Final Project Q3

Wenjing Xu

October 31, 2018

Question 3

(1) Algorithm for K-means clustering

To implement K-Means algorithm, we should do following steps.

- Randomly choose K points (which is 3 here) as the initial centers.
- For each observation, determine which cluster center is the cloest to it using Euclidean distance measure. Then, assign that observation to the corresponding cluster.
- Compute the mean location of the points in each cluster. Set this mean location as the new cluster center point.
- Repeat step 2 and 3, until there is no more change for clusters.

We now write a K-means sampler which takes data for clustering(initial type not included) and number of clusters as parameters and returns a vector showing which cluster each point belongs to.

```r
k_means_cluster <- function(cluster_data,num_cluster){
  cluster_data <- as.matrix(cluster_data)
  num_points <- nrow(cluster_data)
  num_dim <- ncol(cluster_data)
  #sample from dataset to get initial mean of three clusters
  start_mean <- cluster_data[sample(1:num_points,num_cluster,replace = F),]
  new_cluster_label <- numeric(num_points)
  old_cluster_label <- rep(1,num_points)
  #stop iterating when classification does not change
  while (any(old_cluster_label != new_cluster_label)){
    old_cluster_label <- new_cluster_label
    distance <- matrix(0,num_points,num_cluster)
    #calculate each data points' Euclidean distance to each old mean of clusters
    for (j in 1:num_cluster){
      distance[,j] <- apply((cluster_data - matrix(start_mean[j,],nrow = num_points,
                                                    ncol = num_dim,byrow = TRUE))^2,1,sum)
    }
    #find shortest distance and assign each data point to a new cluster
    new_cluster_label <- apply(distance,1,which.min)
    #calculate new mean of new clusters
    for (j in 1:num_cluster){
      start_mean[j,] <- apply(cluster_data[which(new_cluster_label==j),],2,mean)
    }
  }
  #return a vector indicating each data point's cluster
  return(new_cluster_label)
}
```

Now we use this classifier to cluster the wines into 3 groups.

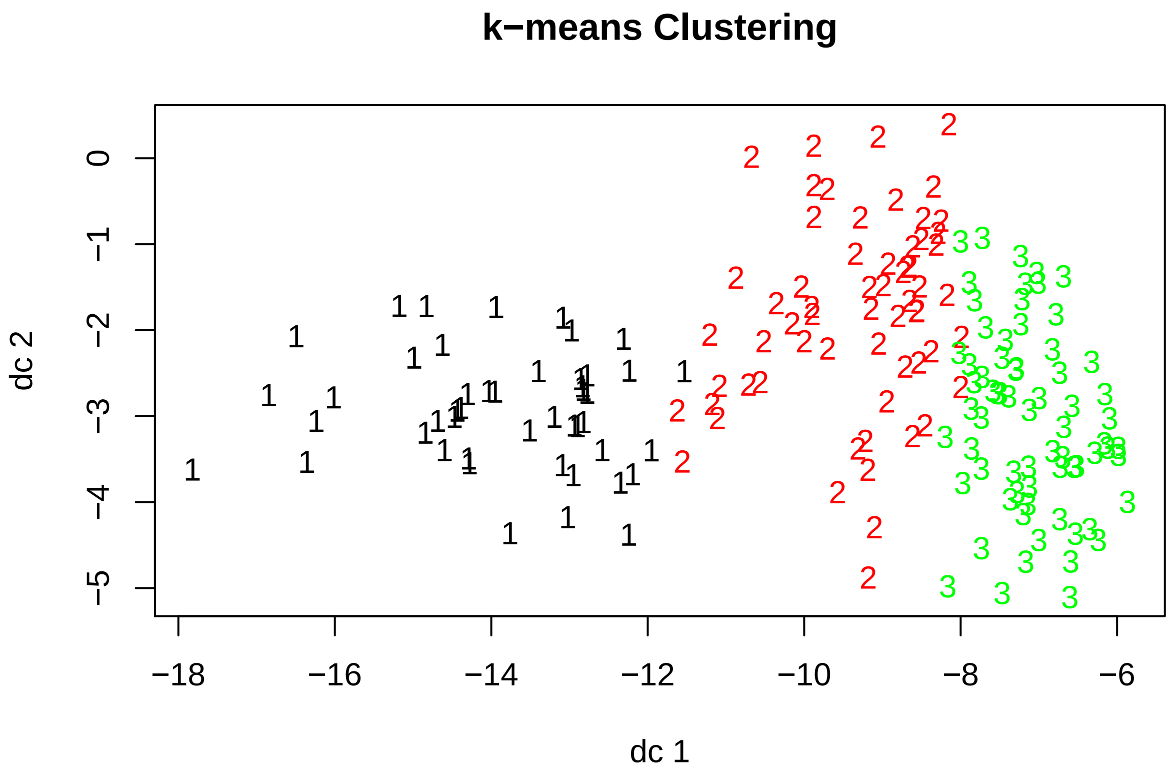
```r
library("rattle")
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.2.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
data(wine)
set.seed((3.1))
num_cluster <- 3
newdata <- wine[,-1] #exclude original Type
label <- k_means_cluster(newdata,num_cluster) #result of k-means
```

The result of k-means clustering is shown below.

```
library(fpc)
plotcluster(newdata,label,main="k-means Clustering")
```



The data points seem well seperated by the classifier although same boundary points are close.

(2) Quantify how well the result corresponds to original types

Now we want to quatify how well the result correspond to the original data. To quantify this, we first need to find out which original types do these new clusters correspond to. To match them, we first calculate the proportion of three original types in each new cluster and match each new cluster with the type whose propotion is largest in this cluster. In case that two new clusters may match the same original type, we match the new cluster of largest number in proportion matrix first. Then match the cluster of largest number in reduced 2*2 proportion matrix. And the remaining cluster matches the remaining type. Hence the match between old types and new clusters are built.

```r
#the function of calculating the proportion matrix
matching_matrix <- function(old_type,new_type,num_cluster){
  proportion <- matrix(0,num_cluster,num_cluster)
  for(i in 1:num_cluster){
    index <- (new_type==i)
    for (j in 1:num_cluster){
      proportion[i,j] <- sum(old_type[index]==j)/sum(index)
    }
  }
  return(proportion)
}
#proportion matrix for wine[,-1]
matching_matrix(wine$Type,label,num_cluster)
```

```
##           [,1]      [,2]      [,3]
## [1,] 0.9787234 0.0212766 0.0000000
## [2,] 0.2096774 0.3225806 0.4677419
## [3,] 0.0000000 0.7246377 0.2753623
```

From the proportion matrix, we can see that new cluster 1 matches type 1, new cluster 2 matches type 3, new cluster 3 matches type 2.

```r
#decide actual type of each new cluster matches according to proportion matrix
actual <- c(1,3,2)
#the function assigns actual label to points
new_cluster <- function(new_type,actual_type){
  new_index <- numeric(length(new_type))
   for (i in 1:length(actual_type)){
     new_index[(new_type==i)] <- actual_type[i]
   }
   return(new_index)
}
#relabel data points to match
new_label <- new_cluster(label,actual)
```

So we can further quantify how well the result fits original types using the number of points whose type(label) does not change before and after k-means clustering.

```r
#The function gives the propotion whose label does not change
right_rate <- function(old_type,new_cluster){
 return(sum(old_type==new_cluster)/length(old_type))
}
right_rate(wine$Type,new_label)
```

```
## [1] 0.7022472
```

So the right rate of the classifier for wine[,-1] is 0.7022472. We think this result is well seperated.

(3) The effect of scaling

```r
#scale the data
newdata1 <- scale(wine[,-1])
apply(newdata1,2,mean)
```

```
##         Alcohol           Malic             Ash      Alcalinity
##   -8.591766e-16   -6.776446e-17    8.045176e-16   -7.720494e-17
##       Magnesium         Phenols      Flavanoids   Nonflavanoids
##   -4.073935e-17   -1.395560e-17    6.958263e-17   -1.042186e-16
## Proanthocyanins           Color             Hue        Dilution
##   -1.221369e-16    3.649376e-17    2.093741e-16    3.003459e-16
##         Proline
##   -1.034429e-16
```

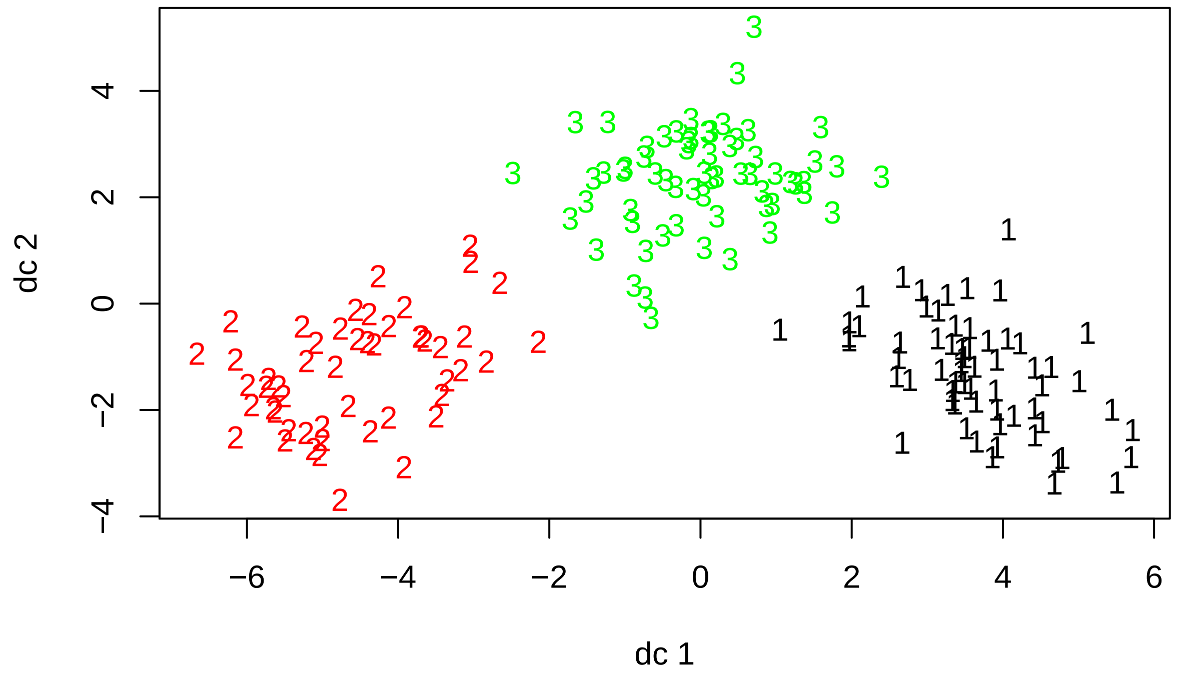
```
apply(newdata1,2,sd)
```

```
##         Alcohol           Malic             Ash      Alcalinity
##               1               1               1               1
##       Magnesium         Phenols      Flavanoids   Nonflavanoids
##               1               1               1               1
## Proanthocyanins           Color             Hue        Dilution
##               1               1               1               1
##         Proline
##               1
```

The function scale() transform all data points into a family of points with mean 0 and standard error 1.

Now we repeat the calculation process for scaled data.

```
#cluster the scaled data
set.seed(3.2)
label1 <- k_means_cluster(newdata1,num_cluster)
plotcluster(newdata1,label1)
```



From the plot, we can see that these data points are well seperated. The distances between clusters seem larger than those in Part 1. Now we show this in calculation.

```r
matching_matrix(wine$Type,label1,num_cluster)
```

```
##          [,1]       [,2]      [,3]
## [1,] 0.921875 0.07812500 0.0000000
## [2,] 0.000000 0.05882353 0.9411765
## [3,] 0.000000 1.00000000 0.0000000
```

So new cluster 1 matches type 1, new cluster 2 matches type 3, new cluster 3 matches type 2.

```r
#match clusters with old corresponding types
actual <- c(1,3,2)
new_label1 <- new_cluster(label1,actual)
right_rate(wine$Type,new_label1)
```

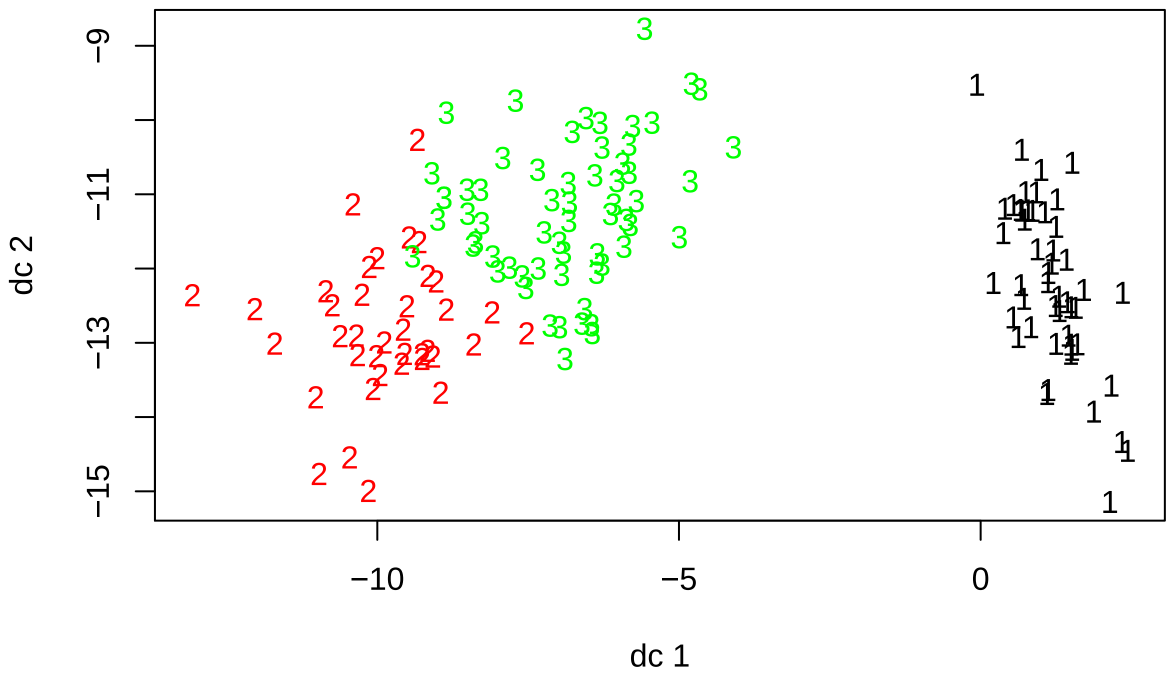
```
## [1] 0.9550562
```

So the right rate of the classifier for scaled data wine[,-1] is 0.9550562 which is obviously larger than the right rate for original data.

Therefore, in this case, using scaled data can help us improve the performance of k-means clustering, making more data points' new cluster matches original type.

(4) Application to iris dataset

```r
#k-means clustering for iris data
data("iris")
levels(iris$Species) <- c(1,2,3)
newdata <- iris[,-5]
set.seed(3.3)
label <- k_means_cluster(newdata,num_cluster)
plotcluster(newdata,label)
```



The data points of new cluster 1 seem well seperated by the classifier. Other points seem not as well seperated as cluster 1 but are good in general although the boudary of new cluster 2 and 3 is not very obvious.

```
matching_matrix(iris$Species,label,num_cluster)
```

```
##      [,1]       [,2]      [,3]
## [1,]    1 0.00000000 0.0000000
## [2,]    0 0.05263158 0.9473684
## [3,]    0 0.77419355 0.2258065
```

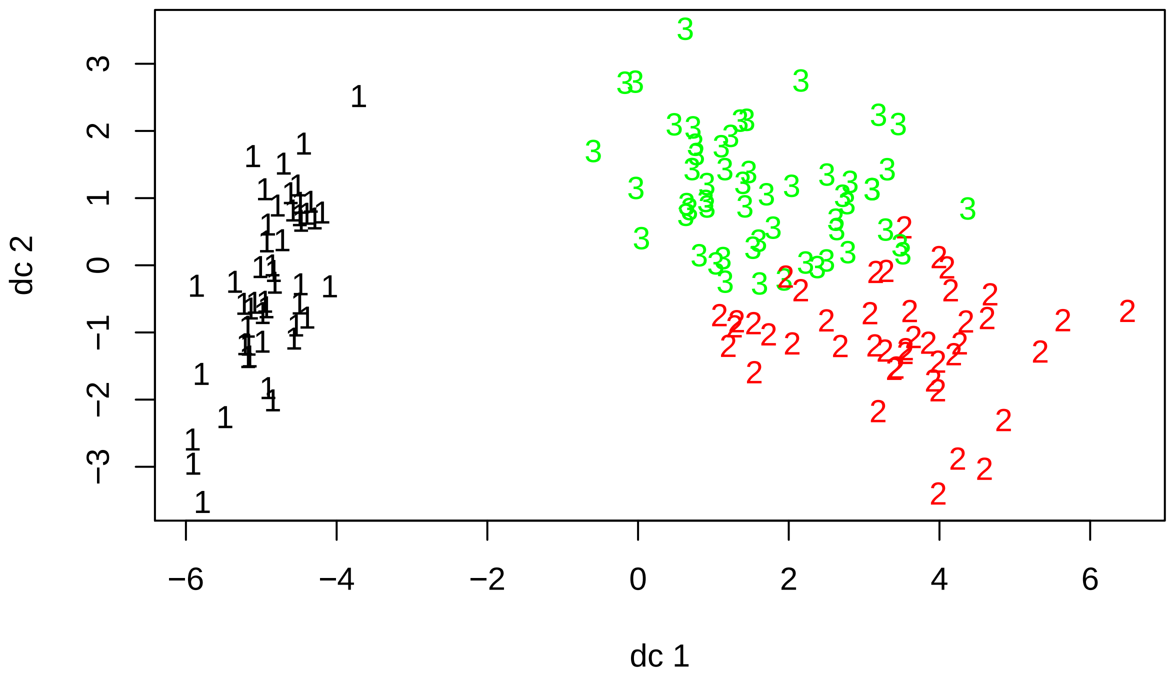
So new cluster 1 matches type 1, new cluster 2 matches type 3, new cluster 3 matches type 2.

```
actual <- c(1,3,2)
new_label <- new_cluster(label,actual)
right_rate(iris$Species,new_label)
```

```
## [1] 0.8933333
```

So the right rate of the classifier for iris[,-5] is 0.8933333, which indicates that the classifier works well for this dataset.

```
set.seed(3.4)
newdata1 <- scale(iris[,-5])
label1 <- k_means_cluster(newdata1,num_cluster)
plotcluster(newdata1,label1)
```



It can be seen from this plot that these data points are well seperated in general but the small problem in results from original iris data still exists. So we might guess the effect of scaling may not be very obvious in this case or even worse.

```
matching_matrix(iris$Species,label1,num_cluster)
```

```
##      [,1]      [,2]      [,3]
## [1,]    1 0.0000000 0.0000000
## [2,]    0 0.2500000 0.7500000
## [3,]    0 0.6964286 0.3035714
```

So new cluster 1 matches type 1, new cluster 2 matches type 3, new cluster 3 matches type 2.

```
actual <- c(1,3,2)
new_label1 <- new_cluster(label1,actual)
right_rate(iris$Species,new_label1)
```

```
## [1] 0.8133333
```

We observe that the right rate declines a little bit for scaled data. So scaling can not improve the accuracy of k-means clustering in this case. We may conclude that whether scaling can help to improve k-means clutering's performance depends on the structure of the dataset.