HomeworK3

Jun Rao

8/24/2020

Download the the zip.test.gz data set.

```
#if didn't find file, download it
if(!file.exists("zip.test.gz")){
  download.file(
    
"https://web.stanford.edu/~hastie/ElemStatLearn/data.matrixs/zip.test.gz",
    "zip.test.gz"
  )
}

#if we didn't install data.table package, we install it
if(FALSE){
  install.packages("data.table")
}

#if we didn't install stats package, we install it
if(FALSE){
  install.packages("stats")
}

#if we didn't install mclust package, we install it
if(FALSE){
  install.packages("mclust")
}


#if we didn't install data.table package, we install it
if(FALSE){
  install.packages("data.table")
}


library(stats)
library(ggplot2)
library(mclust)

## Package 'mclust' version 5.4.6
## Type 'citation("mclust")' for citing this R package in publications.
```

```r
library(data.table)
library(base)
library(pdfCluster)

## pdfCluster 1.0-3

# use data.table::fread to read the compressed CSV data file into R as a data
table.
dt<- fread("zip.test.gz")

#To exclude the label column (that is the first column)
df<-as.matrix(dt[, -1])
```

Use stats::kmeans on the zip.test.gz data set. Plot the sum of squared errors (tot.withinss, within-point scatter) as a function of K, for K=1 to 20 clusters. After what K does the squared error start to look flat?

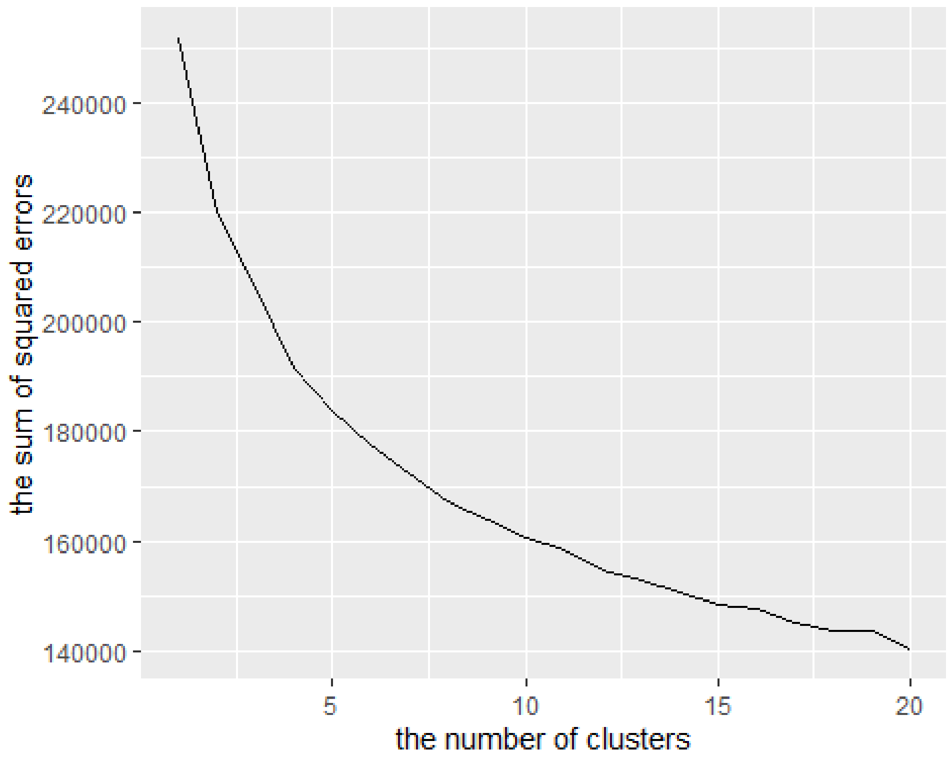
```r
#the sum of squared errors function

SOSE <-function(data.set,k){
  SOSE.dt.list <- list()

  for (index in 1:k){
    clu<-kmeans(data.set, index)
    SOSE.dt.list[[index]] = data.table("SOSE" = clu$tot.withinss, "Ncluster"
= index)
  }
  SOSE.dt <- do.call(rbind,SOSE.dt.list)
  g <- ggplot() + geom_line(aes(SOSE.dt$Ncluster,SOSE.dt$SOSE)) + xlab("the
number of clusters") + ylab("the sum of squared errors")

  return(g)
}

SOSE(df,20)
```

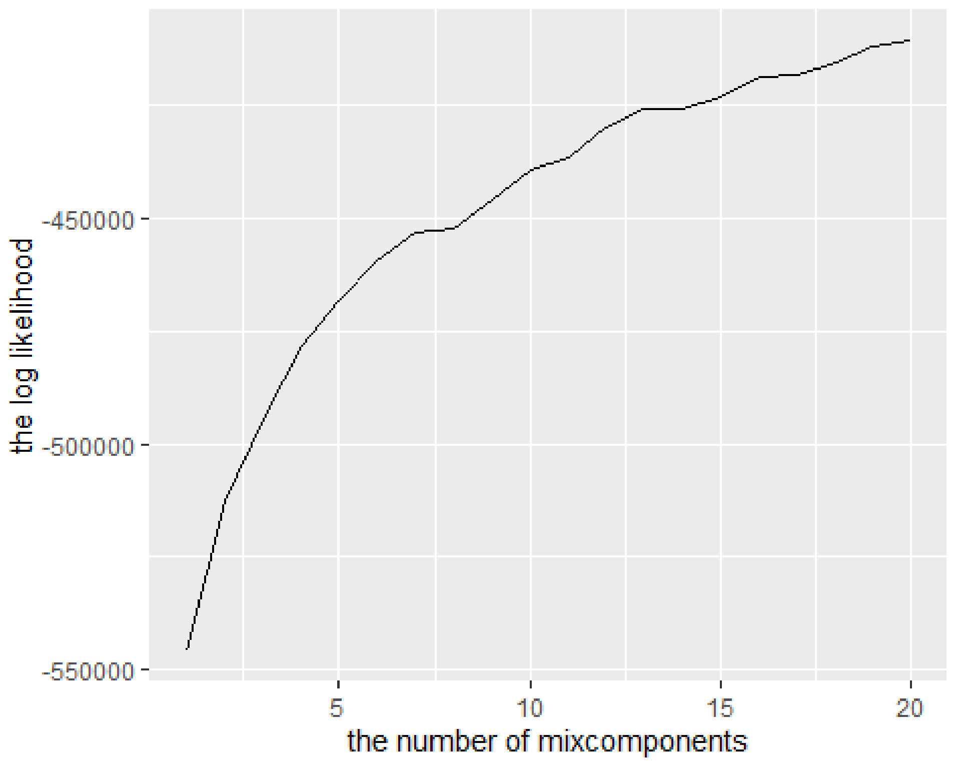


From the picture, we can find that after k =15, the squared error start to look flat.

2.Use mclust::Mclust to fit Gaussian mixture models, for G=1 to 20 mixture components. Plot the log likelihood as a function of G (number of mixture components). After what G does the log likelihood start to look flat?

```
MC <- function(data.set,G){
  MC.dt.list <- list()
   for (index in 1:G){
      mclu<-Mclust(data.set, index, initialization=list(subset=1:100),verbose
= 0,modelNames="EII")
      MC.dt.list[[index]] = data.table("loglik" = mclu$loglik, "Ncluster" =
index)
   }
   MC.dt <- do.call(rbind,MC.dt.list)
   g <- ggplot() + geom_line(aes(MC.dt$Ncluster,MC.dt$loglik)) + xlab("the
number of mixcomponents") + ylab("the log likelihood")
   return(g)
}

MC(df,20)
```



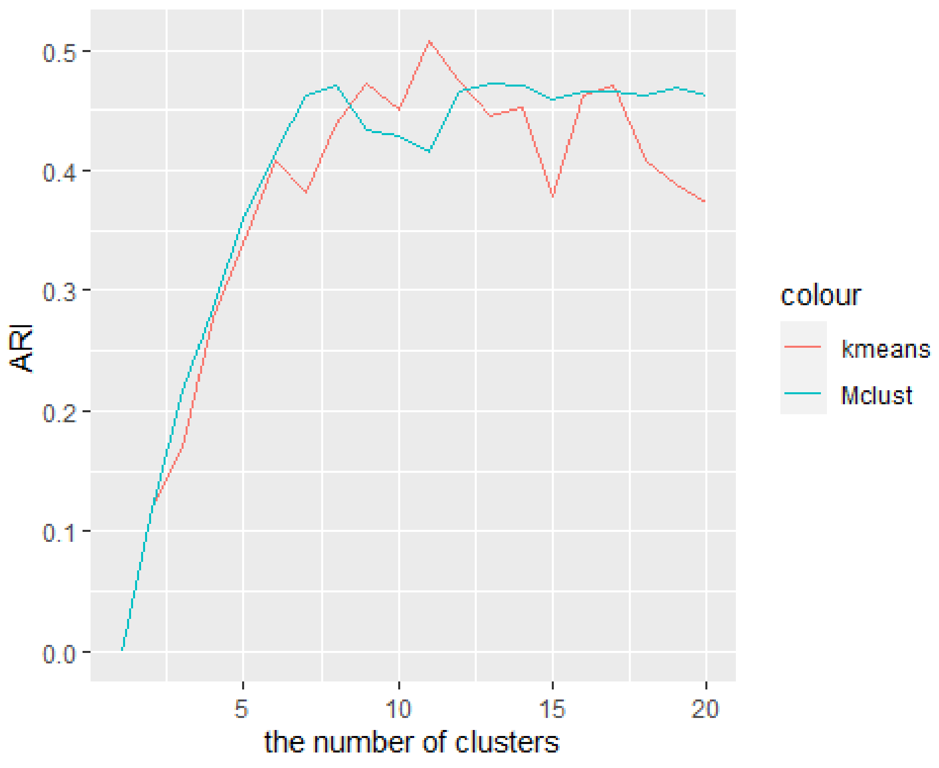
From the picture, we can find that after k =16, the log likelihood start to look flat look flat.

For both models above, use pdfCluster::adj.rand.index to compute the Adjusted Rand Index (ARI) with respect to the true class/digit labels (1 means perfect clustering and values near 0 mean random clusters). Make a plot of ARI versus number of mixture components, each algorithm in a different color (e.g. kmeans=black, Mclust=red). What is the best value of ARI that you observed? What algorithm, and how many components?

```
ARI <-function(dt,df,clusters){
  metrics.dt.list1 <- list()
  metrics.dt.list2 <- list()
  for(n.clusters in 1: clusters){
    fit1 <- kmeans(df,n.clusters)
    first.result <-
data.table(n.clusters,ARI1=pdfCluster::adj.rand.index(fit1$cluster,
dt[["V1"]]))
    fit2 <- Mclust(df,n.clusters,initialization=list(subset=1:100),verbose =
0, modelNames="EII")
    second.result <-
data.table(n.clusters,ARI2=pdfCluster::adj.rand.index(fit2$classification,
dt[["V1"]]))
    metrics.dt.list1[[paste(n.clusters)]] <- first.result
    metrics.dt.list2[[paste(n.clusters)]] <- second.result
  }
  metrics.dt1 <- do.call(rbind, metrics.dt.list1)
  metrics.dt2 <- do.call(rbind, metrics.dt.list2)

  g <- ggplot()+geom_line(aes(n.clusters, ARI1 , color
='kmeans'),data=metrics.dt1)+geom_line(aes(n.clusters, ARI2, color
='Mclust'),data=metrics.dt2) + xlab("the number of clusters") + ylab("ARI")
  return(g)
}


ARI(dt,df,20)
```



From the picture,0.475 is the best value of ARI.

And, Mclust algorithm is better than the kmeans based on the picture. And components = 15 is the best.

Code the K-means algorithm from scratch based on the pseudo-code in the textbooks. Start by taking K random data points as the initial K cluster centers (use base::sample for random selection).

Write a function KMEANS(data.matrix, K) which returns a list similar to the result of the stats::kmeans function.

```
KMEANS<-function(data=NA,k=NA){
```

```r
#we total have two parameters in the function
if(is.na(data) || is.na(k)){
    stop("You need to input valid parameters!!")
}

 #get the number of rows
 rows<-nrow(data)
 #get the number of columns
 cols<-ncol(data)

 #store the distance within the cluster
 within<-matrix(0,nrow=k,ncol=1)

 #store the distance between the cluster
 between<-0

 #the first is the cluster Index
 #the second data is the distance
 indexMatrix<-matrix(0,nrow=rows,ncol=2)

 #initialize the centers
 centers<-matrix(0,nrow=k,ncol=cols)
 #generate k random number
 randSeveralInteger<-as.vector(sample(1:rows,size=k))
 #assign the value to the centers

for(i in 1:k){
    indexMatrix[randSeveralInteger[i],1]<-i
    centers[i,]<-data[randSeveralInteger[i],]
    centers<-matrix(centers,k,cols)
}

 changeFlag=TRUE
 while(changeFlag){
    changeFlag=FALSE

    for(i in 1:rows){
      #assume the initial Distance is infinite
      initialDistance<- Inf

      previousCluster<-indexMatrix[i,1]

      for(j in 1:k){

          currentDistance<-(sum((data[i,]-centers[j,])^2))^0.5

          if(currentDistance < initialDistance){
                initialDistance<-currentDistance
                #then this data is belong to j cluster
```

```
                     indexMatrix[i,1]<-j
                     #update the distance
                     indexMatrix[i,2]<-currentDistance
              }

          }
          if(previousCluster!=indexMatrix[i,1]){
              changed=TRUE
          }

        }

        for(m in 1:k){
            clusterMatrix<-data[indexMatrix[,1]==m,]
            clusterMatrix<-as.matrix(clusterMatrix)
            if(nrow(clusterMatrix)>0){
               centers[m,]<-colMeans(clusterMatrix)
            }
             else{
                centers[m,]<-centers[m,]
            }
        }

    }###############

     ss<-function(x) sum(scale(x,scale=FALSE)^2)
     between<-ss(centers[indexMatrix[,1],])
     within<-sapply(split(as.data.frame(data),indexMatrix[,1]),ss)
     twithin<-sum(within)
    result<-
list(cluster=indexMatrix[,1],tot.withinss=twithin,betweenss=between)
    return(result)
}
```

Make a figure that compares the within-point scatter (tot.withinss element of the resulting list) for the two kmeans algorithms. Plot tot.withinss as a function of K, using two random starts for each algorithm (there should be four points plotted per K, with different algorithms in different colors). To reduce repetition in your code please use three nested for loops: (1) K=1 to 20, (2) algorithm=yours or stats::kmeans, (3) random seed=1 or 2, e.g.

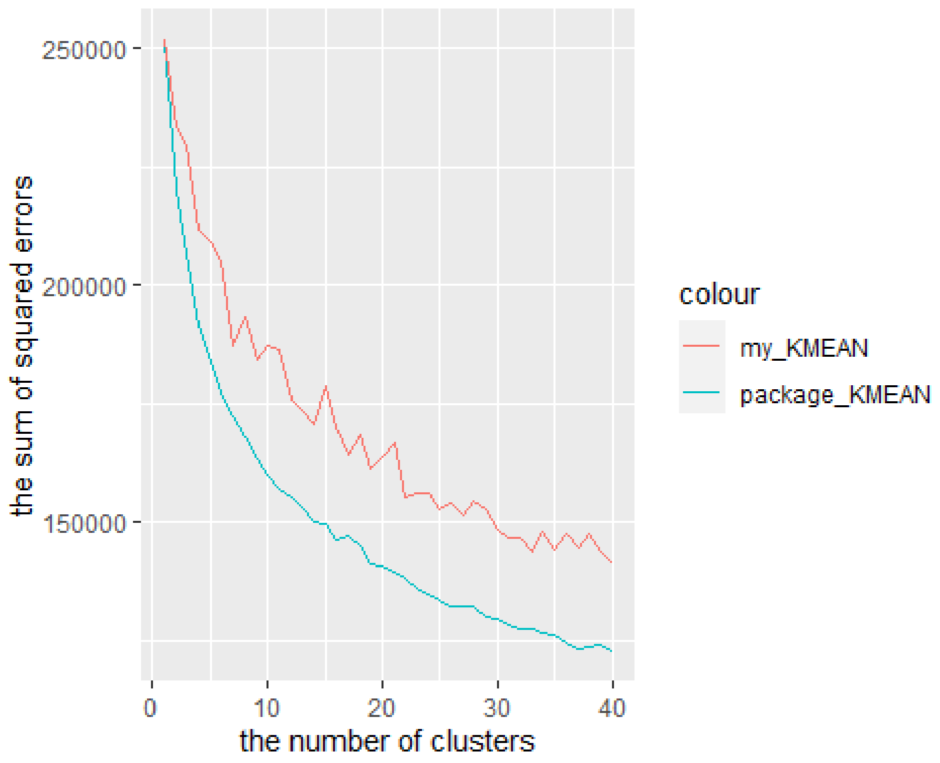
```
MyPlot <-function(data.set,k){
  Package_SOSE.dt.list <- list()
  my_SOSE.dt.list <- list()
  for (index in 1:k){
    package_clu<-kmeans(data.set, index)
    my_clu <-KMEANS(data.set, index)
    Package_SOSE.dt.list[[index]] = data.table("SOSE" =
package_clu$tot.withinss, "Ncluster" = index)
    my_SOSE.dt.list[[index]] = data.table("SOSE" =  my_clu$tot.withinss,
"Ncluster" = index)
```

```
  }
  package_SOSE.dt <- do.call(rbind,Package_SOSE.dt.list)
  my_SOSE.dt <- do.call(rbind,my_SOSE.dt.list)
  g <- ggplot() +
geom_line(aes(package_SOSE.dt$Ncluster,package_SOSE.dt$SOSE,
color='package_KMEAN')) + geom_line(aes(my_SOSE.dt$Ncluster,my_SOSE.dt$SOSE,
color='my_KMEAN'))+ xlab("the number of clusters") + ylab("the sum of squared
errors")

  return(g)
}



MyPlot(df,40)
```


the sum of squared errors
the number of clusters
colour
my_KMEAN
package_KMEAN