Fitting kmeans clustering

Revisiting the fitting of kmeans:

```
x   <- c(3, 19, 5, 1, 12, 13, 17, 7)
out <- kmeans(x, centers = 2)

#Label
out$cluster

#Estimated center
out$centers
```

In kmeans clustering, there are two sets of parameters, class labels and the estimates of centers. We use coordinate descent to solve kmeans clustering i.e. to estimate the class labels and the centers. The objective function is given by,

$$\min_{\mathbf{c},\mu_1,\ldots,\mu_K} \sum_{k=1}^{K} \sum_{i \in A_k} (x_i - \mu_k)^2,$$

where $A_k = \{j : c_j = k\}$. The parameters are thus $\big($membership parameter ($\mathbf{c}$) of the same length as the data, cluster centers ($\boldsymbol{\mu}$, the number of clusters will be equal to the number of clusters)$\big)$

Let's take a simpler case with scalar datapoints and 2 clusters.

Specifically, let $\{3, 19, 5, 1, 12, 13, 17, 7\}$ be 8 datapoints and there are 2 centers $\mu_1 = 5$ and $\mu_2 = 15$. What will then be the idea class label? as in which values should belong to class 1 and which in class 2?

An alternative way to write the above objective function is,

$$\min_{\mathbf{c},\mu_1,\ldots,\mu_K} (x_i - \mu_{c_i})^2,$$

We apply coordinate descent to estimate $\{\mathbf{c}, \mu_1, \ldots, \mu_K\}$ or equivalently $\{c_1, \ldots, c_n, \mu_1, \ldots, \mu_K\}$.

- Update c_j: In the above objective function, c_j only appears in $(x_j - \mu_{c_j})^2$. What value of c_i minimizes this? Possible values of c_j are $\{1, \ldots, K\}$. Then the possible values of $(x_j - \mu_{c_j})^2$ are $(x_j - \mu_1)^2, \ldots, (x_j - \mu_K)^2$. We would want $(x_j - \mu_{c_j})^2 = \min_k (x_j - \mu_k)^2$. Thus $c_i = \arg\min_k (x_j - \mu_k)^2$.

- Update μ_k: Going back to previous representation of the objective function, μ_k only appears in $\sum_{i\in A_k}(x_i-\mu_k)^2$. What is the value of μ_k that minimizes this sum? Just take the derivative and equate that to zero. So, $-2\sum_{i\in A_k}(x_i-\mu_k)=0 \implies \mu_k = \frac{1}{\text{Number of data in } A_k}\sum_{i\in A_k} x_i$

Thus, we need to iterate the above two steps until convergence to get our kmeans clustering. To examine convergence, we can check whether 1) loss-function is changing or not, alternatively, 2) cluster arrangements and centers are changing or not. **Note that you can update the c_i's parallelly for all i and also μ_k's parallelly for all k.**

The above algorithm requires you to pre-specify the number of clusters K. Like any other optimization method, we need to initialize the parameters. In this case, it will be μ_k's and, subsequently, apply the updating scheme for c_i's to get the initial values of c_i's. Some people also call this an application of EM algorithm. However, it depends on how you formulate the objective function.

If the data is in more than one dimension (say of dimension p), we just make minor modification and write

$$\min_{\mathbf{c},\mu_1,\ldots,\mu_K} \sum_{k=1}^{K}\sum_{i\in A_k}\sum_{\ell=1}^{p}(x_{i,\ell}-\mu_{k,\ell})^2 = \min_{\mathbf{c},\mu_1,\ldots,\mu_K} \sum_{k=1}^{K}\sum_{i\in A_k}\|\mathbf{x}_i-\boldsymbol{\mu}_k\|_2^2,$$

the separations are replaced by squared ℓ_2 distance.

```
#Functions for updating the two set of parameters

updatec <- function(data, mu){
  datamu <- rbind(data, mu)

  dismat <- as.matrix(dist(datamu)^2)

  distmu <- dismat[1:n, (n+1):(n+K)]

  cvecnew <- apply(distmu, 1, which.min) #Getting the new cluster assignments

  return(cvecnew)
}
```

```
# Putting the code outside the function to check and debugging
# datamu <- rbind(data, mu)
#
# dismat <- as.matrix(dist(datamu)^2)
#
# distmu <- dismat[1:n, (n+1):(n+K)]
#
# cvecnew <- apply(distmu, 1, which.min) #Getting the cluster assignment

updatemu <- function(data, cvec){
  munew <- mu
  for(k in 1:K){
    munew[k, ] <- colMeans(data[which(cvec==k), ])
  }

  return(munew)
}

library(clusterSim)
stm<-shapes.two.moon(100, shape1a = -0.5, shape2b = 1)
plot(stm$data,col=rainbow(2)[stm$clusters])

data <- stm$data

K <- 2
n <- nrow(data)

tol=1e-3

mu <- data[sample(1:n, K), ] #pick any K datapoints as assign those centers.
loss <- 0
for(i in 1:1000){
  cvec <- updatec(data, mu)
  mu   <- updatemu(data, cvec)

  lossnew <- 0
  for(k in 1:K){
    lossnew <- lossnew + sum((data[which(cvec==k), ] - mu[k, ])^2)
  }
  loss[i] <- lossnew

  if(i>1){
    if(abs(loss[i-1]-loss[i])/loss[i-1] < tol) {break;}
    #The change in loss is very small, so we stop
  }
}

aricode::ARI(cvec, stm$clusters)

plot(data,col=rainbow(2)[cvec])
plot(loss)

out <- kmeans(data, centers = 2, nstart = 25)
aricode::ARI(out$cluster, stm$clusters)
```

Experiment: In the above code, experiment with `shape1a` parameter. See that when the two shapes are well separated (with larger `shape1a`), the loss declines too quickly, and we get very good clusters. However, smaller `shape1a` makes the two shapes overlap, thus the loss declines slowly, and we do not always get accurate clusters.

You can check that the result will often match with kmeans in terms of `ARI`. It may match exactly if we start our computation also from 25 different starting values and pick the best in terms of the loss.