Homework4

Jun Rao

8/31/2020

Comparing hierarchical clustering with kmeans

```
library(stats)
library(ggplot2)
library(ggpubr)
library(data.table)
library(microbenchmark)
library(ggdendro)
# 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])
```

1.First use stats::kmeans on the zip.test data set to compute a clustering with K=10 clusters. Using base::system.time, how much time does it take on your system?

```
KMT <- function(data.set, g){
    start_time <- Sys.time()
    kclu<-kmeans(data.set, g)
    end_time <- Sys.time()
    total_time = end_time - start_time
    return(total_time)
}
KMT(df,10)
```

```
## Time difference of 0.1745679 secs
```

2.Use stats::dist and stats::hclust to compute a hierarchical clustering tree on different sized subsets of the zip.test data set (start with a small size like 100 and increase), then use stats::cutree to compute 10 clusters. Use base::system.time to compute timings of the entire process for each subset size. What is the largest subset / number of observations you can cluster in about the same amount of time as kmeans on the whole data set? (on my system it is about 600, please show the code and timings output on your system)

```
HCT <- function(data.set, g){
    iter <- seq(100,1000,100)
```

```
    for (size in iter) {
      start_time <- Sys.time()
      dist.mat <- dist(data.set[1:size,],method = "manhattan")
      as.matrix(dist.mat)
      cl.tree<-hclust(dist.mat)
      cutree(cl.tree, g)
      end_time <- Sys.time()
      total_time = end_time - start_time
      print(paste0("When the size is ", size, " the total time cost are: ", t
otal_time))
    }


}
HCT(df,10)
```

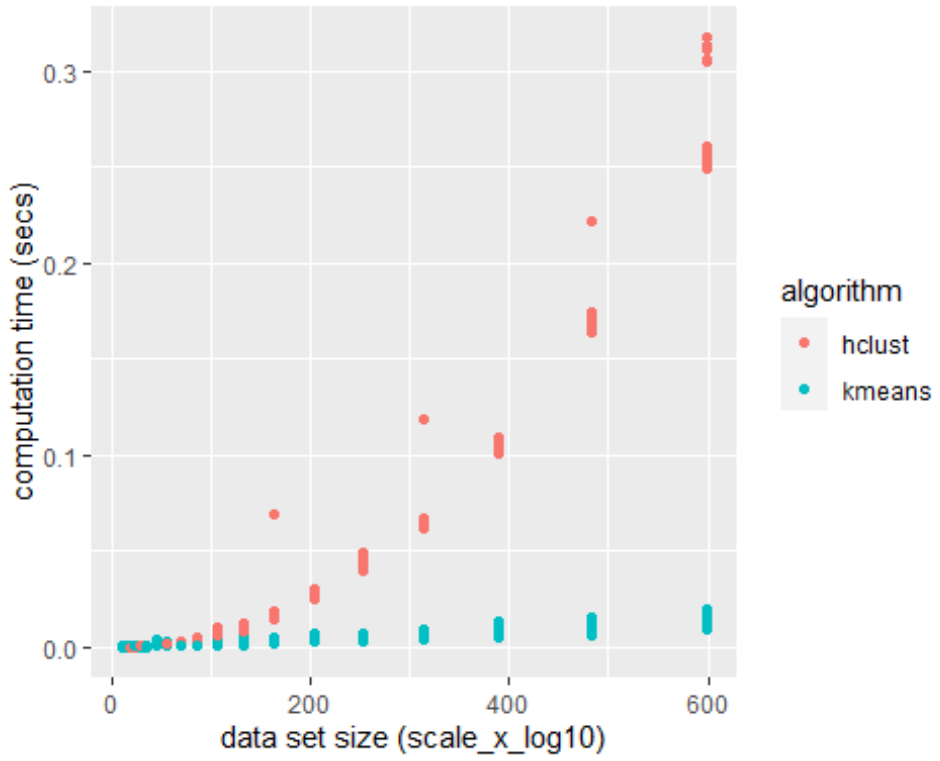
```
## [1] "When the size is 100 the total time cost are: 0.00598287582397461"
## [1] "When the size is 200 the total time cost are: 0.0289289951324463"
## [1] "When the size is 300 the total time cost are: 0.0628299713134766"
## [1] "When the size is 400 the total time cost are: 0.114689111709595"
## [1] "When the size is 500 the total time cost are: 0.18451189994812"
## [1] "When the size is 600 the total time cost are: 0.250330924987793"
## [1] "When the size is 700 the total time cost are: 0.347073078155518"
## [1] "When the size is 800 the total time cost are: 0.540549039840698"
## [1] "When the size is 900 the total time cost are: 0.579733848571777"
## [1] "When the size is 1000 the total time cost are: 0.714091062545776"
```

The largest number in my result is 500, we can you can cluster in about the same amount of time as kmeans on the whole data set

3.Use microbenchmark::microbenchmark to compute timings for both algorithms, and several zip.test data subset sizes. Make a plot of computation time versus data set size, with each algorithm in a different color (e.g. hclust=red, kmeans=black). What does the plot suggest about the time complexity of each algorithm? (write time complexity in big-O notation in terms of N the data set size) use a for loop over data set sizes, on a log scale, from 10 to the subset size you found in problem 1. e.g. 10^seq(1, log10(600), l=5). use the list of data tables idiom and during each iteration of the for loop store a data table with timings from microbenchmark. use microbenchmark(times=3) to run each algo only three times (instead of the default 100). use scale_x_log10() and scale_y_log10() in your ggplot.

```
Myplot <- function(data.set){
    time.dt.list <- list()
```

```
    (log.scale.seq <- as.integer(c(10^seq(1, log10(600), l=20))))
    # seq(10, 1000, by =100)
    for (N in log.scale.seq) {
        X <- data.set[1:N,]
        timing.df <- microbenchmark::microbenchmark(hclust={
            d.mat <- stats::dist(X, method="manhattan")
            as.matrix(d.mat)
            cl.tree <- stats::hclust(d.mat, method="single")
            stats::cutree(cl.tree, k=3)
            }, kmeans={
                  stats::kmeans(X, 3)
          })
          time.dt.list[[paste(N)]] <- data.table(N, timing.df)
    }
    time.dt <- do.call(rbind, time.dt.list)
    time.dt[, data.size := N]
    time.dt[, time.seconds := time/1e9]
    time.dt[, algorithm := expr]

    g<-ggplot()+
      geom_point(aes(x=data.size,y=time.seconds,color=algorithm),data=time.dt
)+
      xlab("data set size (scale_x_log10)") + ylab("computation time (secs)")


    return(g)
}


Myplot(df)
```



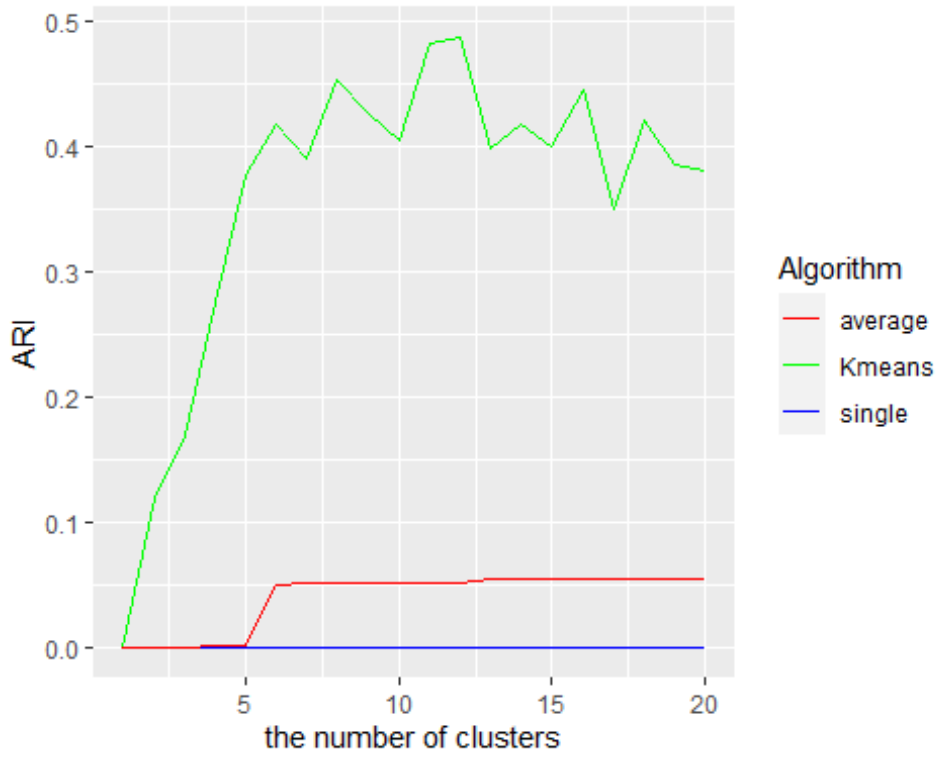
From the picture, we can conclude that:

1. *the Kmeans is a linear O(N) algorithm*
2. *the hclust is a cubic O(N^3) algorithm*

4.Compare two linkage methods (e.g. single and average, if one of those two does not work for some reason, try another one), for cluster sizes from 1 to 20, on the zip.test data set, in terms of Adjusted Rand Index. Make a plot of ARI versus number of clusters, each linkage method in a different color (e.g. single=black, average=red). What is the best value of ARI that you observed? What linkage method, and how many clusters? To save time use the same data subset size you found in problem 1. 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).

```
ARI <-function(dt,df,cluster_sizes){
  metrics.dt.list1 <- list()
  metrics.dt.list2 <- list()
  metrics.dt.list3 <- list()
  for(n.clusters in 1: cluster_sizes){
    dist.mat <- dist(df,method = "manhattan")
```

```
    as.matrix(dist.mat)
    cl.tree1 <- hclust(dist.mat, method="single")
    gr.vector1<-cutree(cl.tree1, n.clusters)
    first.result <- data.table(n.clusters,ARI1=pdfCluster::adj.rand.index(gr.
vector1, dt[["V1"]]))
    cl.tree2 <- hclust(dist.mat, method="average")
    gr.vector2<-cutree(cl.tree2, n.clusters)
    second.result <- data.table(n.clusters,ARI2=pdfCluster::adj.rand.index(gr
.vector2, dt[["V1"]]))
    fit1 <- kmeans(df,n.clusters)
    third.result <- data.table(n.clusters,ARI3=pdfCluster::adj.rand.index(fit
1$cluster, dt[["V1"]]))
    metrics.dt.list1[[paste(n.clusters)]] <- first.result
    metrics.dt.list2[[paste(n.clusters)]] <- second.result
    metrics.dt.list3[[paste(n.clusters)]] <- third.result
  }
  metrics.dt1 <- do.call(rbind, metrics.dt.list1)
  metrics.dt2 <- do.call(rbind, metrics.dt.list2)
  metrics.dt3 <- do.call(rbind, metrics.dt.list3)

  g <- ggplot()+
       geom_line(aes(n.clusters, ARI1 , color ='single'),data=metrics.dt1)+
       geom_line(aes(n.clusters, ARI2, color ='average'),data=metrics.dt2)+
       geom_line(aes(n.clusters, ARI3, color ='Kmeans'),data=metrics.dt3) +
       xlab("the number of clusters") + ylab("ARI")+
       scale_color_manual(values = c("red", "green", "blue"), name = 'Algorit
hm')


  return(g)
}


ARI(dt,df,20)
```



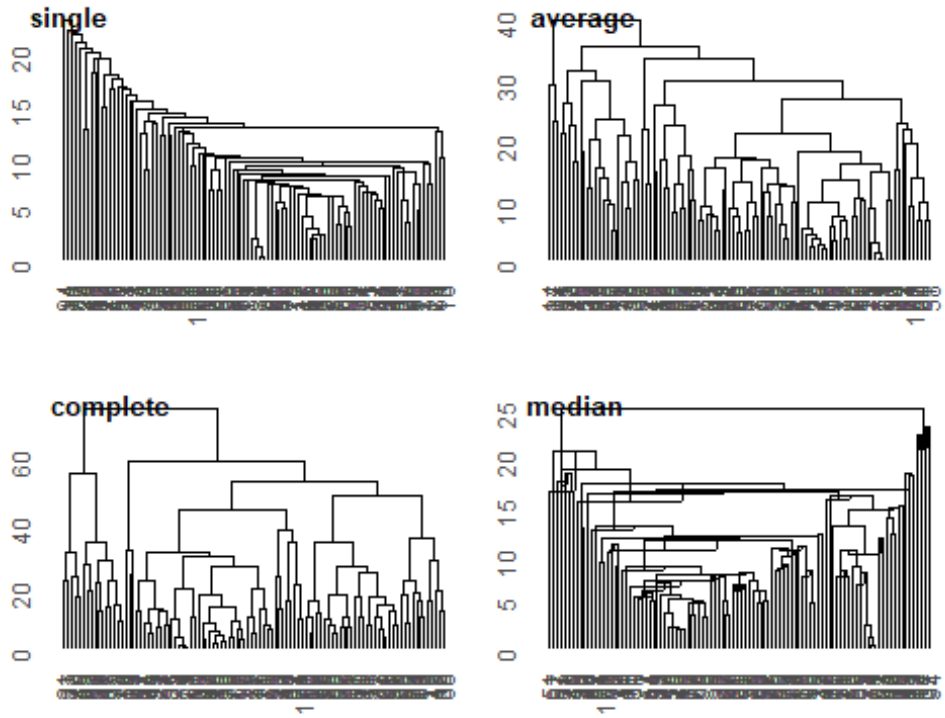
It looks like Kmeans function is the best for this data. The best value we observed is 0.48 around 12 clusters.

In hclust algorithm,thw average mthod is better, the better ARI value is 0.05, the cluster is 6.

Extra credit (10 points): use the ggdendro package to plot the dendrograms from problem 4 in a multi-panel ggplot (one panel for each linkage method). How similar are the two cluster trees?

```
X = df[1:100,1:50]

hc1 <- hclust(dist(X ,method="manhattan"), "single")
p1 <- ggdendrogram(hc1, rotate = FALSE, size = 2)

hc2 <- hclust(dist(X ,method="manhattan"), "average")
p2 <- ggdendrogram(hc2, rotate = FALSE, size = 2)

hc3 <- hclust(dist(X ,method="manhattan"), "complete")
p3 <- ggdendrogram(hc3, rotate = FALSE, size = 2)

hc4 <- hclust(dist(X ,method="manhattan"), "median")
p4 <- ggdendrogram(hc4, rotate = FALSE, size = 2)


ggarrange(p1, p2,p3,p4,
```

```
          labels = c("single", "average", "complete", "median"),
           font.label = list(size = 10),
           legend =c("top","right"),
          common.legend = TRUE,
           ncol = 2, nrow = 2)
```



Code the Hierarchical clustering algorithm from scratch based on the pseudo-code in the textbooks. Start by computing the pairwise distance matrix, then recursively join observations/groups until you have the desired number of clusters.

Write a function HCLUST(data.matrix, K) which computes the clustering. For simplicity, do NOT compute the entire tree; instead your function should return an integer vector of cluster IDs (size=number of observations, entries=from 1 to K).

use single linkage method (distance from a group to an observation is the minimum distance over all points in that group). evaluate the accuracy of your result by running your algorithm alongside hclust/cutree on the iris data. Compute/print a contingency/count table, e.g. base::table(ids.from.hclust.cutree, ids.from.your.HCLUST)… does your algorithm compute the same clustering?

```
HCLUST <- function(data.matrix, K)
{

  i=seq(1,K)
  data.matrix=as.matrix(data.matrix[i,])
  Distcance.Matrix = dist(data.matrix)
```

```
  if(!is.matrix(Distcance.Matrix)){
    Distcance.Matrix = as.matrix(Distcance.Matrix)

  }


  Nrows = nrow(Distcance.Matrix)
  diag(Distcance.Matrix)=Inf
  # Tracks group index
  group_index = -(1:Nrows)
  # hclust matrix result output
  output = matrix(0,nrow=Nrows-1, ncol=2)
  # hclust height output
  height_output = rep(0,Nrows-1)
  for(j in seq(1,Nrows-1))
  {
    # Find the minimum distance over all points in that group
    height_output[j] = min(Distcance.Matrix)
    # get exactly a 0 value.
    i = which(Distcance.Matrix - height_output[j] == 0, arr.ind=TRUE)
    # get more than one, and  merge one pair.
    i = i[1,,drop=FALSE]
    p = group_index[i]
    #to order each m[j,] pair as follows:
    p = p[order(p)]
    output[j,] = p
    # Agglomerate this pair and all previous groups they belong to
    # into the current jth group:
    grp = c(i, which(group_index %in% group_index[i[1,group_index[i]>0]]))
    group_index[grp] = j
    # Concoct replacement distances that consolidate our pair using `method`:
    r = apply(Distcance.Matrix[i,],2,"min")
    # Move on to the next minimum distance, excluding current one by modifyin
g
    # the distance matrix:
    Distcance.Matrix[min(i),] = Distcance.Matrix[,min(i)] = r
    Distcance.Matrix[min(i),min(i)]        = Inf
    Distcance.Matrix[max(i),] = Distcance.Matrix[,max(i)] = Inf
  }
# Return something similar to the output from hclust.

  structure(list(merge = output, height = height_output))
}


K <- 40
data.matrix <-iris[,1:4]
```

```
h1=HCLUST(data.matrix,K)

data.matrix <- as.matrix(data.matrix[seq(1,K),])

h=hclust(dist(data.matrix),method="single")

print(cbind(h$merge,h1$merge))
```

```
##       [,1] [,2] [,3] [,4]
##  [1,]   -8  -40  -40   -8
##  [2,]   -1  -18  -18   -1
##  [3,]  -10  -35  -35  -10
##  [4,]   -5  -38  -38   -5
##  [5,]  -20  -22  -22  -20
##  [6,]    2    4    2    4
##  [7,]  -30  -31  -31  -30
##  [8,]   -9  -39  -39   -9
##  [9,]    1    6    1    6
## [10,]   -2    3   -2    3
## [11,]  -28  -29  -29  -28
## [12,]  -13   10  -13   10
## [13,]    9   11    9   11
## [14,]    7   12    7   12
## [15,]  -26   14  -26   14
## [16,]   -4   15   -4   15
## [17,]  -24  -27  -27  -24
## [18,]  -12   16  -12   16
## [19,]   13   17   13   17
## [20,]   18   19   18   19
## [21,]    5   20    5   20
## [22,]  -14    8  -14    8
## [23,]   -3   21   -3   21
## [24,]   -7   23   -7   23
## [25,]  -21  -32  -32  -21
## [26,]  -11   24  -11   24
## [27,]   22   26   22   26
## [28,]  -25   27  -25   27
## [29,]  -36   28  -36   28
## [30,]  -37   29  -37   29
## [31,]   25   30   25   30
## [32,]   -6  -19  -19   -6
## [33,]   31   32   31   32
## [34,]  -17   33  -17   33
## [35,]  -33  -34  -34  -33
## [36,]  -16   35  -16   35
## [37,]   34   36   34   36
## [38,]  -15   37  -15   37
## [39,]  -23   38  -23   38
```