Introduction to Clustering

STA 325: Lab 5, Fall 2018

Today's agenda: Introduction to the Clustering

Programming partner's: You should have a programming partner for each lab, and you should switch off who is programming, and use each other for help. We will spend about 30–50 minutes per week on lab exercises and you will be expected to bring you laptops to class to work on these exercises in class. Myself and the TA's will be in class to help you.

First we load the package `clusterCrit`, which contains the function `intCriteria`.

```
library(clusterCrit)
```

Next we read in the data `Wimbledon.csv`, setting `stringsAsFactors = FALSE`.

```
myTennisDataD <- read.csv("Wimbledon.csv", stringsAsFactors = FALSE)
```

To clean the data, we remove the first two columns.

```
myTennisData2 <- myTennisDataD[, (-1):(-2)]
```

We now use these two columns to create the row names for the `myTennisData2`. If column 1 was `Player1` and column 2 was `Player2`, we create the new row name `Player1+Player2`.

```
row.names(myTennisData2) <- paste(myTennisDataD[,1], # name of first player
                                  myTennisDataD[,2], # name of second player
                                  sep = "+") # join with "+"
```

1. We now remove any columns containing only `NA` values. The `apply` function can be used to return a vector which is `TRUE` if a column contains only `NA` values and `FALSE` otherwise. We then subset on the logical inverse of this vector, retaining the columns that are not completely `NA` values.

```
na.columns <- apply(myTennisData2, 2, # apply to the columns of myTennisData2
                    function(x){all(is.na(x))}) # TRUE if all NA
iFinalTennisData <- myTennisData2[,!na.columns] # ! means TRUE <--> FALSE
```

Next we replace the remaining `NA` values with the median of their columns.

```
for (i in 1:ncol(iFinalTennisData)){
  iFinalTennisData[is.na(iFinalTennisData[,i]), i] <- # NA values in column i get
        median(iFinalTennisData[,i], na.rm = TRUE) # median, removing NA values
}
```

2. Using the cleaned data frame we create a log-Euclidean distance matrix for the data using the `dist` function.

```
my.distances <- log(dist(iFinalTennisData))
```

With these distances we can perform single linkage and complete linkage clustering on the data using the `hclust` function.

```
singleLinkage <- hclust(my.distances, method = "single") # single linkage
completeLinkage <- hclust(my.distances, method = "complete") # complete linkage
```

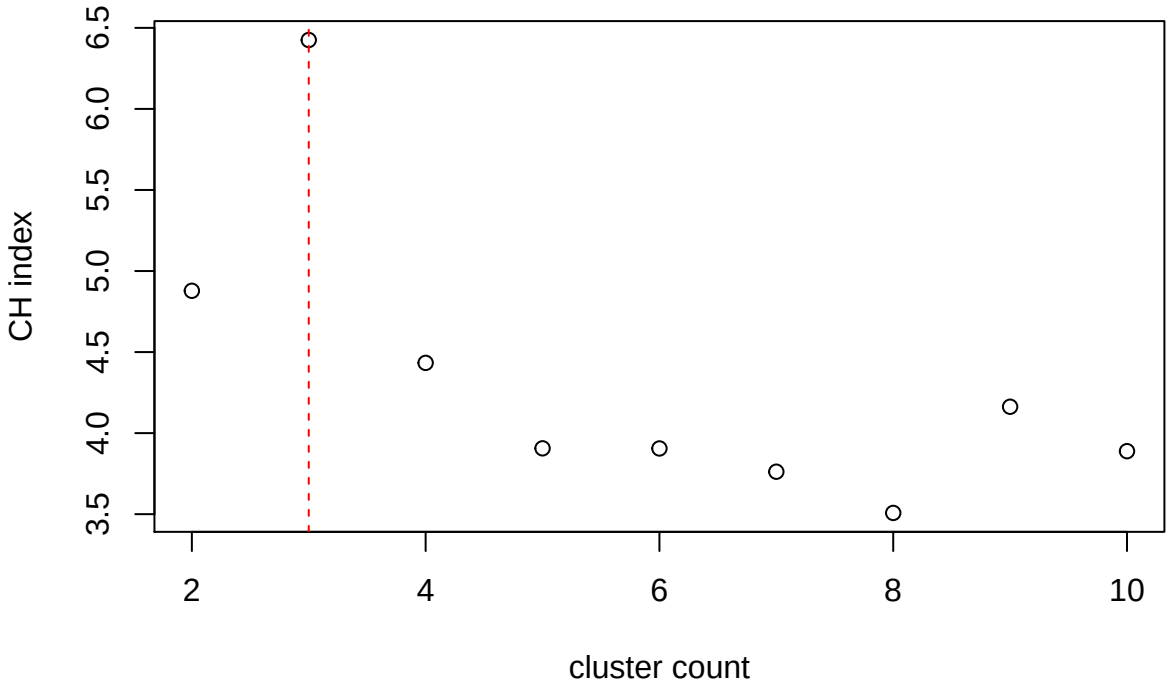
3. We know write a function `computeCH` that computes the CH Index for all cluster sizes smaller than a given value `num.clust` for a given clustered object `clust.obj` and data `data.set`. The function will compute the CH values using the `intCriteria` function. Once all of the CH indices have been computed, the function plots them, indicating the maximum value with dashed red line. The function

then returns the maximum CH index. All of the shifting by 1 in the code is to account for the fact that the CH index is not defined for $k = 1$.

```
computeCH <- function(num.clust, clust.obj, data.set){
#######################################################
# This function creates a plot of CH indices
#  and returns the largest CH index
# when the number of clusters is less than a given value
#
# Inputs:
#   num.clust:  a positive integer,
#   will test all cluster numbers up to this
#   clust.obj:  a cluster object that can be cut into various
  # clusters
#   data.set:  a data frame containing data that generated clust.obj
#
# Output:
#   A plot of the CH indices
#   The largest CH index for # of clusters below num.clust
#######################################################
  CH.values <- vector(length = (num.clust - 1)) # - 1 b/c no CH for k = 1
  for (i in 2:num.clust){ # we start a 2, not defined for 1
    clustering <- cutree(clust.obj, k = i) # divide into i clusters
    CH.values[i-1] <- intCriteria(as.matrix(data.set), # for input data
                                  clustering, # use i clusters
                                  crit = "Calinski_Harabasz") # CH index
  }
  max.CH <- max(unlist(CH.values)) # find the max value
  plot(2:num.clust, # x values from 2 to num.clust
       CH.values, # y values are CH indices
       xlab = "cluster count", # name x axis
       ylab = "CH index") # name y axis
  abline(v = # place a vertical line
           ((which.max((unlist(CH.values)))) + 1), # at x = index of max +1
         # we need +1 because x begins from 2, not 1
           col = "red", # line is red
           lty = 2) # line is dashed
  return(max.CH) # return max CH value
}
```

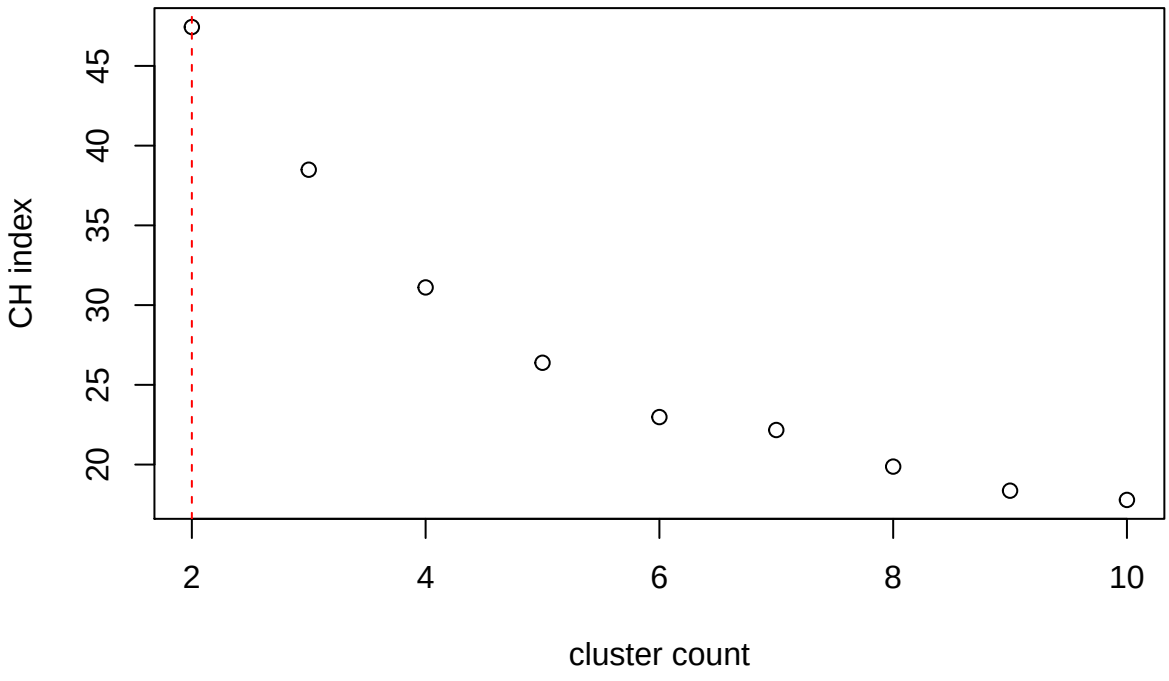
Now we run the function with `num.clust = 10` for both single linkage (`singleLinkage`) and complete linkage (`completeLinkage`) for the data set `iFinalTennisData`.

```
computeCH(10, singleLinkage, iFinalTennisData) # single linkage
```




```
## [1] 6.425097
```

```
computeCH(10, completeLinkage, iFinalTennisData) # complete linkage
```




```
## [1] 47.4376
```

We see that the maximum CH index for single linkage occurs is 6.425, which occurs when there are 3 clusters. For complete linkage, the maximum CH index is 47.438, which occurs when there are 2 clusters. The CH index is not defined when there is only one cluster; the CH index is the ratio of the between cluster variation to the in cluster variation. When there is only one cluster, there concept of between cluster variation is not defined, so the ratio does not exist. This can be confirmed by running the `intCriteria` function with only

one cluster.

```
intCriteria(as.matrix(iFinalTennisData), cutree(singleLinkage, k = 1),
            crit = "Calinski_Harabasz")
```

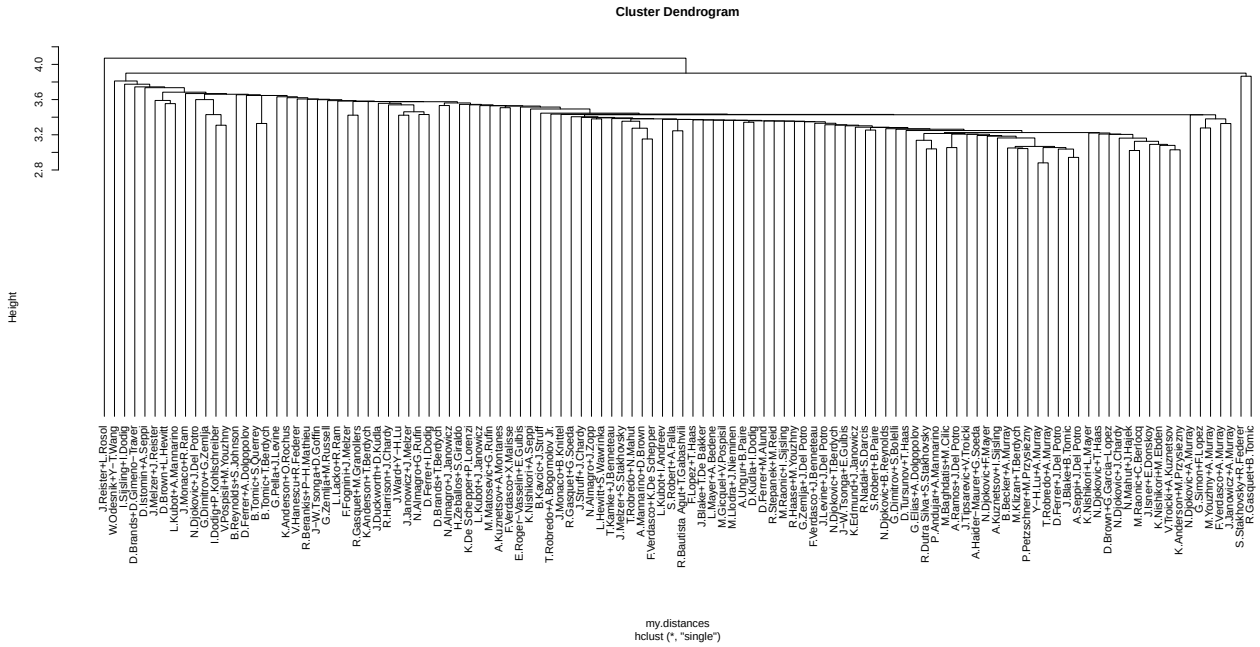
```
## $calinski_harabasz
## [1] NaN
```

The result is `NaN`, which stands for "not a number."

4. The optimal number of clusters corresponds to the number of clusters with the largest CH index. For single clustering, this is $k = 3$. For complete clustering, it is $k = 2$.

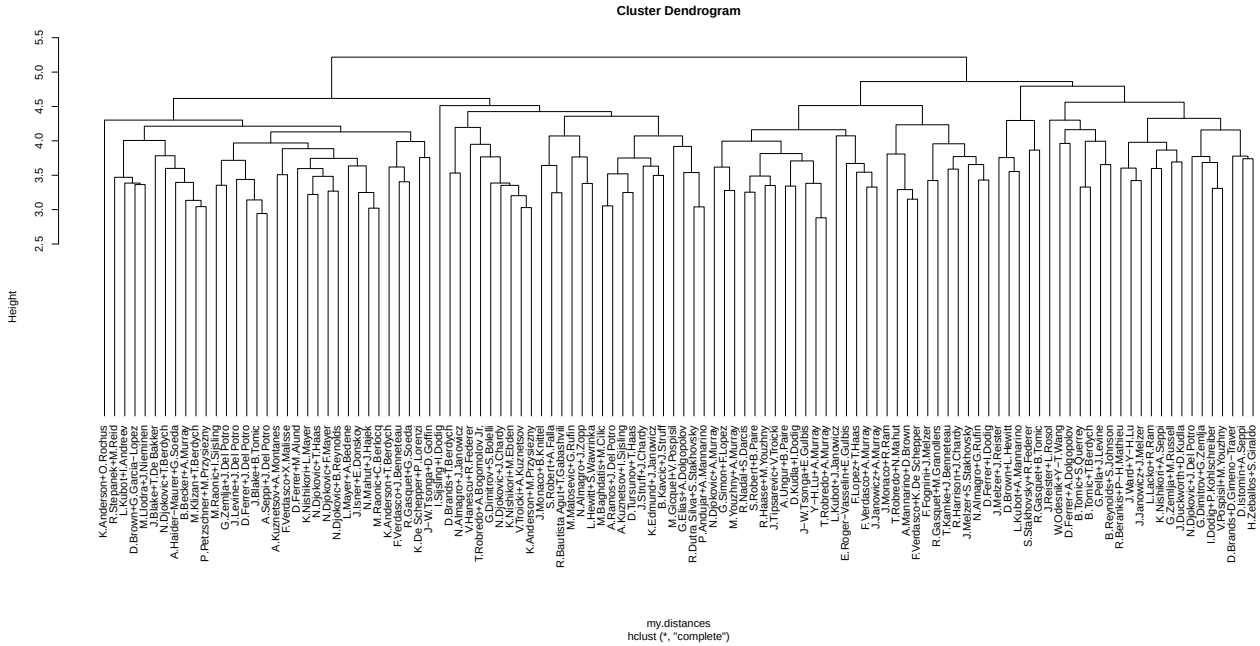
5. First we plot the dendogram for single linkage.

```
plot(singleLinkage, hang = -1e-10)
```



We now plot the dendogram for complete linkage

```
plot(completeLinkage, hang = -1e-10)
```



Each point in the data set describes the results of a tennis match. Tennis matches have a fairly rigid structure in the sense that each match consists of a maximum number of sets, each of which consists of games. If we use the optimal number of clusters as determined by CH indices for single linkage, we get two very small clusters (one of one element, one of two) and one very large cluster. This would suggest that the games in the smaller clusters are radically different from the rest of the games. It is hard to imagine why this would occur. For complete linkage, dividing into 2 clusters (as determined by CH indices) gives a relatively even split of the matches, which seems much more reasonable (especially given knowledge of how the games/matches are played).