K-means algorithm

PoHsiang Peng, Jinjie Zhang

K-means algorithm

Problem:

given the data points $X = \{x_1, x_2, \ldots, x_n\} \subseteq \mathbb{R}^d$ and the desired number of clusters $k \in \mathbb{N}$, we try to find a partition of dataset $\{C_1, C_2, \ldots, C_k\}$ (i.e. $\bigcup_{j=1}^{k} C_j = X$, $C_i \bigcap C_j = \emptyset$ for $i \neq j$) and centroid $m_1, m_2, \ldots, m_k \in \mathbb{R}^d$ such that

$$\sum_{j=1}^{k} \sum_{x \in C_j} \|x - m_j\|^2$$

is minimized. Here, $\|\cdot\|$ is the Euclidean norm.

Algorithm:

$\{m_1, m_2, \ldots, m_k\} \leftarrow$ centroids are initialized randomly;

while(convergence){

$C_1, C_2, \ldots, C_k \leftarrow \emptyset$;

for $i = 1$ to n {

$\hat{j} \leftarrow \arg\min_{j \in \{1,\ldots,k\}} \|x_i - m_j\|$;

$C_{\hat{j}} \leftarrow C_{\hat{j}} \cup \{x_i\}$;

}

for $j = 1$ to k {

$m_j \leftarrow \frac{1}{|C_j|} \sum_{x \in C_j} x$;

}

}

return $\{m_1, m_2, \ldots, m_k\}$

(a) Implement the algorithm without scaling "rattle" data

First of all, we need to write a function for K-means.

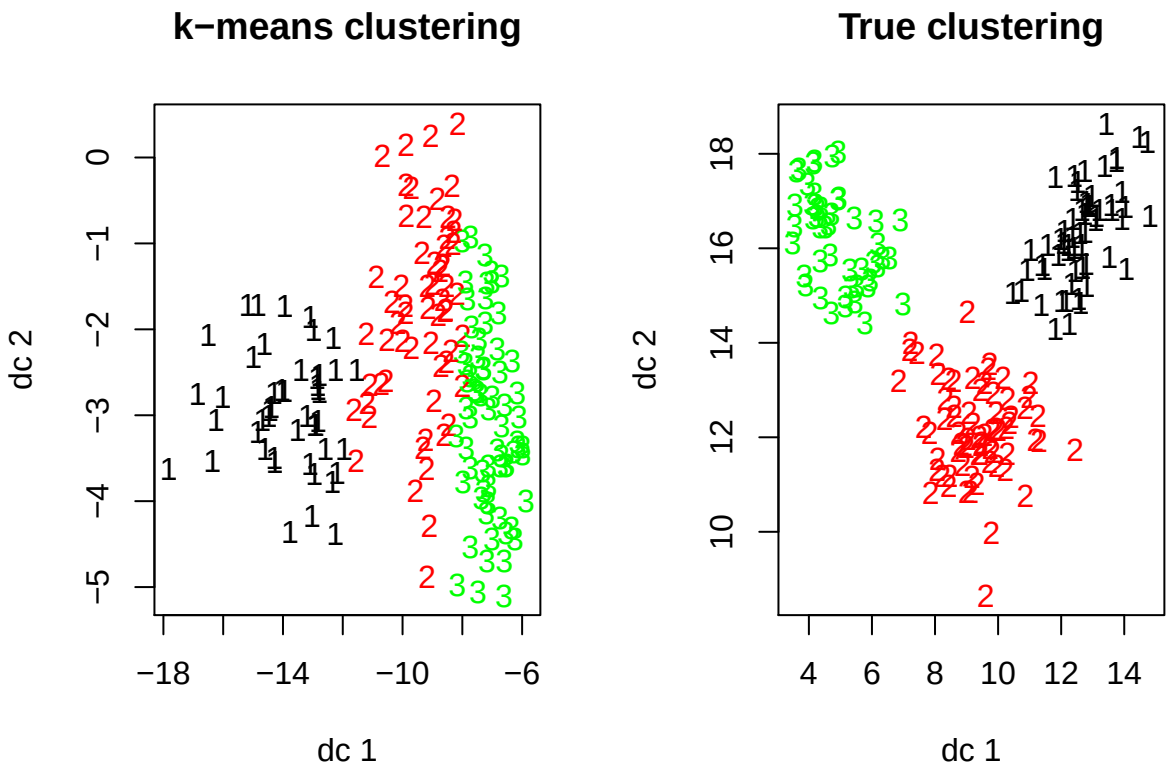
```
km=function(x,k,epsilon=1e-7){ #x is the training data; k is the number of clusters
  r= dim(x)[1]
  c= dim(x)[2]  #the number of rows and columns
  lb = apply(x,2,min) #lower bounds of the features
  ub = apply(x,2,max) #upper bounds of the features
  initial.cen = matrix(runif(c*k,lb,ub), ncol=c, byrow=T) # generate the initial centroids randomly
  new.cen=initial.cen
  dist = matrix(nrow=r,ncol=k) #matrix used to store distances
  error=Inf
  while(error>epsilon) {
    current.cen = new.cen
```

```
    for(j in 1:k)  #calculate the distance from data point to clusters
      dist[,j] = apply(x,1,function(y) norm(y-current.cen[j,],"2")^2)
   index=apply(dist,1,which.min) #each data point is assigned to the closest center

     for(j in 1:k)
      new.cen[j,]=apply(x[index==j,],2,mean) #calculate new centers
    error = norm(current.cen-new.cen,"F")
  }
  return(index)
}
```

Now we can compare our result with original data.

```
library("rattle.data")
library("fpc")
data(wine, package="rattle.data")
data.train =wine[-1]
cluster.index=km(data.train,3)
par(mfrow=c(1,2))
plotcluster(data.train, cluster.index,main="k-means clustering")
plotcluster(data.train, wine$Type, main="True clustering")
```



From above plots, we find that without scaling the data the performance of k-means is poor.

Moreover, we can develop a method to quantify how well our algorithm's clusters correspond to the three wine types. There are three wine types with indices $1 \leq i \leq 59$, $60 \leq i \leq 130$ and $131 \leq i \leq 178$. Now we compute the total number of errors by comparing the indices of all distinct pairs.

```
error=sum(dist(cluster.index[1:59]))+sum(dist(cluster.index[60:130]))+sum(dist(cluster.index[131:178]))
print(error)
```

```
## [1] 2269
```

The the number of errors is pretty high.

(b) Implement the algorithm using scaled "rattle" data

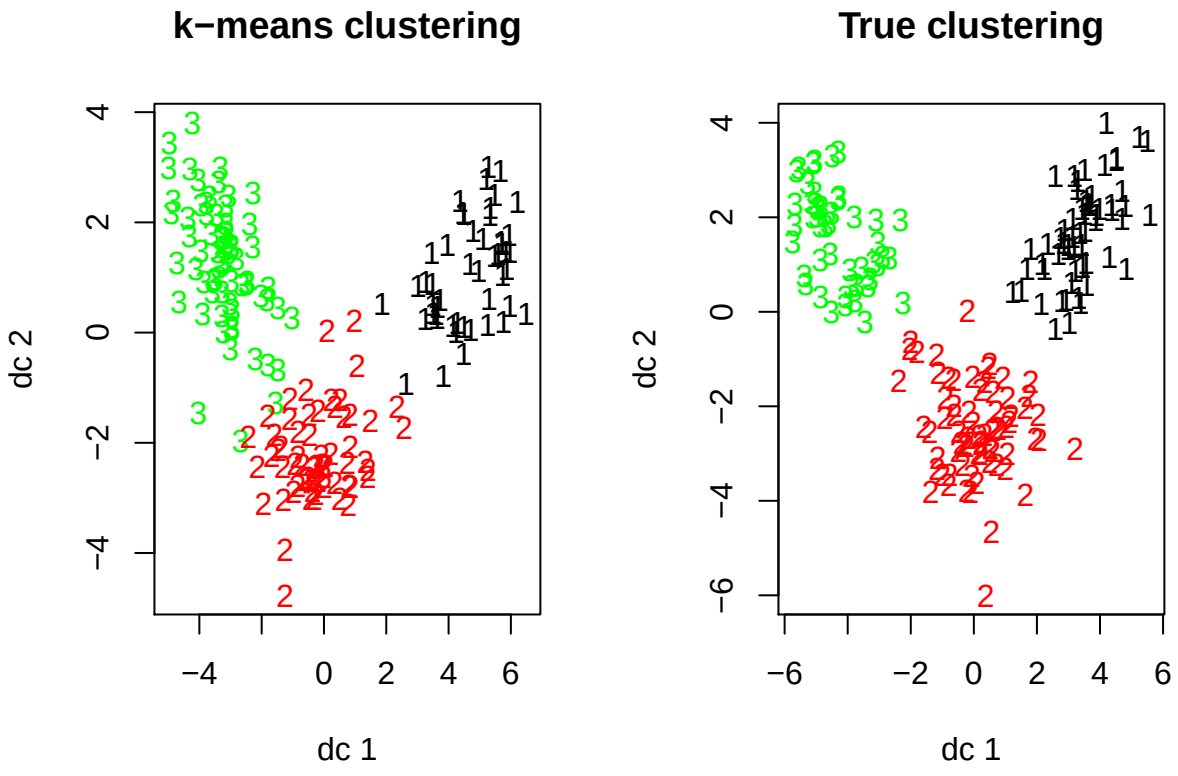
Before we implement the k-means algorithm, the data need to be scaled by using following code:

```
data.train = as.data.frame(scale(wine[-1]))
```

Note that *scale* is a generic function whose default method centers and/or scales the columns of a numeric matrix such that our data points are concentrated, which wil help us to improve the performance of k-means algorithm.

Again, we can compare our result with original data.

```
cluster.index=km(data.train,3)
par(mfrow=c(1,2))
plotcluster(data.train, cluster.index,main="k-means clustering")
plotcluster(data.train, wine$Type, main="True clustering")
```



We notice that the performance of k-means is significantly improved by scaling the data. Moreover, we can show how those data points are classfied into three groups:

```
error=sum(dist(cluster.index[1:59]))+sum(dist(cluster.index[60:130]))+sum(dist(cluster.index[131:178]))
print(error)
```

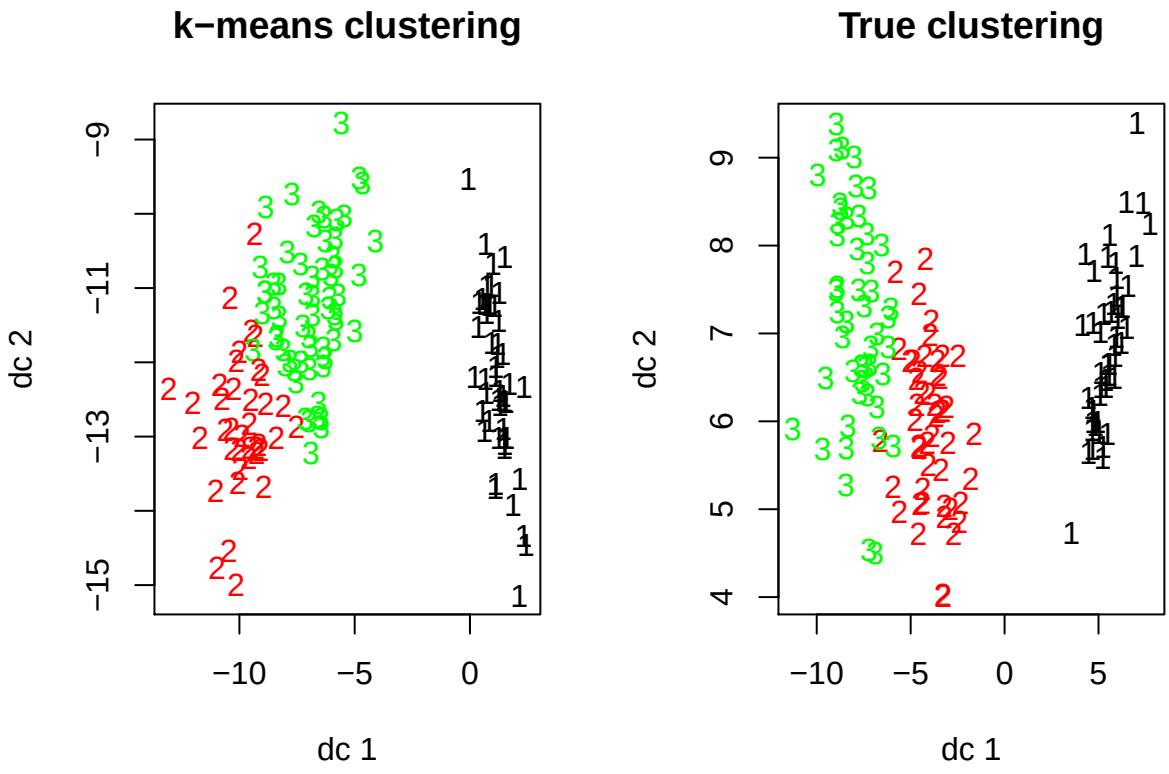
```
## [1] 574
```

The number of errors decreases significantly. It implies that our classification is close to the original wine types.

(c) Implement the algorithm without scaling the "iris" data

Exactly the same method as above, we first compare our result with original data.

```
data.train =iris[-5]
cluster.index=km(data.train,3)
par(mfrow=c(1,2))
plotcluster(data.train, cluster.index,main="k-means clustering")
plotcluster(data.train, iris$Species, main="True clustering")
```



Similarly, we can compute the total number of errors as follows:

```
error=sum(dist(cluster.index[1:50]))+sum(dist(cluster.index[51:100]))+sum(dist(cluster.index[101:150]))
print(error)
```

```
## [1] 600
```

This result shows that k-means fails to give a good classification.

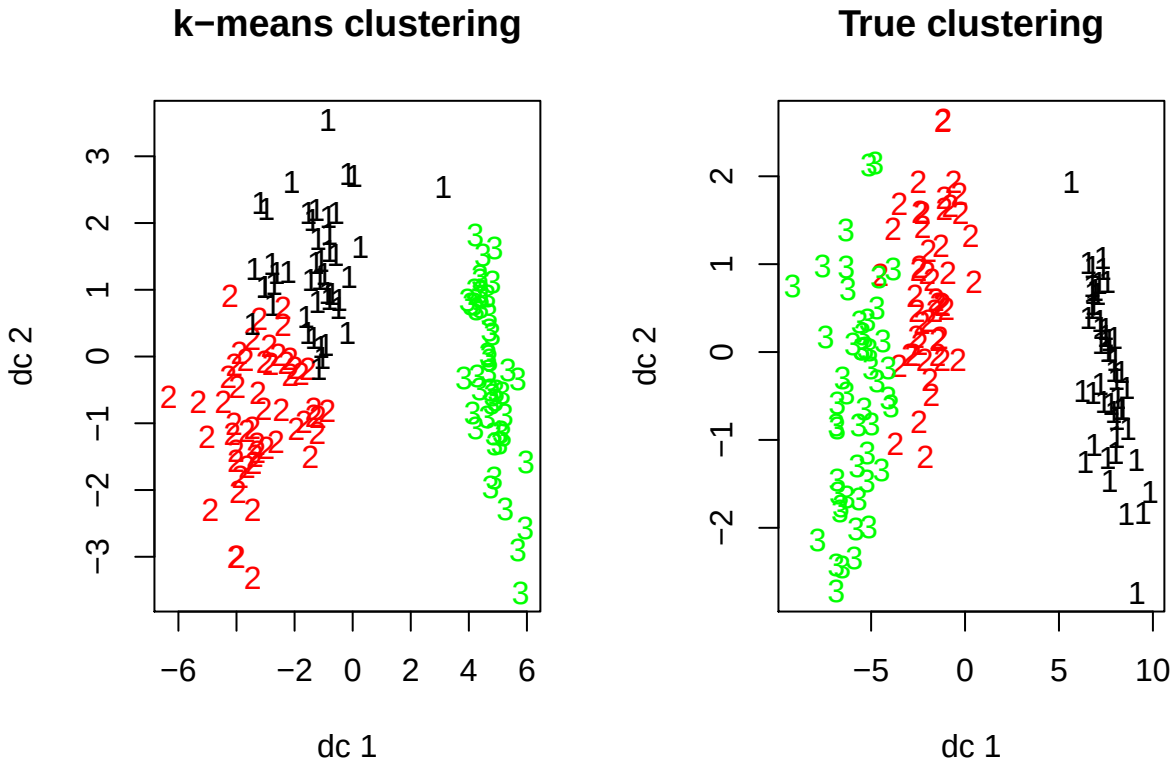
(d) Implement the algorithm using scaled "iris" data

Before we implement the k-means algorithm, the data need to be scaled by using following code:

```
data.train = as.data.frame(scale(iris[-5]))
```

Then, we can compare our result with original data.

```
cluster.index=km(data.train,3)
par(mfrow=c(1,2))
plotcluster(data.train, cluster.index,main="k-means clustering")
plotcluster(data.train, iris$Species, main="True clustering")
```



Compute the total number of errors:

```
error=sum(dist(cluster.index[1:50]))+sum(dist(cluster.index[51:100]))+sum(dist(cluster.index[101:150]))
print(error)
```

```
## [1] 915
```

The number of errors is still large. Hence scaling does not help.