**Cluster Analysis (CA)**

**What is CA?**

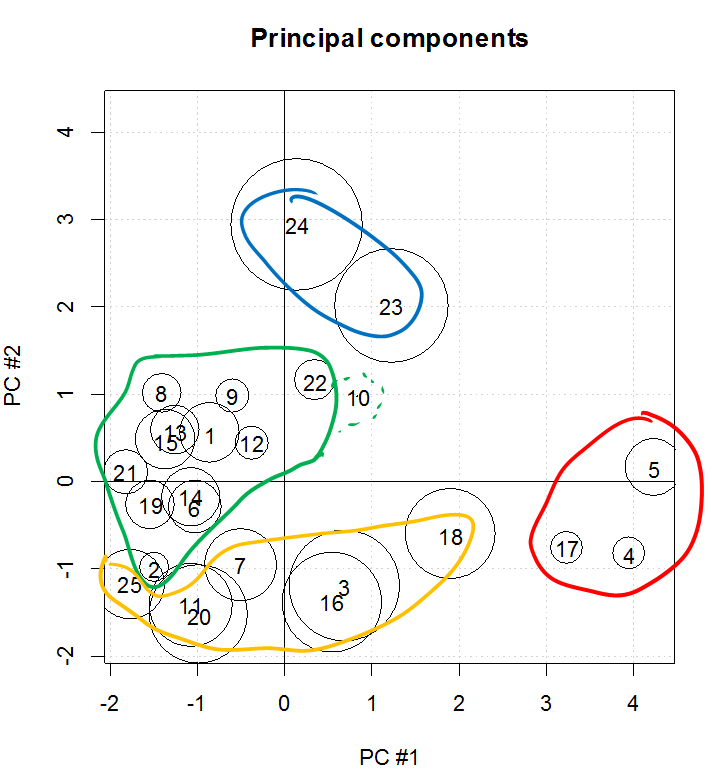
The purpose is to develop a formal classification method to classify observations into groups (a.k.a., clusters). These groups contain observations that are “similar” to each other.

**No prior groups of the observations are known before the cluster analysis**. Because of this, some non-statisticians will refer to cluster analysis as an “unsupervised learning” method. In the next section of the course, the observations will be known to belong to particular groups (e.g., placekicking data – each observation is a success or failure). Some non-statisticians will refer to the corresponding statistical methods as “supervised learning” methods.

There are a lot of CA methods! For example, CRAN has a “task view” set up at [http://cran.r-project.org/web/  
views/Cluster.html](http://cran.r-project.org/web/views/Cluster.html) that details all the packages available for CA. There are also whole textbooks on cluster analysis.

Example: Goblet data

The PCA resulted in the following plot of the first two principal components with informal groupings:



Cluster analysis examines many formal ways to do this grouping. Note that PC scores are not needed for cluster analysis overall. The PC scores are just used here to help motivate the problem in a two-dimensional space.

Example: Two variable data set (TwoVariable.R)

Suppose two variables, x1 and x2, have 14 observed pairs of values (see program for code):



One possible set of natural groupings is shown in the plot below. These groupings are decided upon by placing observations that are close to each other in a group. How is closeness measured? This will be discussed soon!





Another possible grouping results from combining the red and blue into one group and the purple, green, and yellow into one group. An issue then that we will often encounter is determining how many clusters are appropriate for a data set. There will often be more than one justifiable answer!

**Measures of similarity (or dissimilarity)**

How can we determine which observations are “similar”?

* Euclidean distance (ruler distance)



* Standardized Euclidean distance



Why is this a better measure than Euclidean distance in most situations?

Using these measures, we can decide which observations are similar enough to be put into a cluster.

These measures of similarity can be calculated by the dist() function. Many other measurements of similarity are available through this same function as well. If you are interested in seeing what is available, please see the descriptions given in the help file.

**Clustering Methods**

There are three types of clustering methods:

* Hierarchical

Observed data points are grouped into clusters in a nested sequence of clusterings. There are two different directions for this nested sequence:

Agglomerative: Each observation begins as a separate cluster. Clusters are successively merged until only one cluster is present. Ideally, the algorithm would stop some time before just one cluster is present.

Divisive: Each observation starts in one overall cluster. New clusters are successively formed by splitting larger clusters. The algorithm continues until all observations are in separate clusters (i.e., N clusters). Ideally, the algorithm would stop some time before every observation is in its own cluster. R functions that can implement it include diana() from the cluster package. We will not discuss the divisive type of hierarchical clustering in our class.

* Nonhierarchical (partitional)

An initial number of clusters is pre-specified so that each observation is categorized into one cluster. Observations are iteratively reallocated between clusters until an “optimal” solution is found.

* Model-based

An underlying statistical model is assumed such that it led to a specific number of clusters. The mclust package in R provides functions to implement these methods.

**Agglomerative clustering**

Although agglomerative methods continue until there is only one cluster, you typically don’t want only one cluster (if you did, what was the purpose of using CA then?). One of the main topics to be discussed in this section is how to decide on the number of clusters. In other words, when should you stop the clustering process?

There are many agglomerative methods because no one method is best. Next, I will discuss a few of the most commonly used methods.

Nearest neighbor method (single linkage)

Here’s the process:

1. Start with N clusters where each cluster is one of the observations
2. Merge the two clusters that are the “nearest” to each other
3. Define the distance between this new cluster and the other observations as the minimum distance between the two points in the cluster and the other observations
4. Put the two nearest clusters into a new cluster
5. Continue this process of making clusters

This process continues until all observations are within one cluster. The optimal choice for the number of clusters is somewhere between 1 and N. Although this process continues until 1 cluster is found, the actual number used should almost always be greater than 1.

The single linkage name comes from it taking essentially just one observation per cluster to decide whether or not to link two clusters.

Example: Clustering for 4 points (DistanceExample.R)

Below are 4 observations:

|  |  |  |
| --- | --- | --- |
| **Observation** | **x1** | **x2** |
| 1 | 2 | 3.4 |
| 2 | 4 | 5 |
| 3 | 6 | 3 |
| 4 | 4 | 2 |

The observations are plotted below and the Euclidean distances between each are found (remember that standardized Euclidean distances are typically used in practice). For example, the distance between observation #1 and observation #2 is



This can be found in R using the following commands:

> x1 <- c(2, 3.4)

> x2 <- c(4, 5)

> sqrt(t(x1 - x2) %\*% (x1 - x2))

[,1]

[1,] 2.56125

Alternatively, the dist() function can find all possible distances:

> set1 <- data.frame(var1 = c(2, 4, 6, 4), var2 = c(3.4, 5,

3, 2))

> dist.mat <- dist(x = set1, method = "euclidean")

> dist.mat

1 2 3

2 2.561250

3 4.019950 2.828427

4 2.441311 3.000000 2.236068

I will need to extract some distances from dist.mat shortly. The best way to do this is to convert dist.mat to a matrix class type:

> dist.mat <- as.matrix(dist.mat)

> dist.mat

1 2 3 4

1 0.000000 2.561250 4.019950 2.441311

2 2.561250 0.000000 2.828427 3.000000

3 4.019950 2.828427 0.000000 2.236068

4 2.441311 3.000000 2.236068 0.000000

> dist.mat[1,2]

[1] 2.56125

Below is a plot showing the distances between all of the points.

> par(pty = "s")

> plot(x = set1$var1, y = set1$var2, xlab = "Variable #1",

ylab = "Variable #2", pch = 16, col = "blue", cex = 5,

xlim = c(0,7), ylim = c(0,7), panel.first = grid())

> #Draw all of the distances on the plot!

> for (i in 1:3) {

for (j in (i+1):4) {

segments(x0 = set1$var1[i], y0 = set1$var2[i], x1 =

set1$var1[j], y1 = set1$var2[j], lty = "solid", lwd

= 1, col = 2)

text(x = (set1$var1[i]+set1$var1[j])/2, y =

(set1$var2[i]+set1$var2[j])/2, labels =

round(dist.mat[i,j], 2))

}

}

> text(x = set1$var1, y = set1$var2, labels = 1:4, col =

"white", cex = 2)



Steps of the nearest neighbor method:

1. Start with N clusters where each cluster is one of the observations

There are 4 clusters

1. Merge the two clusters that are the “nearest” to each other

Observations #3 and #4 are the “nearest” so they form a new cluster.

1. Define the distance between this new cluster and the other observations as the minimum distance between the two points in the cluster and the other observations.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **1** | **2** | **3,4** |
| 1 |  | 2.56 | min(4.02, 2.44) = 2.44 |
| 2 |  |  | min(2.83, 3.00) = 2.83 |

1. Put the two nearest clusters into a new cluster.

Observation #1 is put into a cluster with 3 and 4.

1. Define the distance between this new cluster and the other clusters as the minimum distance between the three points in the cluster and the other clusters.

|  |  |  |
| --- | --- | --- |
|  | **1,3,4** | **2** |
| 1,3,4 |  | 2.56 |

Observation #2 is put in a cluster with 1, 3, and 4.

Another way to find all sets of distances is to use the daisy() function from the cluster package:

> library(cluster) #Location of daisy()

> daisy(x = set1, metric = "euclidean", stand = FALSE)

Dissimilarities :

1 2 3

2 2.561250

3 4.019950 2.828427

4 2.441311 3.000000 2.236068

Metric : euclidean

Number of objects : 4

The stand argument in daisy() allows you to automatically standardize your observations. Of course, you could use the scale() function with dist() to obtain the same results.

The nearest neighbor method can produce clusters of a variety of shapes. One problem that can occur with the method is it can produce elongated clusters which may or may not be desirable. Below is a plot of an elongated cluster, although it makes sense for these observations:



Elongated clusters can result due to “chaining”. Because only the nearest neighbors between clusters are examined, a chain of points may be grouped together.

Furthest neighbor (complete linkage)

The similarity between two clusters is the distance between the two observations furthest from each other in the clusters. Furthest neighbor tends to produce more compact clusters than nearest neighbor. Also, results from this clustering method can be distorted by the presence of outliers due to the “furthest neighbor” being used to judge similarity.

Please make sure you do not get confused with respect to how to decide which clusters get joined. The “closest” (most similar) two clusters are still joined as with nearest neighbor. The difference between furthest neighbor and nearest neighbor is simply how the similarity between clusters is defined.

The complete linkage name comes from it taking essentially all observations for each cluster to decide whether or not to link two clusters.

Example: Clustering for 4 points (DistanceExample.R)

The distances as given earlier:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** |
| 1 |  | 2.56 | 4.02 | 2.44 |
| 2 |  |  | 2.83 | 3.00 |
| 3 |  |  |  | 2.24 |

The clustering process:

1. 3 and 4 are joined to form (3,4)
2. Next, 1 and 2 are joined to form (1,2)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **1** | **2** | **3,4** |
| 1 |  | 2.56 | max(4.02, 2.44) = 4.02 |
| 2 |  |  | max(2.83, 3.00) = 3.00 |

1. Finally, (1,2) is joined with (3,4).

|  |  |  |
| --- | --- | --- |
|  | **1,2** | **3,4** |
| 1,2 |  | 4.02 |

Centroid

The similarity between two clusters is the distance between cluster means. This method can produce better results than other methods when there are outliers.

Example: Clustering for 4 points (DistanceExample.R)

The distances as given earlier:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** |
| 1 |  | 2.56 | 4.02 | 2.44 |
| 2 |  |  | 2.83 | 3.00 |
| 3 |  |  |  | 2.24 |

The clustering process:

1. 3 and 4 are joined to form (3,4)
2. Next, 1 and 2 are joined to form (1,2)

Note that the mean for the (3,4) cluster is [5, 2.5]′.

> set1[3:4,]

var1 var2

3 6 3

4 4 2

> colMeans(set1[3:4,])

var1 var2

5.0 2.5

This leads to the following new distances:

> set2 <- rbind(set1[1:2,], colMeans(set1[3:4,]))

> set2

var1 var2

1 2 3.4

2 4 5.0

3 5 2.5

> dist.temp <- dist(x = set2, method = "euclidean")

> dist.temp

1 2

2 2.561250

3 3.132092 2.692582

1. Finally, (1,2) is joined with (3,4)

Average (average linkage)

The similarity between two clusters is the average of all distances between each pair of observations across clusters. This method can tend to put together clusters that have small variances, along with produce clusters with the same variance.

Example: Clustering for 4 points (DistanceExample.R)

The distances as given earlier:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** |
| 1 |  | 2.56 | 4.02 | 2.44 |
| 2 |  |  | 2.83 | 3.00 |
| 3 |  |  |  | 2.24 |

1. 3 and 4 are joined to form (3,4)
2. Next, 1 and 2 are joined to form (1,2)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **1** | **2** | **3,4** |
| 1 |  | 2.56 | (4.02 + 2.44)/2 = 3.23 |
| 2 |  |  | (2.83 + 3.00)/2 = 2.915 |

1. Finally, (1,2) is joined with (3,4)

Ward’s minimum variance

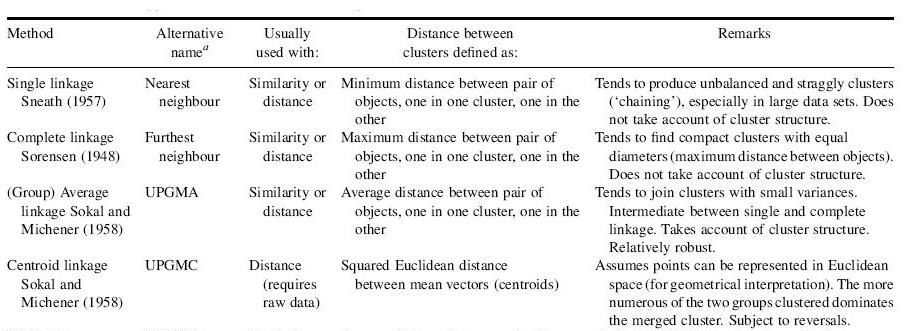
The similarity between two clusters is the square of the distance between cluster means divided by the sum of the reciprocals of the number of observations in each cluster:

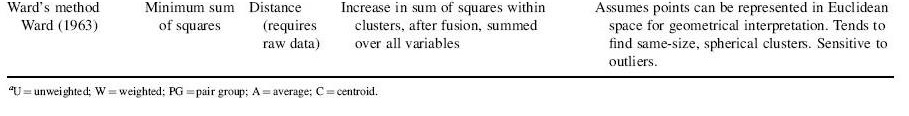


for clusters “a” and “b”. This method can tend to produce clusters with a similar number of observations. Also, results from this clustering method can be distorted by the presence of outliers.

Overall comparisons:

Everitt and Hothorn’s textbook provides the table below to compare the various methods:





* “Does not take account of cluster structure” refers to not using all observations when finding distances.
* “Reversals” refers to an upward branching hierarchical tree diagram due to distances getting smaller (we will discuss these diagrams shortly). Here’s an example:

<https://stats.stackexchange.com/questions/26769/cluster-analysis-in-r-produces-reversals-on-dendrogram>.

The SAS online help for PROC CLUSTER:

<https://support.sas.com/documentation/cdl/en/statug/63033/HTML/default/viewer.htm#statug_cluster_sect012.htm>

provides some discussions about the methods.

Johnson’s textbook says the following about the methods:

* Nearest neighbor tends to maximize the “connectedness” (chaining) of a pair of clusters and create fewer clusters than furthest neighbor
* Furthest neighbor will minimize the intracluster distances which produces compact clusters
* Other methods fall between the above two
* Try more than one method!

**Determining the number of clusters**

Determining the number of clusters can be difficult. When there are only two or three variables, one can easily visualize the clusters and make judgments from scatter plots. With more than three variables, this same type of process using other plotting methods is more difficult. Next, I will first give some examples of what “good” and “bad” clusters for two variables look like and then I will look at general tools available for any number of variables that one may have.

Example of a good and (maybe) bad cluster:



Avoid over joining of clusters. There probably should be three different clusters here. How can you determine that this happens if you can not look at it in two or three dimensions? Look at the similarities between the clusters!

Typically, most procedures will get to a point of the three clusters below. The distance between the two dotted line clusters will be given based upon one of the clustering methods (like nearest neighbor). This distance will be LARGE relative to previous distances, indicating the clusters likely should not be joined!



Here’s another example of a good and bad cluster:



Notice how the yellow cluster is taking away one observation that should be in the purple cluster. How can you determine that this happens if you can not look at it in two or three dimensions? Look at the similarities between the clusters!

Suppose the clustering procedure gets to a point of the four clusters below. The distance between the clusters will be given based upon one of the clustering methods (like furthest neighbor). This distance will be LARGE for the purple cluster to the single observation and for the yellow cluster to the single observation, indicating the clusters should not be joined! Notice that the end result may be four clusters even though the observation of interest is not in the purple cluster. This is what could happen with the furthest neighbor method!



Usually, we would like to avoid situations where a single observation is in a cluster unless the observation is an outlier.

Hierarchical tree diagram

Tree diagrams are helpful to see the distances between clusters along with the ordering for when clusters are formed. These diagrams are essential tools to help decide when to stop a clustering process!

Within the diagram, branches connect observation numbers to show the order that clusters are formed and the distances between clusters. The length of the branches are proportional to the distances.

Example: Clustering for 4 points (DistanceExample.R)

Below is the hierarchical tree diagram through using the nearest neighbor method:



Example: Two variable data set (TwoVariables.R)

A hierarchical tree diagram can be created using a few different function combinations in R.

First, I will use the hclust() function to obtain the cluster analysis results. The resulting object can be used with plot() function to create the hierarchical tree diagram. When using hclust(), I

1. Standardize the data first (again, standardized data is typically used with CA for reasons described previously)
2. Specify the matrix of standardized Euclidean distances within hclust() by using the d argument
3. Use the nearest neighbor (single linkage) method

> Z <- scale(set1)

> dist.mat <- dist(x = Z, method = "euclidean")

> clust.nn <- hclust(d = dist.mat, method = "single")

> clust.nn # Not useful

Call:

hclust(d = dist.mat, method = "single")

Cluster method : single

Distance : euclidean

Number of objects: 14

> names(clust.nn)

[1] "merge" "height" "order" "labels"

[5] "method" "call" "dist.method"

> clust.nn$merge #History of the clustering

[,1] [,2]

[1,] -13 -14

[2,] -1 -2

[3,] -3 2

[4,] -10 -12

[5,] -11 1

[6,] -6 -8

[7,] -7 6

[8,] -9 4

[9,] -4 -5

[10,] 5 8

[11,] 3 9

[12,] 7 11

[13,] 10 12

> clust.nn$height

[1] 0.1346317 0.1346317 0.2020020 0.2020020 0.2020020

[6] 0.2020020 0.2020020 0.2020020 0.3085156 0.3299124

[11] 0.5259419 0.6172453 1.7242287

Within clust.nn$merge, here’s what each row denotes:

1. Observations #13 and #14 were joined into one cluster
2. Observations #1 and #2 were joined into one cluster
3. Observation #3 was joined with cluster #2 to form a new cluster

13. Cluster #10 and #12 were joined to form a new

cluster

The difference in the display between a single observation number and a cluster number is given by whether or not a minus sign is in front of the number (minus is used for observation number). The clust.nn$height object shows the distances between the corresponding items in clust.nn$merge that are grouped together.

Below is a plot of the standardized data.

> Z <- scale(set1)

> par(pty = "s")

> common.limits <- c(min(Z) - 0.25, max(Z) + 0.25)

> plot(x = Z[,1], y = Z[,2], xlab = expression(z[1]), ylab

= expression(z[2]), panel.first = grid(), xlim =

common.limits, ylim = common.limits)

> text(x = Z[,1], y = Z[,2] + 0.1, labels = 1:nrow(Z))



Does the clustering history make sense given the above plot? Go through the process of drawing out the clusters step-by-step!

Below are additional code and output showing the class of the object and the corresponding method functions:

> class(clust.nn)

[1] "hclust"

> methods(class = hclust)

[1] as.dendrogram.hclust\* coef.hclust

[3] identify.hclust\* plot.hclust\*

[5] print.hclust\*

Non-visible functions are asterisked

Hierarchical tree diagram:

> plot(clust.nn)



The clustering order and the Euclidian distances are given on the hierarchical tree diagram (“cluster dendrogram”). For example, observations #13 and #14 were joined first with a standardized distance between them of 0.13. Also, observation #3 is joined with the (1,2) cluster with a distance of 0.2. A few sets of observations happen to have the same distances. For example, #9, #10, and #12 each have a distance of 0.2.

The hierarchical tree diagram allows you to see when clusters are being joined together and when they are actually far apart. To demonstrate this on a plot, a horizontal line is typically plotted to show where to “cut the tree”.

> plot(clust.nn)

> abline(h = 0.4, lty = "dashed", lwd = 2)

> abline(h = 1, lty = "dashed", lwd = 2)



At the very least, we see that there should be two clusters (notice there are 2 vertical lines intersecting my horizontal line drawn at 1.0). The standardized distance between joining the last two clusters is very large. There is also some justification for 4 clusters (notice there are 4 vertical lines intersecting my horizontal line drawn at 0.4). The standardized distance is relatively large when joining (1,2,3) with (4,5). Compare these conclusions with the plot of the standardized data!

The cutree() function can be used to obtain the cluster memberships with respect to a specific similarity or a specific number of clusters:

> clust2 <- cutree(tree = clust.nn, h = 1)

> clust2

[1] 1 1 1 1 1 1 1 1 2 2 2 2 2 2

> clust4 <- cutree(tree = clust.nn, k = 4)

> clust4

[1] 1 1 1 2 2 3 3 3 4 4 4 4 4 4

> data.frame(Z, clust2, clust4)

x1 x2 clust2 clust4

1 -1.1251360 -1.25855152 1 1

2 -0.9905044 -1.25855152 1 1

3 -0.8558727 -1.40914743 1 1

4 -0.7212411 -0.80676380 1 2

5 -0.4519777 -0.95735971 1 2

6 -0.8558727 -0.20438016 1 3

7 -0.8558727 0.09681166 1 3

8 -0.7212411 -0.05378425 1 3

9 0.8943389 0.54859938 2 4

10 1.0289706 0.69919529 2 4

11 1.0289706 1.15098302 2 4

12 1.1636022 0.84979120 2 4

13 1.1636022 1.30157893 2 4

14 1.2982339 1.30157893 2 4

> plot(x = Z[,1], y = Z[,2], xlab = expression(z[1]), ylab

= expression(z[2]), panel.first = grid(), xlim =

common.limits, ylim = common.limits, pch = clust4, col

= clust4)

> # Numbers correspond to colors, https://www.color-hex.com

> # 1 = black, 2 = redish, 3 = greenish, 4 = blueish,

> # 5 = cyan, 6 = magenta, 7 = yellowish, 8 = gray

> palette()

[1] "black" "#DF536B" "#61D04F" "#2297E6" "#28E2E5"

[6] "#CD0BBC" "#F5C710" "gray62"

A graph with numbers and points

Description automatically generated

The rect.hclust() function can be used with the hierarchical tree diagram to help draw rectangles around branches corresponding to a particular cluster:

> save.cl <- rect.hclust(clust.nn, k = 4, border = "red")

> save.cl

[[1]]

[1] 9 10 11 12 13 14

[[2]]

[1] 6 7 8

[[3]]

[1] 1 2 3

[[4]]

[1] 4 5



Of course, other agglomerative methods can be used as well. Below are some additional results:

> clust.fn <- hclust(d = dist.mat, method = "complete")

> data.frame(obs.cluster = clust.fn$merge, clust.fn$height)

obs.cluster.1 obs.cluster.2 clust.fn.height

1 -13 -14 0.1346317

2 -1 -2 0.1346317

3 -10 -12 0.2020020

4 -6 -8 0.2020020

5 -7 4 0.3011918

6 -4 -5 0.3085156

7 -11 1 0.3085156

8 -3 2 0.3085156

9 -9 3 0.4040040

10 6 8 0.7374677

11 7 9 0.8544644

12 5 10 1.5059591

13 11 12 3.5251936

> plot(clust.fn)

> abline(h = 2, lty = "dashed", lwd = 2)

> abline(h = 1.25, lty = "dashed", lwd = 2)



> clust.ave <- hclust(d = dist.mat, method = "average")

> data.frame(obs.cluster = clust.ave$merge, clust.ave$height)

obs.cluster.1 obs.cluster.2 clust.ave.height

1 -13 -14 0.1346317

2 -1 -2 0.1346317

3 -10 -12 0.2020020

4 -6 -8 0.2020020

5 -7 4 0.2515969

6 -11 1 0.2552588

7 -3 2 0.2552588

8 -9 3 0.3030030

9 -4 -5 0.3085156

10 6 8 0.5837071

11 7 9 0.6182830

12 5 11 1.1121687

13 10 12 2.6089888

> plot(clust.ward)

> abline(h = 7, lty = "dashed", lwd = 2)

> abline(h = 2, lty = "dashed", lwd = 2)

A diagram of a diagram

Description automatically generated

All of the agglomerative methods (except nearest neighbor) suggest that three clusters may be sufficient. With respect to furthest neighbor, examine the plot of the standardized data to see why.

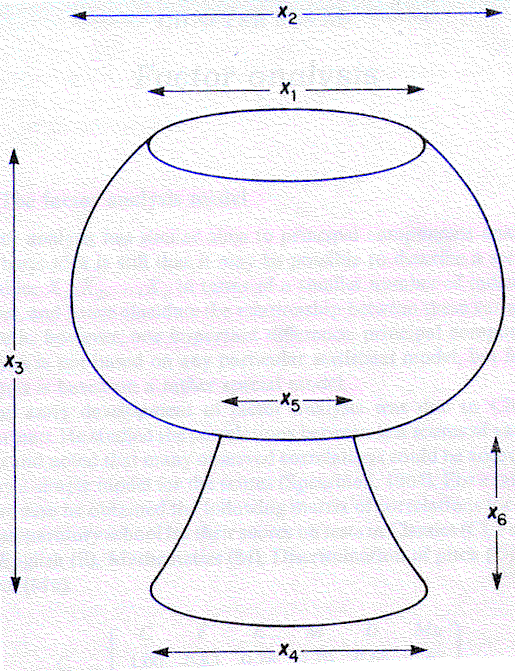
Note: Due to a computational quirk with hclust(), the distances inputted for the d argument need to be squared whenever the centroid method is used!

A second commonly used function for CA is agnes() from the cluster package. My program shows how to use this function and obtain a hierarchical tree diagram. Below are some notes about the code:

* A data frame, rather than a distance matrix, can be given directly to agnes().
* The stand = TRUE argument value in agnes() does not standardize the data as we would normally do. Rather, it mean adjusts the data and then divides by the mean absolute deviation.
* A banner plot can be produced with plot(). This provides an alternative way to view the distances shown in a hierarchical tree diagram.

Example: Goblet data (GobletCA.R, goblet.csv)

Below is our diagram from before:



We will examine the nearest neighbor method here. Applying other agglomerative methods is part of the practice problems.

There is justification for using the Euclidean distance rather than the standardized Euclidean distance. I will use the standardized first:

> goblet <- read.csv("C:\\chris\\goblet.csv")

> head(goblet)

goblet x1 x2 x3 x4 x5 x6

1 1 13 21 23 14 7 8

2 2 14 14 24 19 5 9

3 3 19 23 24 20 6 12

4 4 17 18 16 16 11 8

5 5 19 20 16 16 10 7

6 6 12 20 24 17 6 9

> goblet2 <- data.frame(ID = goblet$goblet,

w1 = goblet$x1/goblet$x3,

w2 = goblet$x2/goblet$x3,

w4 = goblet$x4/goblet$x3,

w5 = goblet$x5/goblet$x3,

w6 = goblet$x6/goblet$x3)

> Z <- scale(goblet2[,-1])

> dist.mat <- dist(x = Z, method = "euclidean")

> clust.nn <- hclust(d = dist.mat, method = "single")

> data.frame(obs.cluster = clust.nn$merge, clust.nn$height)

obs.cluster.1 obs.cluster.2 clust.nn.height

1 -3 -16 0.4018334

2 -6 -14 0.4547302

3 -1 -15 0.5640762

4 -19 2 0.5945182

5 -12 3 0.7048235

6 -21 4 0.7274185

7 5 6 0.7514834

8 -13 7 0.7607212

9 -7 -11 0.7780301

10 -9 8 0.7929662

11 -8 10 0.8639470

12 -25 9 0.8670719

13 -4 -17 0.8690132

14 -20 12 0.9578009

15 11 14 0.9972367

16 -10 -22 1.1025049

17 15 16 1.1740236

18 -5 13 1.2855017

19 1 17 1.3709665

20 -2 19 1.5829651

21 -23 -24 1.6915480

22 -18 20 1.7023425

23 18 22 1.9059615

24 21 23 2.0378799

> plot(clust.nn)

> abline(h = 1.75, lty = "dashed", lwd = 2)

> abline(h = 1.45, lty = "dashed", lwd = 2)

> abline(h = 1.25, lty = "dashed", lwd = 2)



Based on the plot, I think either 3 or 6 and maybe 8 clusters are possible. Note that you need to be careful when counting the number of times the horizontal line intersects with one of the vertical branches here. For example, observations #23, #24, #18, and #2 are separate clusters when there are 6 total clusters. Below is what the 6 clusters look like when the rect.hclust() function is used to highlight them:

> plot(clust.nn)

> save.cl <- rect.hclust(clust.nn, k = 6, border = "red")

> save.cl

[[1]]

[1] 23

[[2]]

[1] 24

[[3]]

[1] 4 5 17

[[4]]

[1] 18

[[5]]

[1] 2

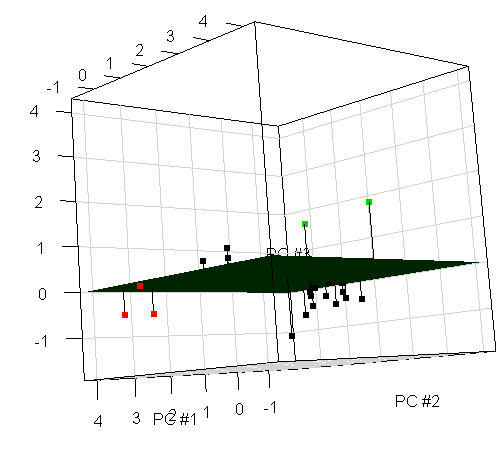
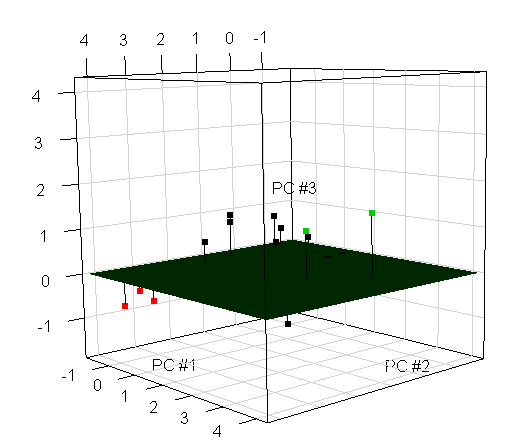
[[6]]

[1] 1 3 6 7 8 9 10 11 12 13 14 15 16 19 20 21 22 25



I wrote my own function named PCA.CA.plot() that constructs plots of the PC scores, where the plotting point corresponds to a particular cluster. Please see the code in the corresponding program. Below are the results from running the function with specifying three clusters:



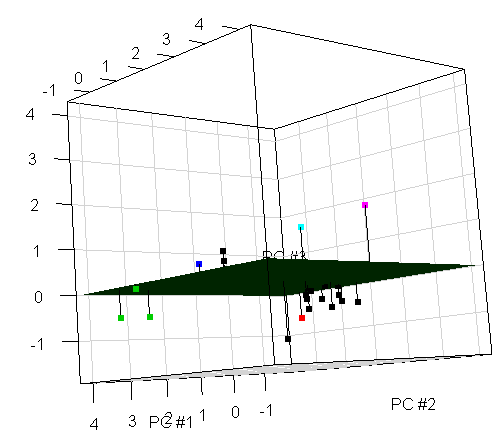
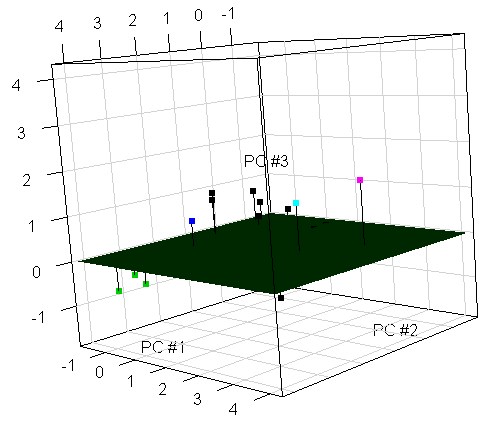


Remember that PCA does NOT show ALL of the information in the data set. However, it can still be one way to visualize the clusters, determine an appropriate number of clusters, and make sense of them. Also, because 2 or 3 principal components seem to be appropriate for this data set, this leads strength to using PCA to help justify the clusters.

What do you think of three clusters?

Below are the results from PCA.CA.plot() with six clusters:





What do you think of six clusters?

Notice that some observations are in clusters by themselves. At least for this situation, I do not think this is desirable.

Comments:

* Examine these same types of PC plots for a number of clusters NOT suggested by a hierarchical tree diagram. Do conclusions reached by examining these PC plots agree with those conclusions reached by examining the hierarchical tree diagram?
* What if 2 or 3 PCs do not explain as much of the total variation as desired in other data settings? You can still examine PC plots, but the conclusions about the CA will not be as strong. Also, you can use other plotting techniques discussed earlier in the course. In particular, a parallel coordinates plot (with the standardized variables or with > 3 PCs), where lines are colored by cluster may work well. For example, below is the parallel coordinate plot with the standardized variables and three clusters (code is in program):

A graph of lines and colors

Description automatically generated

What do you think about the clusters?

Clusters 2 and 3 are somewhat similar for w1 and w2 (large cups; notice they are also mainly higher than cluster 1). Clusters 2 and 3 differences are shown in w4, w5, and w6 (base).

Examine the same plot for six clusters on your own.

* After clusters are found, one can try to interpret what the clusters represent in the context of the problem:
  + Cluster #1 (black) has smaller PC #1 values indicating it represents the smaller goblets relative to the overall height.
  + Cluster #2 (red) has large PC #1 values indicating that these are large goblets relative to the overall height.
  + Cluster #3 (green3) has large PC #2 values indicating its top is larger than most when contrasted with its base size (relative to the overall height).
  + From the parallel coordinates plot, we also see that clusters #2 and #3 are similar cup sizes (w1 and w2), but their differences are with respect to the size of their base (w4, w5, and w6)

Next, here are some of the highlights from using Euclidean distance rather than the standardized Euclidean distance:





Comments:

* From the hierarchical tree diagram, 2, 3 (even if it is a relatively shorter distance from 4 to 3 clusters than perhaps for other cluster merges), 8, and 11 (maybe) clusters are at least somewhat justifiable. Overall, I think 2 or 3 are best from this plot.
* The bubble plot for the first three PCs with 2 clusters shows that using only 2 clusters may be questionable. In particular, notice how 4, 5, 17 and even 18 are away from most other points. Notice that in the hierarchical tree diagram we saw that joining these observations with the large cluster of points was perhaps questionable.
* Based on what we see in the bubble plot, perhaps four clusters would be o.k.. These clusters are the same as with three clusters, but observations 3, 10, 16, and 22 may be in their own cluster. Perhaps other agglomerative methods will find this or other clusters.
* Overall, I think three clusters are best for the non-standardized data and the nearest neighbor method.

Other methods to determine the number of clusters

Johnson’s textbook and other SAS-based sources have advocated in the past to use numerical measures based on classical multivariate methods to determine where to stop the clustering process. These methods include:

* A pseudo Hotelling’s T2 test statistic

Hotelling’s T2 test statistic can be used to test H0:**μ**1=**μ**2 vs. Ha:**μ**1≠**μ**2 for two multivariate normal populations. The corresponding test statistic will be discussed later in the course. The key point to understand is that a larger statistic generally indicates more evidence against the null hypothesis.

This same test could be used to help determine if two different clusters should be combined. For example, if the 1 and 2 subscripts on **μ** denote the clusters, the hypothesis test determines if the cluster means are not equal. If they are not equal (reject H0), then the clusters should not be combined. If there is not evidence to prove different means (don’t reject H0), then the clusters may be combined.

The reason why this is called “pseudo” is because a formal test should not be done. Because the independence of the random vectors is violated (observations are not put into groups at random), the statistic is not distributed as a random variable from an F distribution as it would be in a normal situation. Instead of using an actual critical value then, one can look for “large” values and large relative increases in value to help determine when there is a difference between cluster means. When these criteria are met, this indicates the clusters should not be combined.

* A pseudo F statistic

This statistic measures the separation among all the clusters at the current number of clusters. This is a statistic that could be used to test H0:**μ**1 = **μ**2 = = **μ**K vs. Ha:not all equal where K is the number of clusters at that time. The corresponding statistic can be used in a similar manner as the pseudo Hotelling’s T2 test statistic with the same caveats.

**Nonhierarchical clustering**

We will focus on a popular nonhierarchical method known as K-means clustering, where K is the number of clusters of interest.

Possible problem with this method: K needs to be specified in advance before starting to implement the method! How could you choose K?

Possible advantage with this method: Unlike in hierarchical clustering, all pairwise distances between observations do not need to be found. This means the procedure can more easily be applied to LARGE data sets.

K-means clustering process

1. Specify K initial cluster centers. These centers are often called “seed points” since they help grow the clusters. In addition, these seed points are often K observations chosen at random from the data set.

   
K = 3 in diagram

1. For each observation, assign it to a cluster with the nearest seed. Euclidean distances of standardized variables are typically used here.



1. Replace the seed points with the corresponding cluster means (these are often still called seeds).



1. Reassign observations to clusters with the nearest cluster mean (seed).



1. Continue this process of finding the new cluster means and reassigning the observations to clusters.

Notes:

* This is an iterative process! The process can continue until the cluster means (seeds) do not change too much.
* Equivalently, the algorithm can be presented as assigning an observation to the cluster which minimizes the within cluster sum of squares. In other words, minimize

,

where xrik denotes observation r for variable i in cluster k and  is the mean of variable i in cluster k, at each step of the algorithm. Of course,  at step #1 are not the cluster means, but rather the initial cluster seed points. Also, note that the algorithm is not guaranteed to find the cluster membership that minimizes the within cluster sum of squares. Rather, a “local” minimum could be found instead.

* Notice the procedure is dependent on what observations are chosen as the initial cluster seeds. Therefore, it is good to re-run the analysis a few times with different cluster seeds to make sure the results do not change significantly. Without re-running the analysis, the concern is that a local minimum could have been found which is not as good as what could be achieved.
* There are a number of small changes to the algorithm presented here that can make it better (e.g., faster, finding “true” clusters). For example, each possible re-assignment given in step #4 could be done one-at-a-time so that the cluster means can be updated. The kmeans() function of R provides four different algorithms that involve small changes like this. We will use the default algorithm given by the function.
* Other definitions for a center of a cluster or distance lead to other clustering methods. For example, there is a k-medians clustering method.

Example: Clustering for 4 points (DistanceExample.R)

Below are 4 observations

|  |  |  |
| --- | --- | --- |
| **Observation** | **x1** | **x2** |
| 1 | 2 | 3.4 |
| 2 | 4 | 5 |
| 3 | 6 | 3 |
| 4 | 4 | 2 |

Below is the same plot of the data as we saw earlier.



Remember that all pairwise distances are NOT needed for K-means clustering despite them being show in the above plot.

Below is the code used to implement K-means clustering in R:

> set.seed(7812)

> clust.2means <- kmeans(x = set1, centers = 2)

> clust.2means

K-means clustering with 2 clusters of sizes 2, 2

Cluster means:

var1 var2

1 5 2.5

2 3 4.2

Clustering vector:

[1] 2 2 1 1

Within cluster sum of squares by cluster:

[1] 2.50 3.28

(between\_SS / total\_SS = 54.4 %)

Available components:

[1] "cluster" "centers" "totss" "withinss"

[5] "tot.withinss" "betweenss" "size"

> names(clust.2means) # Duplicates info from clust.2means

[1] "cluster" "centers" "totss" "withinss"

[5] "tot.withinss" "betweenss" "size"

> clust.2means$cluster

[1] 2 2 1 1

> set1.clust <- data.frame(set1, cluster =

clust.2means$cluster)

> set1.clust

var1 var2 cluster

1 2 3.4 2

2 4 5.0 2

3 6 3.0 1

4 4 2.0 1

> class(clust.2means)

[1] "kmeans"

> methods(class = kmeans) #fitted() is not useful

[1] fitted.kmeans\* print.kmeans\*

Comments:

* The initial cluster “means” are selected as observations from the data set. This is why I set a seed number prior to using kmeans() so that I could duplicate my exact results at a later time.
* The final cluster means are at [5, 2.5]′ for cluster #2 containing observations #1 and #2 and [3, 4.2]′ for cluster #1 containing observations #3 and #4.

> clust.2means$center

var1 var2

1 5 2.5

2 3 4.2

> points(x = clust.2means$center[,1], y =

clust.2means$center[,2], pch = 2, col = "darkgreen", lwd = 5) #Means on plot shown as triangles



* The within cluster sums of squares is 2.5 for cluster #1 and 3.28 for cluster #2. Below are the calculations of this quantity without using kmeans():

> save.var <- aggregate(formula = cbind(var1, var2) ~

cluster, data = set1.clust, FUN = var)

> sum(save.var[1, 2:3]\*(2-1))

[1] 2.5

> sum(save.var[2, 2:3]\*(2-1))

[1] 3.28

> clust.2means$tot.withinss

[1] 5.78

> sum(clust.2means$withinss)

[1] 5.78

* The “between\_SS / total\_SS = 54.4 %” part of the output gives a measure of how much total variation in the data is being explained by the clusters.

> clust.2means$totss

[1] 12.67

> sum((set1[,1] - mean(set1[,1]))^2 + (set1[,2] –

mean(set1[,2]))^2)

[1] 12.67

> sum(scale(set1, scale = FALSE)^2) #From kmeans()

function

[1] 12.67

> clust.2means$betweenss

[1] 6.89

> clust.2means$totss - clust.2means$tot.withinss

[1] 6.89

> clust.2means$betweenss/clust.2means$totss

[1] 0.5438043

This measure can be helpful to determine the number of clusters (more on this shortly). Do not think of this percentage as like an R2 from regression analysis!

* When I change my seed number to 7813, I obtain a different set of clusters:

> set.seed(7813)

> kmeans(x = set1, centers = 2)

K-means clustering with 2 clusters of sizes 2, 2

Cluster means:

var1 var2

1 3 2.7

2 5 4.0

Clustering vector:

[1] 1 2 2 1

Within cluster sum of squares by cluster:

[1] 2.98 4.00

(between\_SS / total\_SS = 44.9 %)

Available components:

[1] "cluster" "centers" "totss" "withinss"

[5] "tot.withinss" "betweenss" "size"

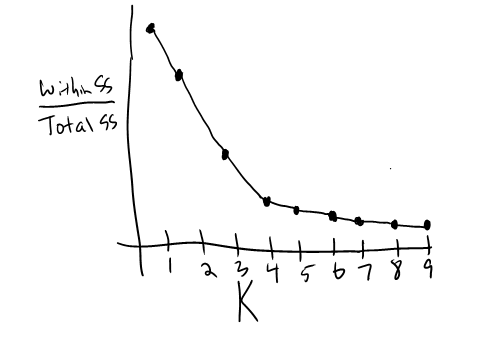
Determining the number of clusters

As with hierarchical clustering, there may be more than one justifiable answer. PCA can be used to provide an initial idea of the number of clusters to use.

Once performing the clustering, we can calculate



where  is the sample mean of variable i in cluster k and  is the sample mean for variable i. The value for W can be examined across a number of different K values. A plot of these values versus K can be constructed like a scree plot. Once W values tend to roughly level off, this suggests that no larger values of K are needed (not much is gained by a larger K). For example, the plot below suggests using 4 clusters.



Equivalently, a plot of just the within cluster sums of squares on the y-axis could be plotted. This is because the total sums of squares will not change for different K.

Everitt and Hothorn’s textbook suggest using hierarchical clustering first to obtain an initial idea for the number of clusters. I am not in complete agreement with their suggestion. If one performs hierarchical clustering, then why perform k-means clustering? Maybe this is o.k. if one is just trying a variety of exploratory methods.

Of course, scatter plots of the PCs, parallel coordinates plots, …, should be used to make sure the number of clusters chosen make sense.

Different intitial cluster seeds

Different implementations of K-means clustering may result in different clusters due to the choice of initial custer seeds. We should re-run the clustering algorithm a few times to make sure that significantly better clusters would not be found with different initial cluster seeds.

The re-running can be accomplished in one of two ways:

1. Use a few different seed numbers in set.seed() prior to running kmeans(). Compare the clusters found in the new implementations to what you obtained originally. Also, compare the value of W in the new implementations to the original. If there are significant changes that result in a “better” set of clusters (e.g., an increase in W), this is a sign that the K-means clustering could be better than originally obtained.
2. Use the nstart argument of kmeans() to automatically run additional analyses. For example, nstart = 10 will essentially implement the K-means process 10 different times with 10 randomly selected intial cluster seeds. The kmeans() function will only return the results from the implementation with the smallest W.

One problem with using 1) is that it can be difficult to compare cluster results from one run to the next. The reason is that the cluster numbers given by the $cluster component of the list from kmeans() may not use the same numbering scheme. For example, a cluster with the same members in two runs of kmeans() could be numbered as cluster #1 for one run and then cluster #5 for another run due to the random process of selecting initial cluster seeds.

A small downside of 2) is that you do not see these other implementations which you would with the first option above. This probably is not a problem unless for some reason you really wanted to see the variability from one to another.

Example: Goblet data (GobletCA.R, goblet.csv)

Using the standardized data, I ran kmeans() with K = 1, …, 10 and nstart = 10 to obtain the within cluster sums of squares. A plot is then constructed to examine W over these different values of K.

> K.levels <- 1:10

> save.wss <- numeric(length(K.levels))

> set.seed(1221)

> for(K in K.levels) {

clust.means <- kmeans(x = Z, centers = K)

save.wss[K] <- sum(clust.means$withinss)

}

> tss <- clust.means$totss

> data.frame(K, save.wss/tss)

K save.wss.tss

1 10 1.00000000

2 10 0.54608020

3 10 0.35756472

4 10 0.24342909

5 10 0.18282526

6 10 0.13175090

7 10 0.10836545

8 10 0.09040402

9 10 0.07925411

10 10 0.06484325

> plot(x = K.levels, y = save.wss/tss, ylab = "Within SS /

Total SS", xlab = "K", type = "o", panel.first =

grid())

A graph with a line

Description automatically generated

The plot suggests maybe five clusters, but other choices are possible. First, let’s examine five clusters more closely.

> set.seed(8912)

> clust.5means <- kmeans(x = Z, centers = 5, nstart = 10)

> clust.5means

K-means clustering with 5 clusters of sizes 3, 9, 2, 8, 3

Cluster means:

w1 w2 w4 w5 w6

1 1.5478715 1.5631580 2.0833772 2.20516957 0.9376561

2 -0.7997718 -0.8968361 -0.2751575 -0.69705152 0.3336271

3 1.7564959 1.4777022 -0.8731928 -0.44852575 -1.4158609

4 -0.2836492 -0.1444758 -0.5486403 0.10617017 -0.8654335

5 0.4368443 0.5274844 0.7872647 -0.09811831 1.3131926

Clustering vector:

[1] 4 2 5 1 1 2 2 4 4 4 2 4 4 2 4 5 1 5 2 2 2 4 3 3 2

Within cluster sum of squares by cluster:

[1] 1.514714 8.836347 1.430667 7.911141 2.246162

(between\_SS / total\_SS = 81.7 %)

Available components:

[1] "cluster" "centers" "totss" "withinss"

[5] "tot.withinss" "betweenss" "size"

I wrote my own function named PCA.CA.plot2() that constructs plots of the PC scores, where the plotting point corresponds to a particular cluster. Please see the code in the corresponding program. Below are the results from running the function with five clusters:

A graph with numbers and symbols

Description automatically generated A graph with red and blue circles

Description automatically generated

A diagram of a green square

Description automatically generatedA graph of a graph with numbers and lines

Description automatically generated with medium confidence

Comments:

* We do see some separation among the clusters when taking into account three PCs; however, I am a little concered that there are too many clusters from these plots.
* How could you plot the cluster means?
* Below is a parallel coordinates plot with the same colors as used in the PC plots:

> parcoord(x = goblet2, col = clust.5means$cluster,

main = "Goblet parallel coordinates plot",

lwd = clust.5means$cluster)

A graph of lines and colors

Description automatically generated with medium confidence

Overall, this plot perhaps more sense than the PC plots. Below is a discussion of the clusters:

* + Black: These all have large values for all variables
  + Green: Large values for w1 and w2, but smaller values for w4, w5, and w6
  + Red: Very small w1 and w5 values but middle values for w4 and mostly large values for w6
  + Lighter blue: Similar to red but with larger w1, w2, and w4 values
  + Dark blue: Similar to RED but can have larger w1 and w5 values

Overall, the red, lighter blue, and darker blue have less distinguishing features, which may suggest a fewer number of clusters may be preferred. Also, examine the observations with these same colors in the PC plots.

Examine some different numbers of clusters (both smaller and larger than 5) on your own. Below are the corresponding plots for three clusters:

A graph with numbers and lines

Description automatically generated A graph with red and blue circles with Ice hockey rink in the background

Description automatically generated

A diagram of a green square with red dots and numbers

Description automatically generatedA diagram of a graph

Description automatically generated

A green and red lines

Description automatically generated

What do you think about three clusters? Note that with four clusters, observations in the lower left corner of the scatter plot (2, 3, 7, 11, 16, 20, and 25), form their own cluster. This split with the previous overall cluster roughly corresponds to a split between positive and negative PC #3 values.

Consider these results for 3 and 4 clusters relative to what the W vs. K plot suggested.