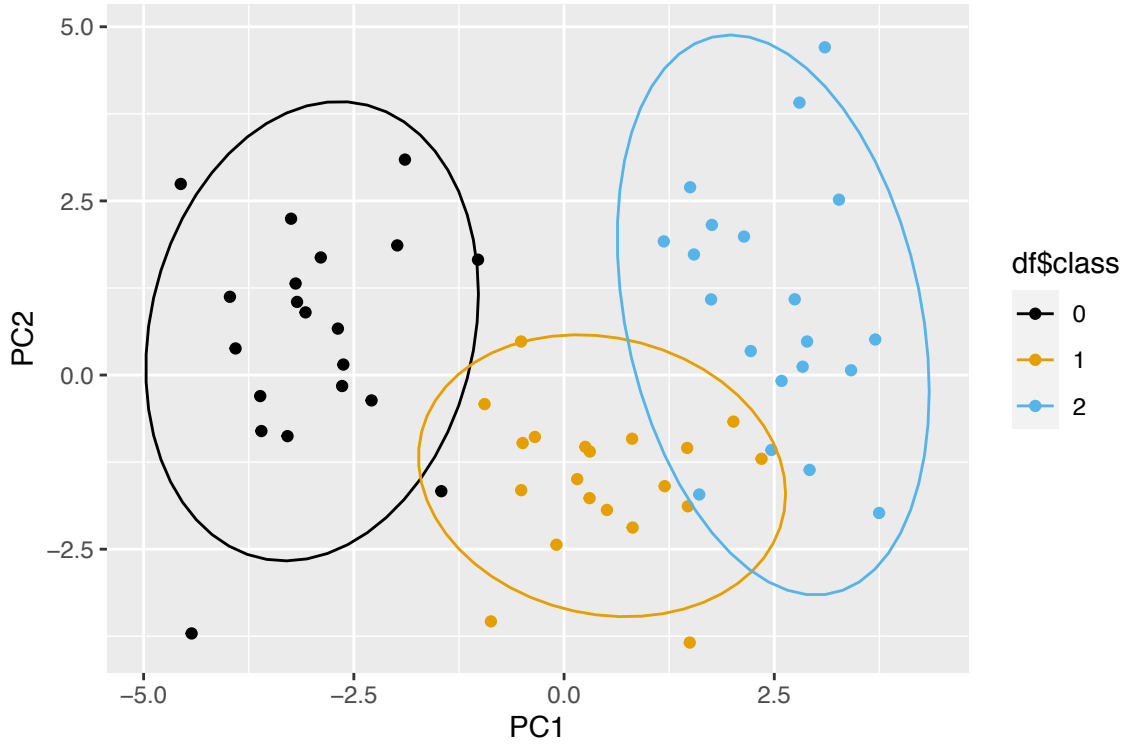
A)

```
set.seed(123)

require(tidyverse)
require(janitor)
require(MLmetrics)
require(LICORS)

df <- data.frame(replicate(50, rnorm(20, mean = rnorm(1, mean = 0), sd = 3))) %>%
    rbind(data.frame(replicate(50, rnorm(20, mean = rnorm(1, mean = 1), sd = 3)))) %>%
    rbind(data.frame(replicate(50, rnorm(20, mean = rnorm(1, mean = 2), sd = 3)))) %>%
    clean_names() %>%
    dplyr::mutate(class = ifelse(row_number() <= 20, "0",
                                 ifelse(row_number() > 20 & row_number() <= 40, "1", "2")))

res <- prcomp(df %>%
         dplyr::select(-c(class)),
       scale. = TRUE)

head(res$x %>%
       data.frame() %>%
       dplyr::select(PC1:PC10))
```

```
##         PC1        PC2         PC3         PC4        PC5        PC6
## 1 -3.599741 -0.8040346 -0.14826007  1.19002268  2.5687074 -1.9255866
## 2 -1.459633 -1.6674755 -1.01988783  1.99040039 -1.6554270 -0.6138785
## 3 -3.175022  1.0494639 -0.11562471  0.41592368 -0.1280092 -0.8203033
## 4 -3.074665  0.8994304  3.24555837  0.50422037  0.8655841  1.2432481
## 5 -2.625246  0.1506271  1.05989162  0.04765814  1.4004633 -0.4352534
## 6 -3.246501  2.2443454  0.08463199 -1.51525747 -1.5802781  2.0304992
##           PC7        PC8        PC9       PC10
## 1 -0.13123134  0.8183368  0.2714540 -1.4578003
## 2  3.30119495 -2.8426974 -2.5388748  1.4410693
## 3 -1.21414831 -0.0442793  0.7460123 -0.6239612
## 4 -1.82899598  2.5445949 -0.5808259 -0.5326450
## 5  1.33139235 -1.4813495  2.0233303  1.6077639
## 6 -0.08079525  1.2605349 -0.1320301 -1.8497137
```

B)

```
res$x %>%
  data.frame() %>%
  ggplot() +
  geom_point(aes(x = PC1, y = PC2, color = df$class)) +
  stat_ellipse(aes(x = PC1, y = PC2, color = df$class)) +
  ggthemes::scale_color_colorblind()
```



C)

```
km <- kmeans(df %>%
               dplyr::select(-c(class)),
             3)

print(table(df$class, km$cluster))
```

```
##
##       1  2  3
##    0  0 19  1
##    1  3  0 17
##    2 20  0  0
```

The K-Means clustering does a good job of differentiating between the classes that we created. We see that Clusters 0 and 2 got most of their observations assigned to their own cluster, but some of Cluster 1 got split, which will be interesting to keep an eye on. From looking at the plot in part B, that's not super surprising.

D)

```
km <- kmeans(df %>%
               dplyr::select(-c(class)),
             2)

print(table(df$class, km$cluster))
```

```
##
##      1  2
##   0  0 20
##   1 10 10
##   2 20  0
```

With `K = 2`, we see that our classes 0 and 2 are distinctly their own thing, while class 1 gets split 50/50 between those two classes. Again, this makes sense from looking at the way the PCA vectors came out.

E)

```
km <- kmeans(df %>%
               dplyr::select(-c(class)),
             4)

print(table(df$class, km$cluster))
```

```
##
##      1  2  3  4
##   0  0 20  0  0
##   1  2  0 14  4
##   2 15  0  0  5
```

`K = 4` starts to return some interesting results where we begin to see some separation and dispersion among the classes. Class 0 remains undefeated in being its own cluster, but the other two classes get spread more. It's interesting to see that classes 1 got split across 3 different clusters which kinda indicated how that data generated was more dispered itself.

F)

```
km <- kmeans(
  res$x %>%
    data.frame() %>%
    dplyr::select(PC1, PC2),
  3
)

table(df$class, km$cluster)
```

```
##
##      1  2  3
##   0  1 19  0
##   1 20  0  0
##   2  4  0 16
```

It's interesting to see that this approach accuractely classified 92% of the observations, barely below the 93% of the first `K = 3` classifier that we ran in part C. Being able to get that close while using just two inputs for each variable, rather than 50 is quite the improvement and shows how the PCA vectors improved our clustering.

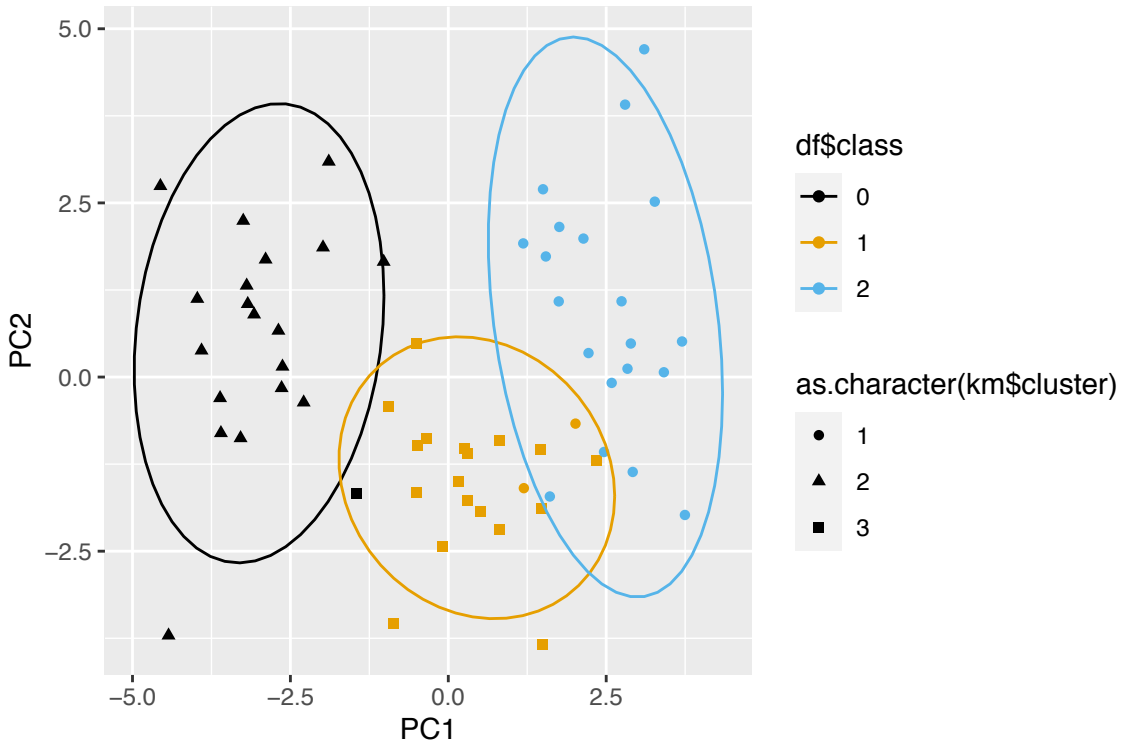
G)

```
km <- scale(df %>%
               dplyr::select(-c(class))) %>%
  data.frame() %>%
  kmeans(
    centers = 3
  )

table(df$class, km$cluster)
```

```
##
##      1  2  3
##   0  0 19  1
##   1  2  0 18
##   2 20  0  0
```

Interestingly, once we scale the data, we see the most accurate of our `K = 3` classifiers, predicting 95% of the classes properly, which is good to see. Putting everything onto a consistent scale helped out the quality of our data and allowed the model to improve it's ability to classify our generated classes. On the diagnostic plot below, the observations that are mis-classified make sense as they are outliers among their generated class and almost fit within the elipse of another cluster.

```
res$x %>%
  data.frame() %>%
  ggplot() +
  geom_point(aes(x = PC1, y = PC2, color = df$class, shape = as.character(km$cluster))) +
  stat_ellipse(aes(x = PC1, y = PC2, color = df$class)) +
  ggthemes::scale_color_colorblind()
```


5.0
2.5
0.0
−2.5
PC2
−5.0
−2.5
0.0
2.5
PC1
df$class
0
1
2
as.character(km$cluster)
1
2
3