Final Project Part 3

Feng Wang

October 27, 2016

```
library('rattle')
library('fpc')
data(wine, package="rattle")
```

1. Algorithm and function

Algorithm

a. Randomly assigned three clusters means
b. Decide each data should belong in which cluster
   Compute the distance between one data with all means, and the data should belong in the cluster with the least distance.

c. Compute the objection function

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|x_n - \mu_k\|^2$$

d. Now change the means for different cluster Since all the data has changed their cluster, for each cluster mean, it also need to update.

$$\mu_k = \frac{\sum_n r_{nk} x_n}{\sum_n r_{nk}}$$

e. Then compute the objection function again
f. start again from step b, until the J is stable

Main function

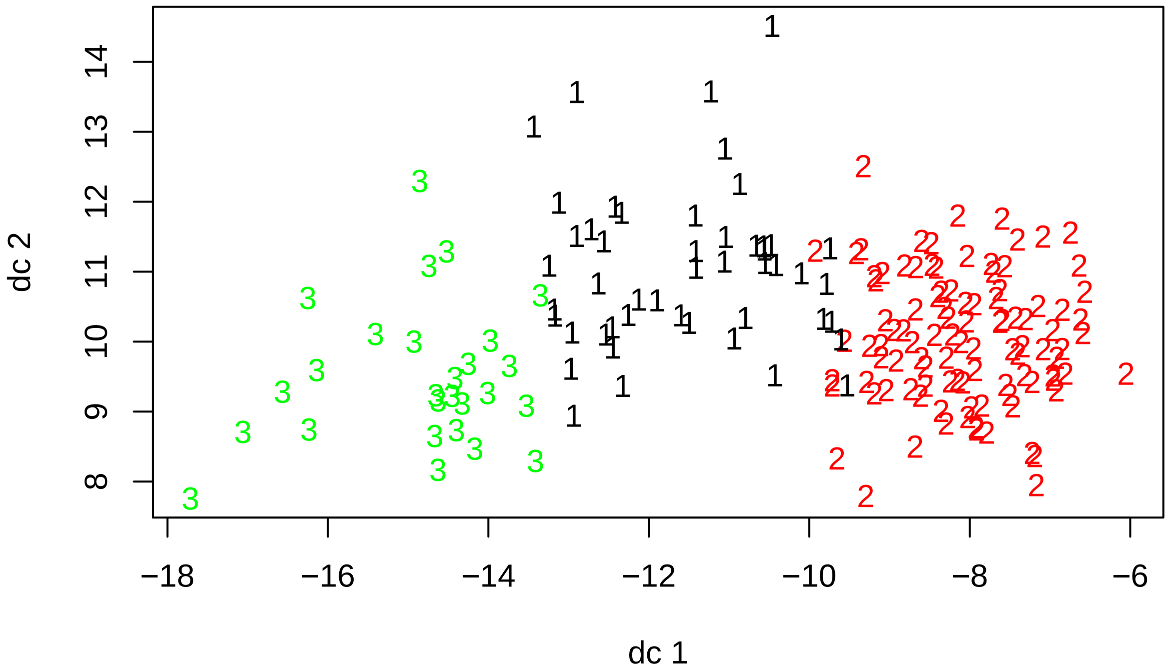
```
K_means = function(X, k){
  # group function compute for each data should belong to which cluster
    # compute the distance between data with each means
    group = function(mu, X, n){
    r = rep(0,n)
    for(i in 1:n){r[i] = which.min(rowSums(sweep(mu, 2, X[i,], "-") ^ 2))}
    return(r)
  }
  # j functions compute given means and assigned cluster, the objection function
  # clusting error
  J = function(X, mu, n, r){
    s = 0
    for(i in 1:n){s = s + sum((X[i,] - mu[r[i],])^2)}
    return(s)
  }
  # mu_adj function compute each cluster means
```

```
  mu_adj = function(r, X ,k){
    mu = matrix(0, k, dim(X)[2])
    for(i in 1:k){
      mu[i,] = ((r == i) * 1) %*% X / sum(r == i)
    }
    return(mu)
  }
  # Main function
  # Compute the first two steps
  n = dim(X)[1]; mu_start = X[1:k,]
  r_start=group(mu_start,X,n)
  Iter=0
  j_dm=J(X,mu_start,n,r_start)
  mu=mu_adj(r_start,X,k)
  Iter=1
  j_dm = c(j_dm,J(X,mu,n,r_start))
  r=group(mu,X,n)
  Iter = 2

  # Use the while loops to do the K-means, and stop when J keeps the same
  while(abs(j_dm[Iter]-j_dm[Iter-1])!=0){
      mu=mu_adj(r,X,k)
      Iter=Iter+1
      j_dm=c(j_dm,J(X,mu,n,r))

      r=group(mu,X,n)
      Iter=Iter+1
      j_dm=c(j_dm,J(X,mu,n,r))
  }
  return(r)
}
```

2. Wine data

Train data without scale

```r
data.train = data.matrix(wine[-1]); k = 3
r1 = K_means(data.train,k)
plotcluster(data.train,r1)
```



It seems that the data has been separated, but the boundares of these three clusters are not clear. Now we bring another quantify called "accuracy" to test the clustering efficiency.

$$Accuracy = \frac{correct}{n}$$

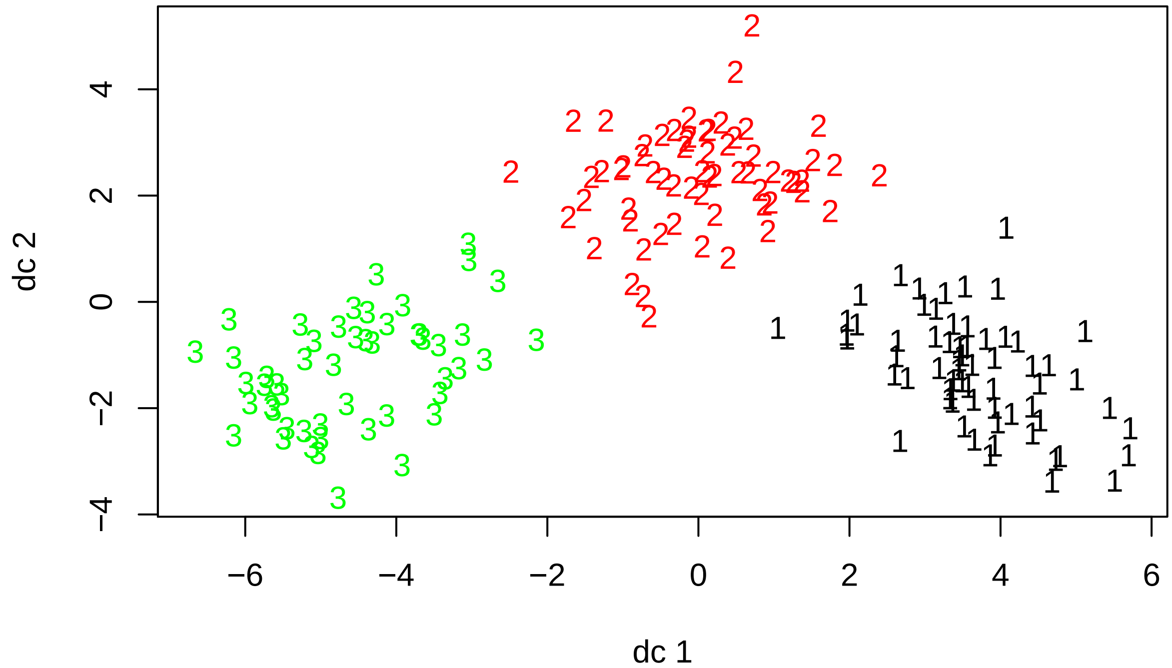
```r
Accuracy = sum(r1 == wine[1])/length(r1)
print(paste("the Accuracy for non-scale data is ", Accuracy))
```

```
## [1] "the Accuracy for non-scale data is  0.533707865168539"
```

We find the accuracy for the data without scale is slightly greater than 50%, and it meets our observation that the boundary is blured.

Train data with scale

```
data.train = scale(wine[-1])
r2 = K_means(data.train,k)
plotcluster(data.train,r2)
```




```
Accuracy = sum(r2 == wine[1])/length(r2)
print(paste("the Accuracy for scale data is ", Accuracy))
```

```
## [1] "the Accuracy for scale data is  0.955056179775281"
```

The scale data has a highly accuracy (>90%) after the clustering, and three clusters in the graph also separate thoroughly.
The reason is that for the non-scale data, some variable may much larger than the others, and it will dominate the error when doing the K-means.But this problem could be solved using scale data.

3. Iris data

Train data without scale

```
data(iris)
data.train = data.matrix(iris[1:4]); k = 3; true_cat = iris[5]
true_cat = as.numeric(factor(true_cat$Species, levels = sort(unique(true_cat$Species))))
```

```
r3 = K_means(data.train,k)
plotcluster(data.train,r3)
```