ANLY512_HW10

Hongyang Zheng

2019/4/24

```r
library(ISLR)
library(mlbench)
library(dbscan)
library(deldir)
```

```
## Warning: package 'deldir' was built under R version 3.5.2
```

```
## deldir 0.1-16
```

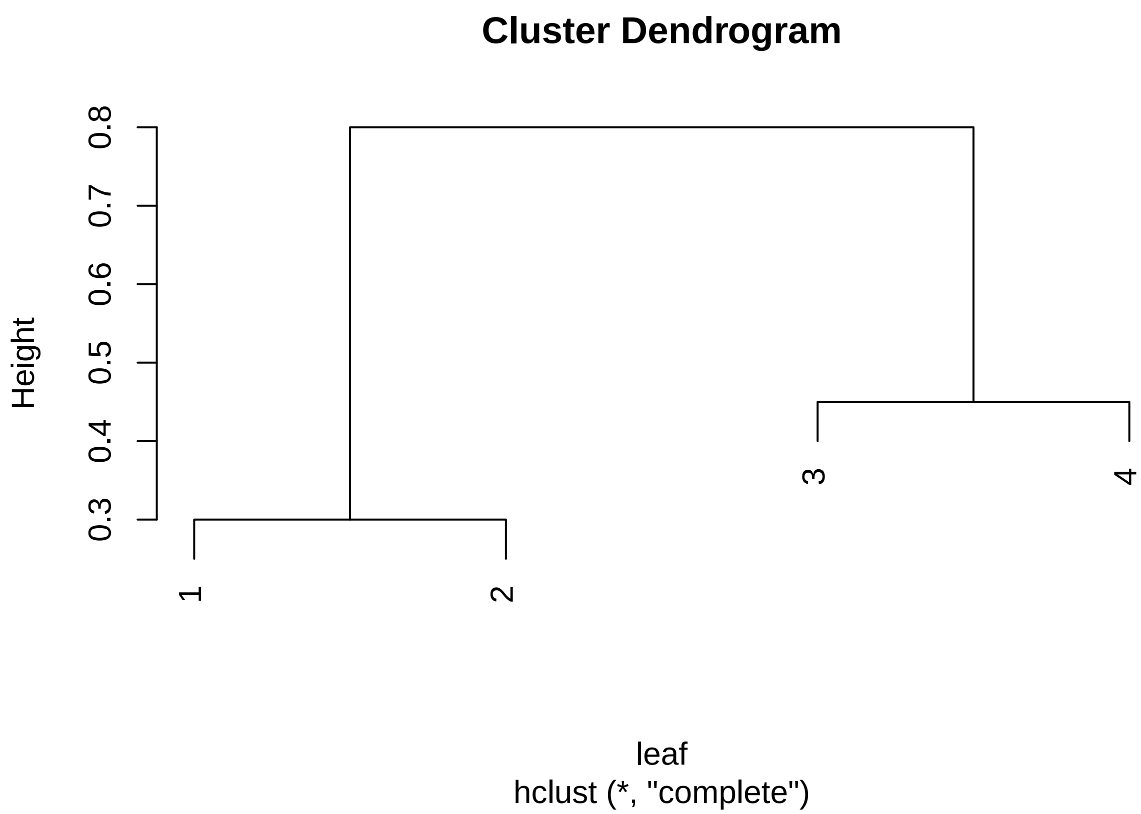
```r
library(RnavGraphImageData)
library(cluster)
```

```
## Warning: package 'cluster' was built under R version 3.5.2
```

Problem #2

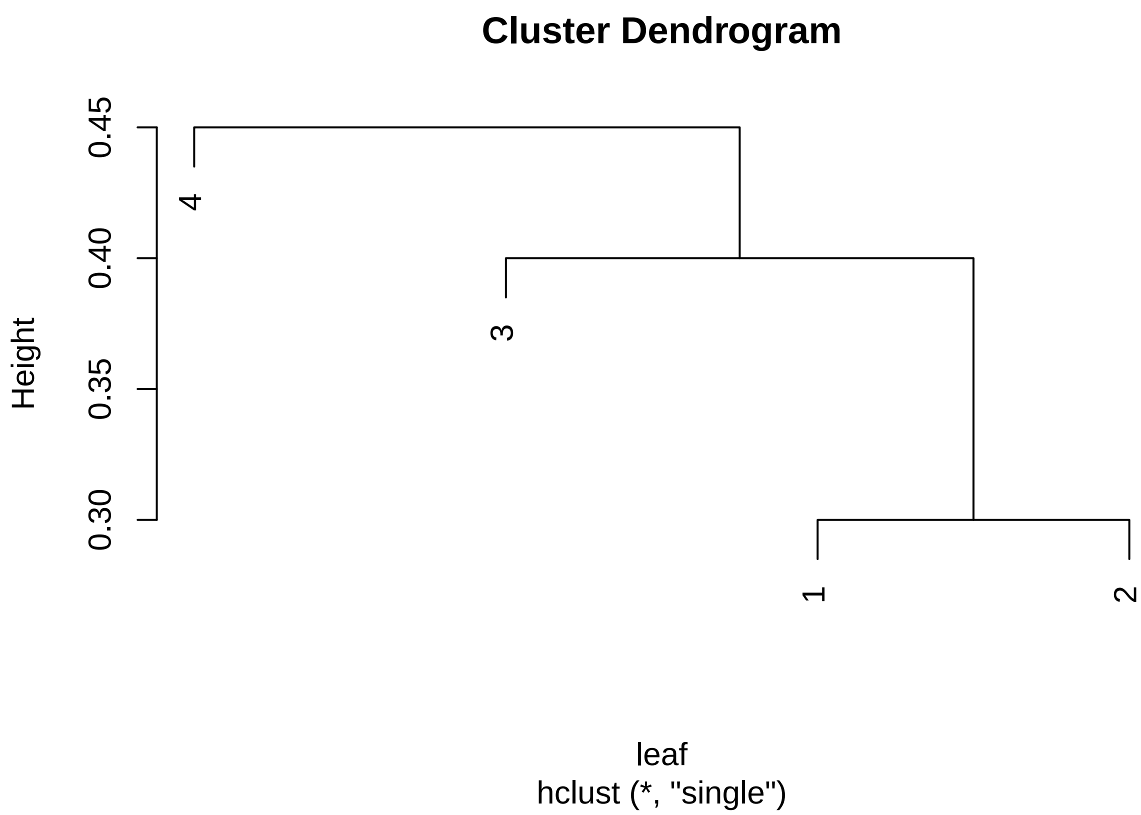
a)

```r
dissimilarity = as.dist(matrix(c(0, 0.3, 0.4, 0.7,
                                 0.3, 0, 0.5, 0.8,
                                 0.4, 0.5, 0.0, 0.45,
                                 0.7, 0.8, 0.45, 0.0), nrow=4))

# Make a plot using complete linkage
plot(hclust(dissimilarity, method="complete"), xlab='leaf')
```



b)

```r
# Make a plot using single linkage
plot(hclust(dissimilarity, method="single"), xlab='leaf')
```



c)

Cluster one: 1, 2

Cluster two: 3, 4

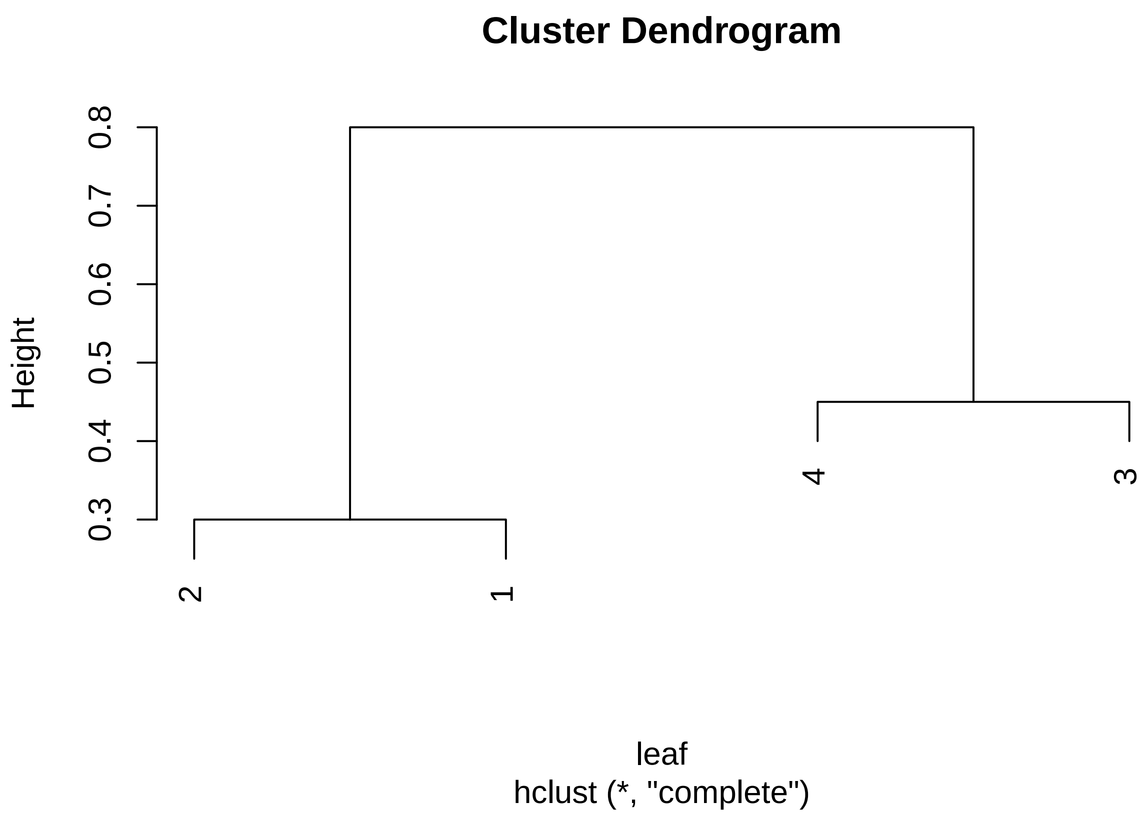
d)

Cluster one: 4

Cluster two: 1, 2, 3

e)

```
plot(hclust(dissimilarity, method="complete"), labels=c(2,1,4,3), xlab='leaf')
```

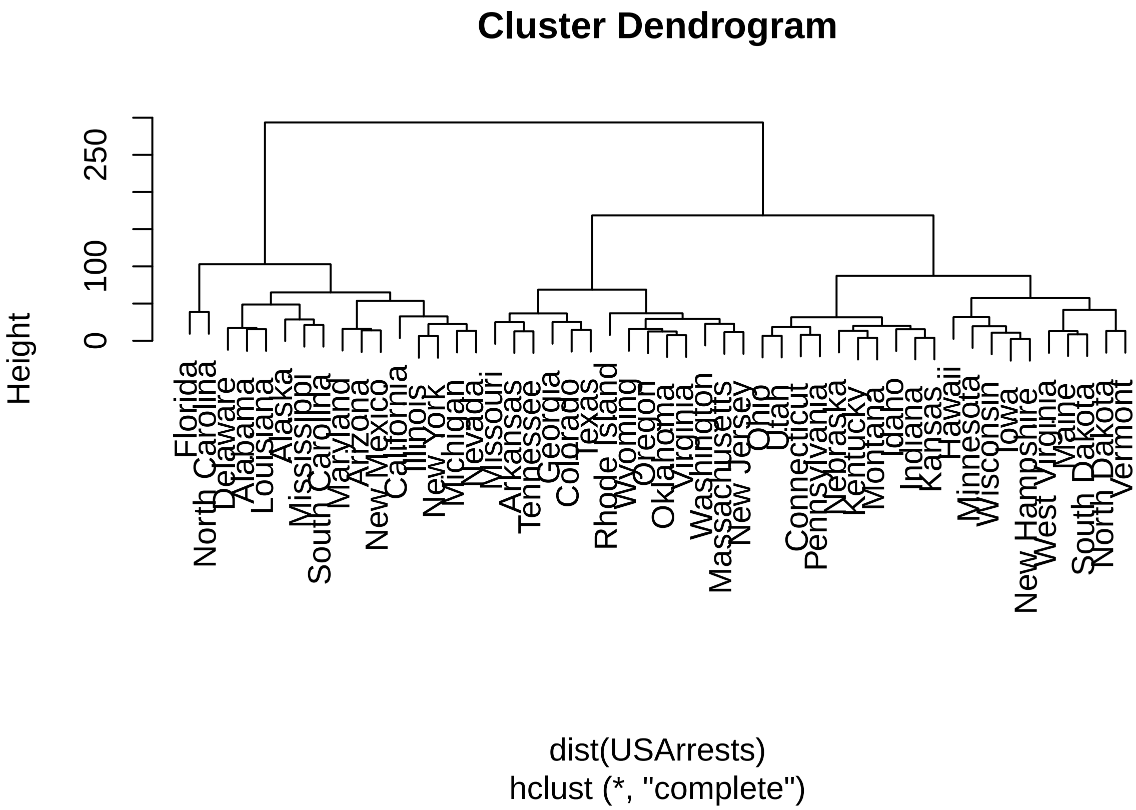


Since 1, 2 are in the same cluster; 3, 4 are in the same cluster, then if we switch the label of leaves in the same cluster, the meaning of the dendrogram does not change.

Problem #9

a)

```r
set.seed(1234)

# Build cluster
hc.cp = hclust(dist(USArrests), method="complete")

# Make a plot
plot(hc.cp)
```



b)

```r
# Cut the cluster
cut.hc=cutree(hc.cp, 3)

# See the results
print(cut.hc)
```

```
##        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
```

```r
table(cut.hc)
```

```
## cut.hc
##  1  2  3
## 16 14 20
```

There are 16 states belong to cluster 1, 14 states belong to cluster 2, 20 states belong to cluster 3.

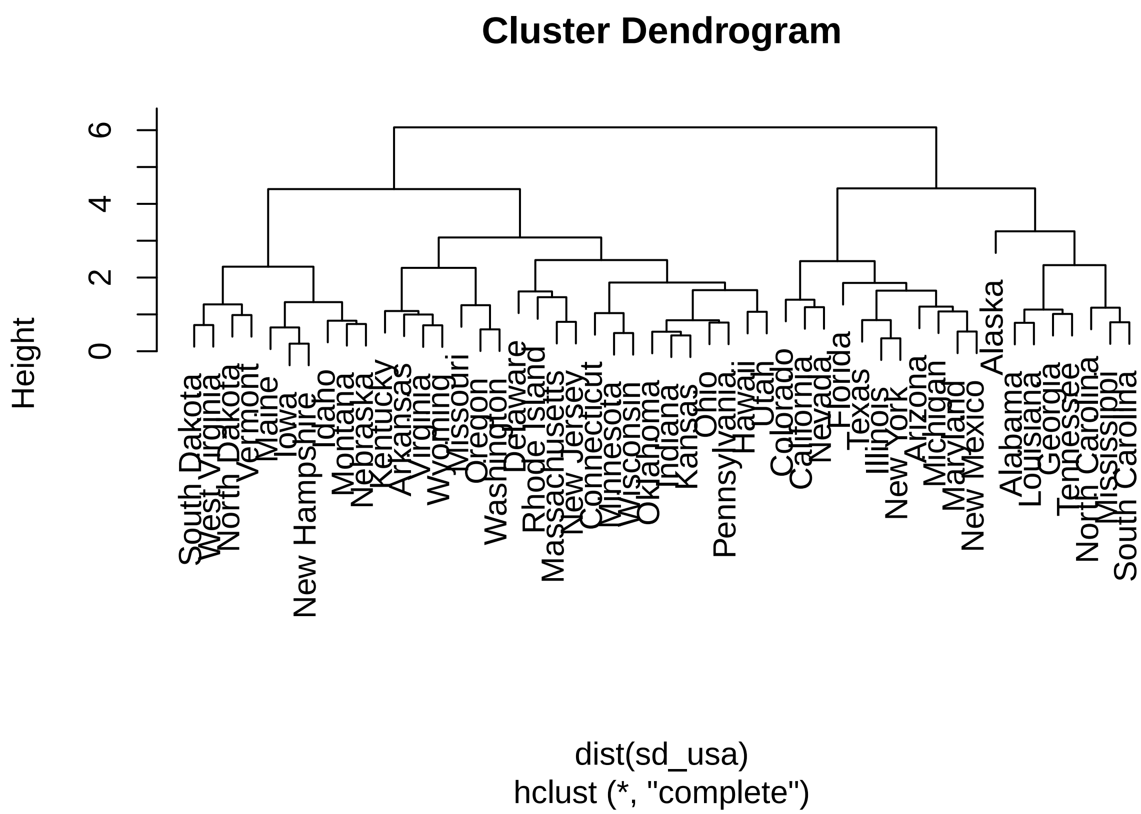
Cluster 1: Alabama, Alaska, Arizona, California, Delaware, Florida, Illinois, Louisiana, Maryland, Michigan, Mississippi, Nevada, New Mexico, New York, North Carolina, South Carolina

Cluster 2: Arkansas, Colorado, Georgia, Massachusetts, Missouri, New Jersey, Oklahoma, Oregon, Rhode Island, Tennessee, Texas, Virginia, Washington, Wyoming

Cluster 3: Connecticut, Hawaii, Idaho, Indiana, Iowa, Kansas, Kentucky, Maine, Minnesota, Montana, Nebraska, New Hampshire, North Dakota, Ohio, Pennsylvania, South Dakota, Utah, Vermont, West Virginia, Wisconsin

c)

```r
# Scale the data
sd_usa = scale(USArrests)

# Build a cluster
sd.hc.cp = hclust(dist(sd_usa), method="complete")

# Plot the cluster
plot(sd.hc.cp)
```



d)

```r
# Cut the cluster again
sd.cut.hc=cutree(sd.hc.cp, 3)

# See the results
print(sd.cut.hc)
```

```
##        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
```

```r
table(sd.cut.hc)
```

```
## sd.cut.hc
##  1  2  3
##  8 11 31
```

```r
# Compare
table(sd.cut.hc, cut.hc)
```

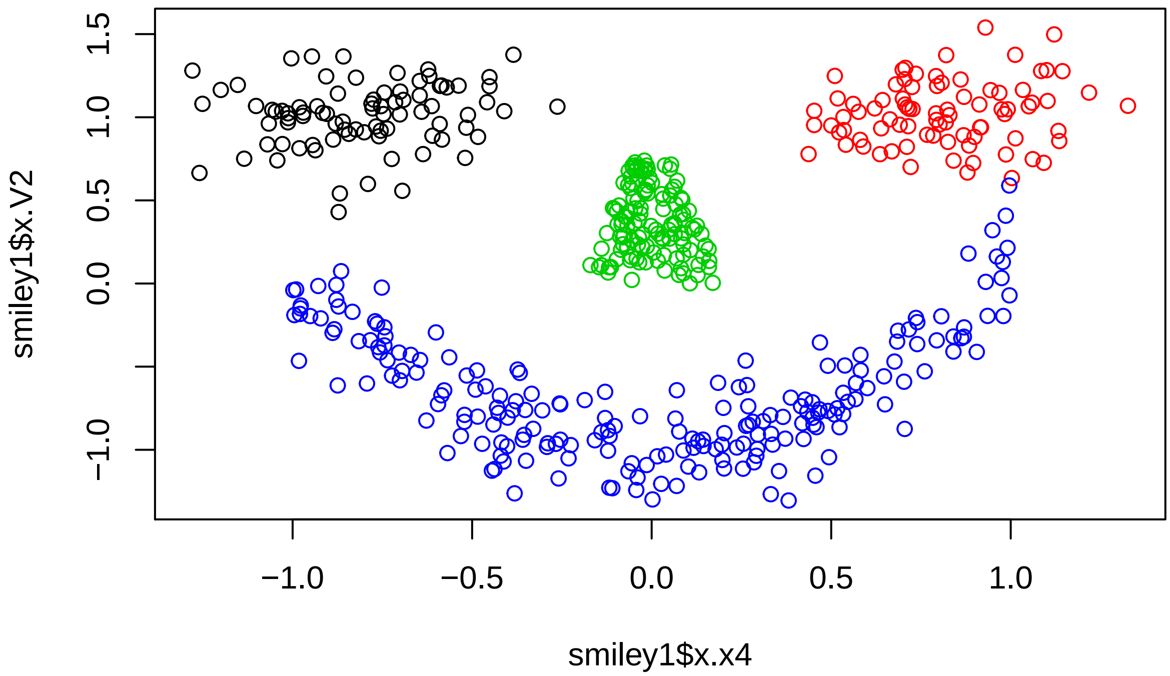
```
##          cut.hc
## sd.cut.hc  1  2  3
##         1  6  2  0
##         2  9  2  0
##         3  1 10 20
```

From part a) and part c) graphs we can see that the max height of the dendogram is impacted by the scaling, but the bushiness is not impacted(at least looks similar). By cutting both clustering to 3 clusters we can see that the results are not same. I think for this dataset, we should scale the data before doing clustering analysis since the data measured in different units.

Problem Extra #67

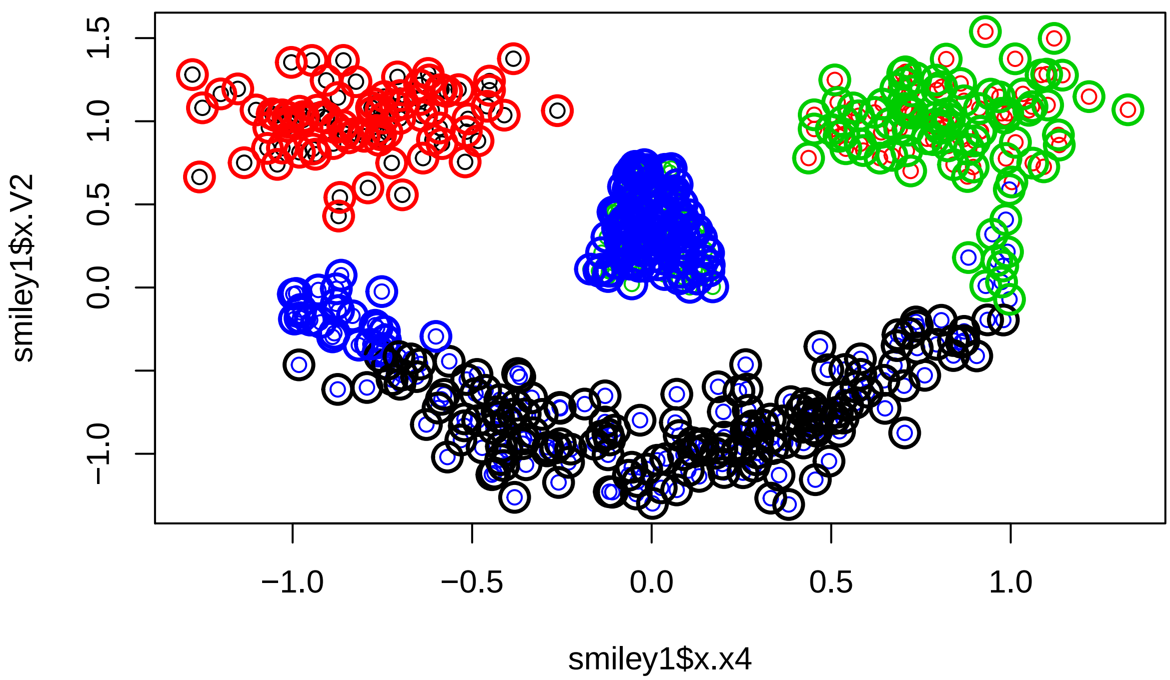
```r
set.seed(120)

# Generate data
smiley1=mlbench.smiley(n=500, sd1 = 0.2, sd2 = 0.2)
```

```r
smiley1=as.data.frame(smiley1)
plot(smiley1$x.x4, smiley1$x.V2, col=smiley1$classes)
```



a)

```r
# Build clusters
try.1=kmeans(smiley1[,1:2], 4, nstart=20)
plot(smiley1$x.x4, smiley1$x.V2, col=smiley1$classes, lwd=1)
points(smiley1$x.x4, smiley1$x.V2, col=try.1$cluster, pch=1, cex=2, lwd = 2)
```



From the graph, we can see that the k-means cannot form four clusters recovering the four original clusters exactly, since the mouth is splited into two different clusters.

```r
set.seed(1234)
# Build another clusters
try.2=kmeans(smiley1[, 1:2], 4, nstart=20)

# Make a confusion matrix
table(smiley1$classes)
```

```
##
##   1   2   3   4
##  83  83 125 209
```

```r
table(try.2$cluster)
```

```
##
##   1   2   3   4
##  93 151  83 173
```

```r
table(smiley1$classes, try.2$cluster)
```

```
##
##       1   2   3   4
##   1   0   0  83   0
##   2  83   0   0   0
##   3   0 125   0   0
##   4  10  26   0 173
```

From the confusion matrix we know that there are 5 groups of points, which means the k-means clusters cannot cover the original points exactly.

b)

```r
# See the distribution of each class for original data
table(smiley1$classes)
```

```
##
##   1   2   3   4
##  83  83 125 209
```

```r
# Build hierarchical clusters using complete linkage
try.3=hclust(dist(smiley1[, 1:2]),method= "complete")
cut.1=cutree(try.3, 4)
table(smiley1$classes, cut.1)
```

```
##    cut.1
##       1   2   3   4
##   1  83   0   0   0
##   2   0  67  16   0
##   3   0   0 125   0
##   4   0   0  26 183
```

```r
# Build hierarchical clusters using single linkage
try.4=hclust(dist(smiley1[, 1:2]),method= "single")
cut.2=cutree(try.4, 4)
table(smiley1$classes, cut.2)
```

```
##    cut.2
##       1   2   3   4
```

```
##   1  83   0   0   0
##   2   0  82   1   0
##   3   0   0   0 125
##   4   0 209   0   0
```

```
# Build hierarchical clusters using average linkage
try.5=hclust(dist(smiley1[,1:2]),method= "average")
cut.3=cutree(try.5, 4)
table(smiley1$classes, cut.3)
```

```
##    cut.3
##       1   2   3   4
##   1  83   0   0   0
##   2   0  83   0   0
##   3   0   0 125   0
##   4   0   1  34 174
```

From the confusion matrices we know that when using single linkage, we can seperate the four clusters more accurately (only one point is misclassified). However when using complete linkage or average linkage, we have more points are mislabeled.

Problem Extra #69

a)

```
set.seed(10086)
# Try different small sd
for (n in seq(0.01, 0.05, 0.01))
{
smiley=mlbench.smiley(n=500, sd1 = n, sd2 = n)
smiley.df=as.data.frame(smiley)
model=kmeans(smiley.df[, 1:2], 4, nstart=20)
print(table(smiley.df$classes))
print(table(smiley.df$classes, model$cluster))
}
```

```
##
##   1   2   3   4
##  83  83 125 209
##
##       1   2   3   4
##   1  83   0   0   0
##   2   0  83   0   0
##   3   0   0   0 125
##   4   0  13 172  24
##
##   1   2   3   4
##  83  83 125 209
##
##       1   2   3   4
##   1   0   0   0  83
##   2  83   0   0   0
##   3   0   0 125   0
##   4   0 172  34   3
##
```

```
##   1   2   3   4
##  83  83 125 209
##
##       1   2   3   4
##   1  83   0   0   0
##   2   0   0   0  83
##   3   0   0 125   0
##   4   0 179  20  10
##
##   1   2   3   4
##  83  83 125 209
##
##       1   2   3   4
##   1   0   0   0  83
##   2  83   0   0   0
##   3   0 125   0   0
##   4   0  24 185   0
##
##   1   2   3   4
##  83  83 125 209
##
##       1   2   3   4
##   1   0   0   0  83
##   2   0   0  83   0
##   3   0 125   0   0
##   4 175  27   7   0
```

From these five confusion matrices, we can see that most of the points are correct and the misclassified points are no more than 40. Therefore, when sd is small, we can get a good clustering.

b)

```r
# Try different big sd
for (n in seq(1, 1.5, 0.1))
{
smiley=mlbench.smiley(n=500, sd1 = n, sd2 = n)
smiley.df=as.data.frame(smiley)
model=kmeans(smiley.df[, 1:2], 4, nstart=20)
print(table(smiley.df$classes))
print(table(smiley.df$classes, model$cluster))
}
```

```
##
##   1   2   3   4
##  83  83 125 209
##
##       1   2   3   4
##   1   3  13  56  11
##   2   4  46   5  28
##   3   0   0   0 125
##   4 105   7  20  77
##
##   1   2   3   4
##  83  83 125 209
##
```

```
##       1   2   3   4
##   1   3  37  10  33
##   2   1  16  43  23
##   3   0   0   0 125
##   4 110   9  10  80
##
##   1   2   3   4
##  83  83 125 209
##
##       1   2   3   4
##   1  10   7  40  26
##   2  37   6  18  22
##   3   0   0   0 125
##   4   4 105  19  81
##
##   1   2   3   4
##  83  83 125 209
##
##       1   2   3   4
##   1  12  43   3  25
##   2  41  16   6  20
##   3   0   0   0 125
##   4  13  13  97  86
##
##   1   2   3   4
##  83  83 125 209
##
##       1   2   3   4
##   1  20  13  45   5
##   2  26  33  13  11
##   3 125   0   0   0
##   4 106  10   9  84
##
##   1   2   3   4
##  83  83 125 209
##
##       1   2   3   4
##   1  37  20  18   8
##   2  13  35  25  10
##   3   0   0 125   0
##   4  10   6  88 105
```

From these confusion matrices we can see that all models have error more or less, and the error happens across multiple clusters. So when sd becomes larger, the k-means cannot over the original points very well.

```r
set.seed(1234)
for (n in seq(0.5, 2, 0.5))
{
smiley=mlbench.smiley(n=500, sd1 = n, sd2 = n)
smiley.df=as.data.frame(smiley)
model=kmeans(smiley.df[, 1:2], 4, nstart=20)
print(table(smiley.df$classes))
print(table(smiley.df$classes, model$cluster))
}
```

```
##
##   1   2   3   4
##  83  83 125 209
##
##       1   2   3   4
##   1  73   6   4   0
##   2   3   8  72   0
##   3   0 125   0   0
##   4   3  45  11 150
##
##   1   2   3   4
##  83  83 125 209
##
##       1   2   3   4
##   1   2  39   7  35
##   2   2  13  57  11
##   3   0   0   0 125
##   4 106   5  11  87
##
##   1   2   3   4
##  83  83 125 209
##
##       1   2   3   4
##   1   8  48  15  12
##   2  11  11  26  35
##   3   0   0 125   0
##   4 101  12  87   9
##
##   1   2   3   4
##  83  83 125 209
##
##       1   2   3   4
##   1  16   5  23  39
##   2  31   8  19  25
##   3   0   0 125   0
##   4   0  75 104  30
```

```r
1-(6+4+3+8+3+45+11)/(125+209+83+83)
```

```
## [1] 0.84
```

```r
1-(2+7+35+2+13+11+5+11+87)/(125+209+83+83)
```

```
## [1] 0.654
```

From above, we can see that when sd=0.5, the error rate is okay, while when sd = 1, the error rate is pretty big, so the threshold should below 0.5 (assume at least the accurate rate should be 0.90).

```r
for (n in seq(0.3, 0.5, 0.1))
{
smiley=mlbench.smiley(n=500, sd1 = n, sd2 = n)
smiley.df=as.data.frame(smiley)
model=kmeans(smiley.df[,1:2], 4, nstart=20)
print(table(smiley.df$classes))
print(table(smiley.df$classes, model$cluster))
}
```

```
##
##   1   2   3   4
##  83  83 125 209
##
##       1   2   3   4
##   1   8   0   0  75
##   2   3   0  79   1
##   3 125   0   0   0
##   4  22 169  14   4
##
##   1   2   3   4
##  83  83 125 209
##
##       1   2   3   4
##   1   3   2  78   0
##   2  10  73   0   0
##   3 125   0   0   0
##   4  31   6   1 171
##
##   1   2   3   4
##  83  83 125 209
##
##       1   2   3   4
##   1  15   0   0  68
##   2  17  64   0   2
##   3 125   0   0   0
##   4  42   6 154   7
```

```
1-(8+3+1+22+14+4)/(125+209+83+83)
```

```
## [1] 0.896
```

```
1-(3+2+10+31+6+1)/(125+209+83+83)
```

```
## [1] 0.894
```

```
1-(15+17+2+42+6+7)/(125+209+83+83)
```

```
## [1] 0.822
```

From above, we can see the error rate is increasing when we increasing the sd, so the threshold should less than 0.3.

```
for (n in seq(0.25, 0.29, 0.01))
{
smiley=mlbench.smiley(n=500, sd1 = n, sd2 = n)
smiley.df=as.data.frame(smiley)
model=kmeans(smiley.df[, 1:2], 4, nstart=20)
print(table(smiley.df$classes))
print(table(smiley.df$classes, model$cluster))
}
```

```
##
##   1   2   3   4
##  83  83 125 209
##
##       1   2   3   4
##   1   0   0   5  78
```

```
##    2  82   0   1   0
##    3   0   0 125   0
##    4   3 167  38   1
## 
##   1   2   3   4 
##  83  83 125 209 
## 
##        1   2   3   4
##    1   0  82   1   0
##    2  82   0   1   0
##    3   0   0 125   0
##    4   1  11  17 180
## 
##   1   2   3   4 
##  83  83 125 209 
## 
##        1   2   3   4
##    1   3   0  80   0
##    2   0  83   0   0
##    3 125   0   0   0
##    4  29   9   2 169
## 
##   1   2   3   4 
##  83  83 125 209 
## 
##        1   2   3   4
##    1  75   0   8   0
##    2   0   0   1  82
##    3   0   0 125   0
##    4   1 168  28  12
## 
##   1   2   3   4 
##  83  83 125 209 
## 
##        1   2   3   4
##    1   8  75   0   0
##    2   2   0  81   0
##    3 125   0   0   0
##    4  22   1  14 172
```

```r
1-(5+1+3+38+1)/(125+209+83+83)
```

```
## [1] 0.904
```

```r
1-(1+1+1+11+17)/(125+209+83+83)
```

```
## [1] 0.938
```

```r
1-(3+29+9+2)/(125+209+83+83)
```

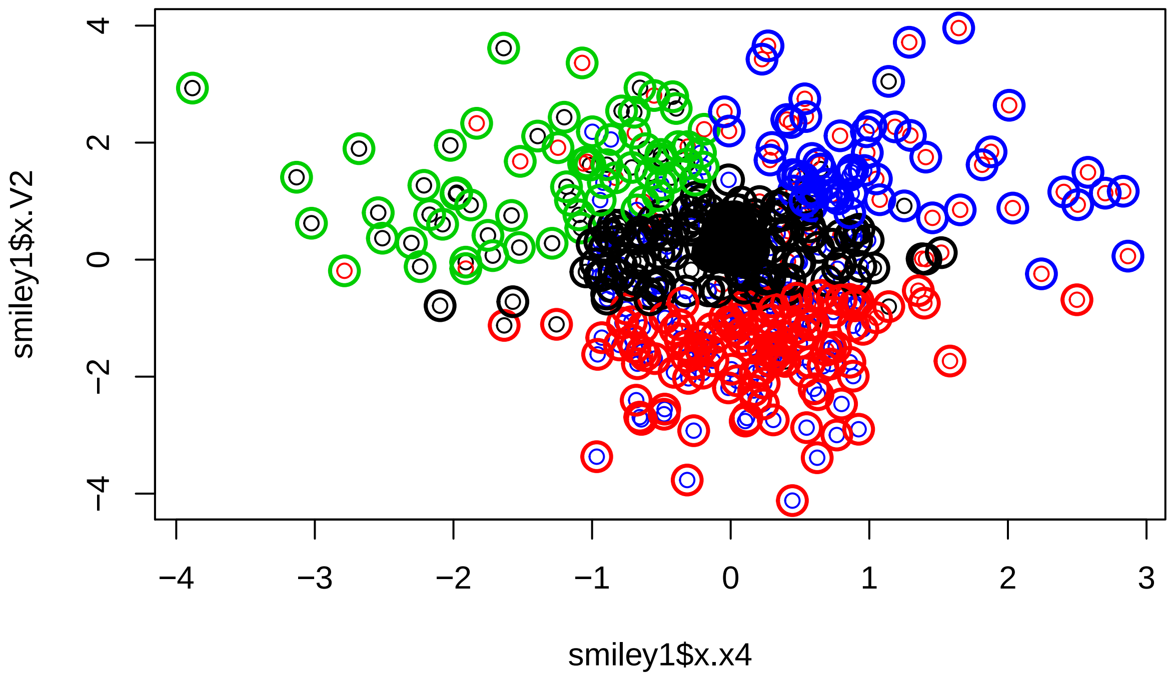
```
## [1] 0.914
```

```r
1-(8+1+1+28+12)/(125+209+83+83)
```

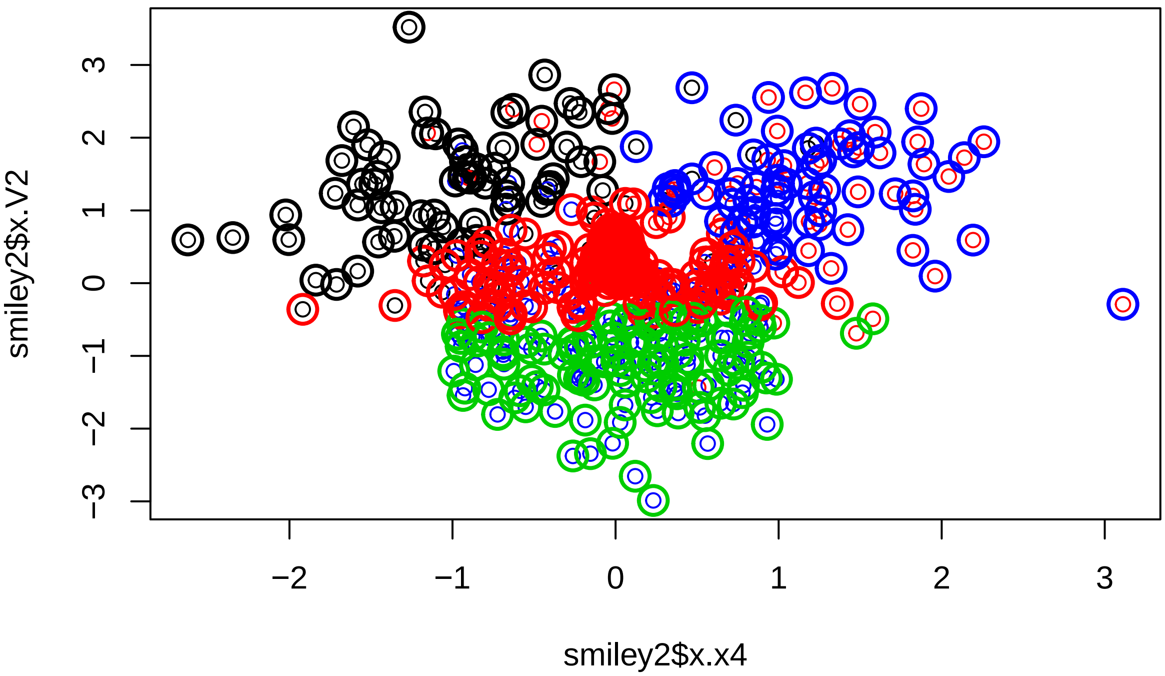
```
## [1] 0.9
```

From above results, I choose to set the boundary as 0.26 since after that the accurate rate will decrease.

```r
# Example 1
smiley1=mlbench.smiley(n=500, sd1 = 1.2, sd2 = 1.2)
smiley1=as.data.frame(smiley1)
model1=kmeans(smiley1[,1:2], 4, nstart=20)
plot(smiley1$x.x4, smiley1$x.V2, col=smiley1$classes, lwd=1)
points(smiley1$x.x4, smiley1$x.V2, col=model1$cluster, pch=1, cex=2, lwd = 2)
```




```r
# Example 2
smiley2=mlbench.smiley(n=500, sd1 = 0.8, sd2 = 0.8)
smiley2=as.data.frame(smiley2)
model2=kmeans(smiley2[,1:2], 4, nstart=20)
plot(smiley2$x.x4, smiley2$x.V2, col=smiley2$classes, lwd=1)
points(smiley2$x.x4, smiley2$x.V2, col=model2$cluster, pch=1, cex=2, lwd = 2)
```



From the graphs we can see that when sd is larger, we can hardly recognize the smile face and can easily find mislabeled points from the graph.

Problem Extra #71

a)

```
set.seed(123)
MNIST=load('~/Desktop/other/Data/mnist_all.RData')

# Generate dataframe
train.data=as.data.frame(train)
```

```
# k-means clustering
model.1=kmeans(train.data, 2)
table(train.data$y, model.1$cluster)
```

```
##
##        1    2
##   0  225 5698
##   1 6729   13
##   2 2278 3680
##   3 2299 3832
##   4 5030  812
##   5 2852 2569
##   6 2392 3526
##   7 6091  174
##   8 3810 2041
##   9 5420  529
```

We can see that number 1, 4, 7, 9 are tend to be clustered together, because cluster1 contains a very large part of these digits, while cluster2 only contains a few. For other digits, the percentage for cluster1 and cluster2 are similar.

b)

```
# k-means clustering with n=10
model.2=kmeans(train.data, 10)

# Confusion matrix
table(train.data$y)
```

```
##
##    0    1    2    3    4    5    6    7    8    9
## 5923 6742 5958 6131 5842 5421 5918 6265 5851 5949
```

```
table(train.data$y, model.2$cluster)
```

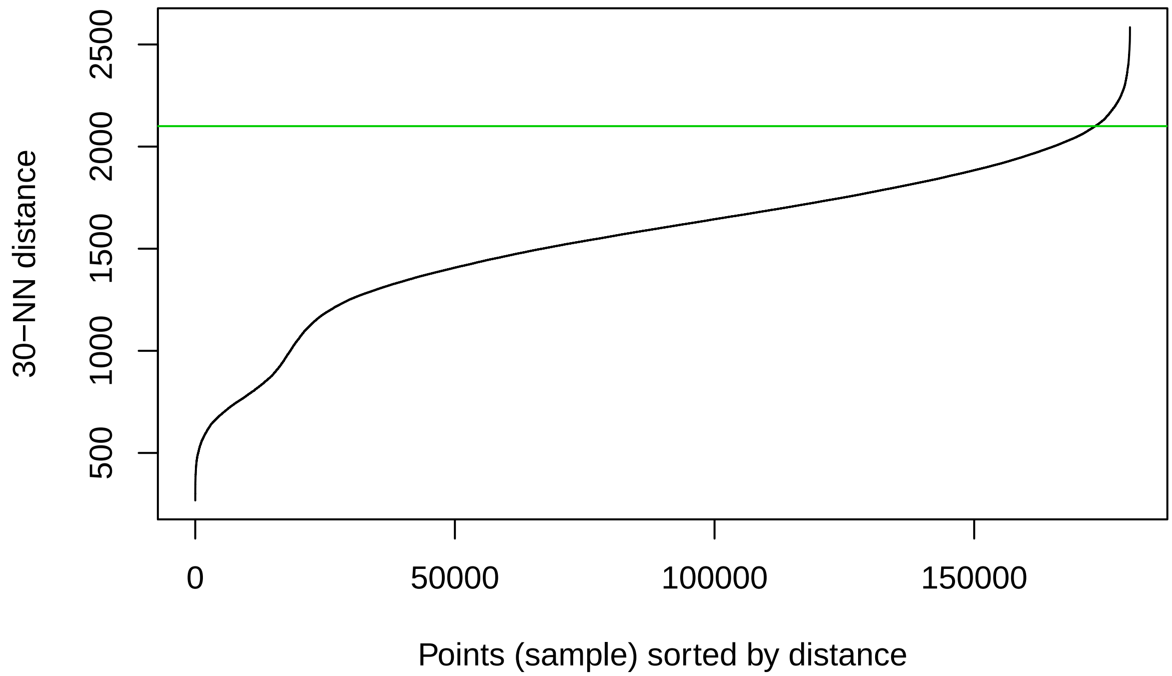
```
##
##        1    2    3    4    5    6    7    8    9   10
##   0   38   73   10    7   10  157    8  205 1802 3613
##   1    6    7   10    9 3031    5 3668    5    1    0
##   2  179  185   68 4155  336  173  382  278  173   29
##   3  168  879   39  144   71   39  451 3867  454   19
##   4 3182   16 1954   35  275  140  184    1   46    9
##   5  378  891  232    4  519   83  248 1798 1222   46
##   6   75   40    1   74  134 4543  382   27  564   78
##   7 1793   18 3765   37  275    4  340    5   12   16
##   8  173 3536  174   44  361   44  323 1018  150   28
##   9 2897   69 2453   15  102    7  269   85   14   38
```

We can see that this k-means clustering performs not very good. For example, for the original digit 0, there are two clusters containing more than 1000 observations, which means at least 1802 observations are wrong and we cannot classify digit 0 accurately. For digit 1, cluster 5 and 7 both contain more than 3000 observations, which menas at least half of the original digit 1 are mislabeled.

c)

```
# Since I cannot generate results by using the whole dataset
# I decide to choose 10% dataset randomly for this step
n=dim(train.data)[1]
index=sample(n, n*0.1)
X.1=train.data[index,]

# Prepare the dataset
X = train.data[index,2:785]
X = as.matrix(X)
```

```
# With minPts = 30
kNNdistplot(X,30)
abline(h = 2100, col = 3)
```



From the plot we can see that when minPts = 30, we should use eps = 2100.

```
# Build DBSCAN
try1=dbscan(X, eps = 2100, minPts = 30)
```

```
# Confusion matrix
table(X.1$y)
```

```
##
##   0   1   2   3   4   5   6   7   8   9
## 594 663 582 593 556 535 604 657 628 588
```

```
table(X.1$y, try1$cluster)
```

```
##
##       0   1
##   0   2 592
##   1   0 663
##   2   8 574
##   3   0 593
##   4   2 554
##   5   1 534
##   6   2 602
##   7   1 656
##   8   6 622
##   9   5 583
```

From dbscan clustering, I only get two clusters and most of the numbers belong to cluster 1. Obviously it is not a good cluster for a dataset that has 10 groups.

Problem Extra #72

```r
# Load the data and change the names
library(readxl)
concrete <- read_excel("~/Desktop/other/Data/Concrete_Data.xls")
names(concrete)=c('Cement', 'Slag', 'Fly', 'Water', 'Super', 'Coarse', 'Fine', 'Age', 'Strength')

# Split the data
n=dim(concrete)[1]
index=sample(n, n*0.7, replace = FALSE)
train=concrete[index,]
test=concrete[-index,]
```

a)

```r
# Compute PCA
train.0 = train[ ,-9]
train.pca = prcomp(train.0, scale=F)
pca = train.pca$x
PCA1 = pca[,1]

# New dataframe
df1=data.frame(train$Strength)
df1$pca1=PCA1

# Build a linear model
pca.model=lm(train.Strength~pca1, data=df1)
summary(pca.model)
```

```
##
## Call:
## lm(formula = train.Strength ~ pca1, data = df1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -31.958 -10.855  -1.045   9.445  44.055
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 35.643050   0.547520   65.10   <2e-16 ***
## pca1        -0.060437   0.004819  -12.54   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.7 on 719 degrees of freedom
## Multiple R-squared:  0.1795, Adjusted R-squared:  0.1784
## F-statistic: 157.3 on 1 and 719 DF,  p-value: < 2.2e-16
```

b)

```r
# Loading vectors for traning data
load = train.pca$rotation
```

```r
# Score for test data
test.0 = test[ , -9]
score = as.matrix(test.0)%*%(as.matrix(load))
score1 = score[, 1]

# New dataframe
df2=data.frame(test$Strength)
df2$pca1=score1

# Predict for test data
pred.test = predict(pca.model, newdata=df2)
rms = sqrt(mean((pred.test-test$Strength)^2))
```

The rms error is 16.7392838 on the test data.