Introduction to Unsupervised Learning

36-290 – Statistical Research Methodology

Week 3 Thursday – Fall 2021

The Setting

The setting for *unsupervised learning* is that you have a collection of p measurements for each of n objects $X_1, \ldots, X_n$, where for a given object i,

$$X_{i1}, X_{i2}, \ldots, X_{ip} \sim P$$

where P is some p-dimensional distribution that we might not know much about *a priori*.

The term "unsupervised" simply means that none of the variables are response variables, i.e., there are no labeled data.

One can think of unsupervised learning as being an extension of EDA, where the goal is to discover interesting things about the data. The main, overriding issue with unsupervised learning is that *there are no universally accepted mechanisms for model assessment or selection, i.e., there is no unique right answer!*

Clustering

What is clustering?

- It is the partitioning of data into homogeneous subgroups.

What is the goal of clustering?

- To define clusters for which the within-cluster variation is relatively small.

...but what defines small?

- Generally, we want small Euclidean distances between the points within a cluster, where the Euclidean distance between datum i and datum j is

$$d_{ij} = \sqrt{(X_{i1} - X_{j1})^2 + \cdots + (X_{ip} - X_{jp})^2}$$

  Note that *units matter*! One axis might dominate the others when computing Euclidean distance because the range of values along that axis is much larger than the ranges along other axes. So it is a good idea to standardize each column of the input data frame so as to have mean zero and standard deviation one. See `scale()`.

K-Means Clustering

The algorithm for K-means clustering is straightforward:

Algorithm 10.1 *K-Means Clustering*

1. Randomly assign a number, from 1 to K, to each of the observations. These serve as initial cluster assignments for the observations.
2. Iterate until the cluster assignments stop changing:
   (a) For each of the K clusters, compute the cluster *centroid*. The kth cluster centroid is the vector of the p feature means for the observations in the kth cluster.
   (b) Assign each observation to the cluster whose centroid is closest (where *closest* is defined using Euclidean distance).

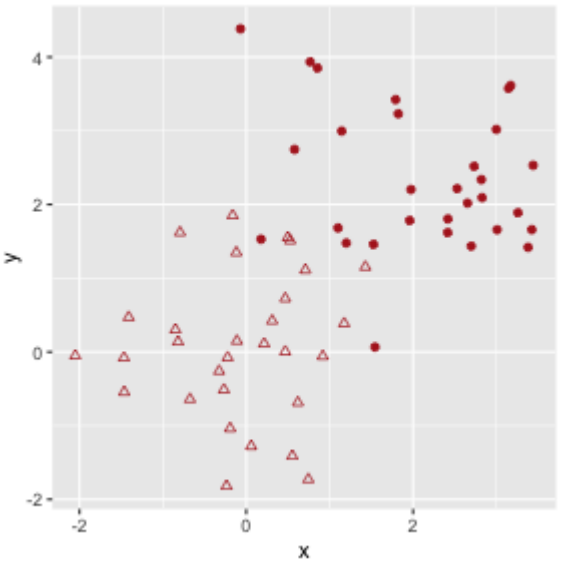
Note the following:

- As stated above, there is no universally accepted metric that leads one to conclude that a particular value of K is the optimal one.
- Your results will change from run to run unless you explicitly set the random number seed immediately before calling `kmeans()`.
- Step 1 of the algorithm indicates that the algorithm randomly associates data to clusters. To mitigate this aspect of randomness, set the `nstart` argument in the function call to a large number (e.g., 50), and select the best result, with best result meaning the one with the smallest value of within-cluster variation.

K-Means Clustering: Example

Let's generate some fake data:

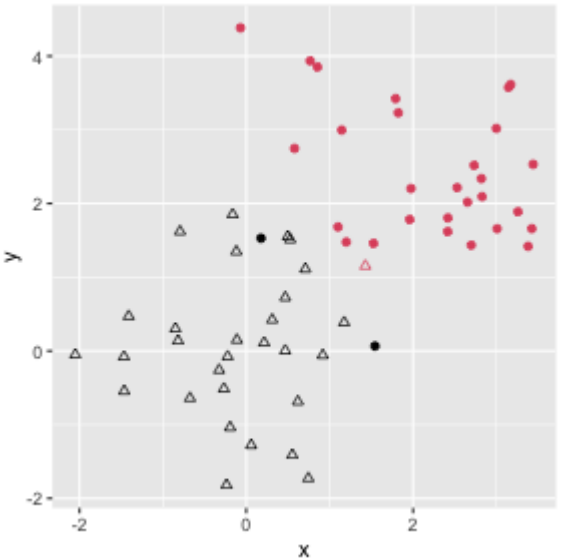
```
set.seed(101)
x = c(rnorm(30),rnorm(30,mean=2.25))
y = c(rnorm(30),rnorm(30,mean=2.25))
s = c(rep(2,30),rep(19,30))
library(ggplot2)
ggplot(data=data.frame(x,y),mapping=aes(x=x,y=y)) +
  geom_point(color="firebrick",shape=s)
```



K-Means Clustering: Example

What happens if we assume two clusters?

```
km.out = kmeans(data.frame(x,y),2,nstart=20)
color = km.out$cluster
ggplot(data=data.frame(x,y),mapping=aes(x=x,y=y)) +
  geom_point(color=color,shape=s)
```



Note that here I do not utilize `scale()` (as in, e.g., `data=scale(data.frame(x,y))`) because the variances along the `x` and `y` directions are the same.

K-Means Clustering: Output

```
km.out
```

```
## K-means clustering with 2 clusters of sizes 31, 29
##
## Cluster means:
##             x         y
## 1 -0.07075038 0.1007261
## 2  2.17206608 2.4048297
##
## Clustering vector:
##  [1] 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 2 2 2
## [38] 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2
##
## Within cluster sum of squares by cluster:
## [1] 50.24476 49.41480
##  (between_SS / total_SS =  60.9 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

K-Means Clustering: Output

Look at the available components above:

totss: compute the distances between all pairs of data, square the results, sum the squared quantities, and divide by the number of data points:

```
km.out$totss
```

```
## [1] 254.574
```

```
sum(dist(data.frame(x,y))^2)/length(x) # gets the same result!
```

```
## [1] 254.574
```

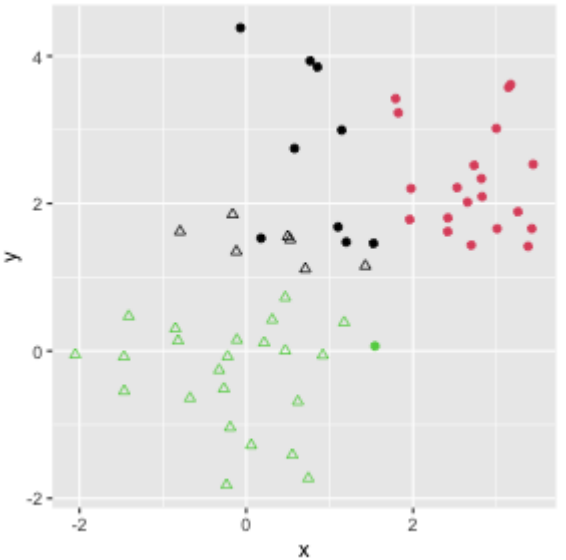
tot.withinss is the same, except we carry out the computation only within *each defined cluster*, and then sum the k results. We want this to be small relative to `totss`.

betweenss is `totss` - `tot.withinss`. We want this to be large relative to `totss`.

K-Means Clustering: Example

What happens if we assume three clusters?

```
km.out = kmeans(data.frame(x,y),3,nstart=20)
color = km.out$cluster
ggplot(data=data.frame(x,y),mapping=aes(x=x,y=y)) +
  geom_point(color=color,shape=s)
```



K-Means Clustering: Example

```
km.out
```

```
## K-means clustering with 3 clusters of sizes 16, 20, 24
##
## Cluster means:
##            x         y
## 1  0.5848076  2.138317
## 2  2.7232537  2.302554
## 3 -0.1260559 -0.308399
##
## Clustering vector:
##  [1] 3 3 3 3 3 3 3 3 3 1 1 1 3 3 3 3 3 3 1 1 3 3 3 3 1 3 1 2 2 2 1 2 1 2
## [38] 2 1 2 2 2 1 2 1 2 2 3 2 1 1 2 2 1 2 1 2 2 2 2
##
## Within cluster sum of squares by cluster:
## [1] 23.76652 15.06153 30.18075
##  (between_SS / total_SS =  72.9 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

As $k \to n$, `(between_SS/total_SS)` goes to 100%. 100% is not necessarily your goal: you'd be "overfitting" the data at that point. (Each point is its own "cluster.")

Hierarchical Clustering

The algorithm for hierarchical clustering is also straightforward:

Algorithm 10.2 *Hierarchical Clustering*

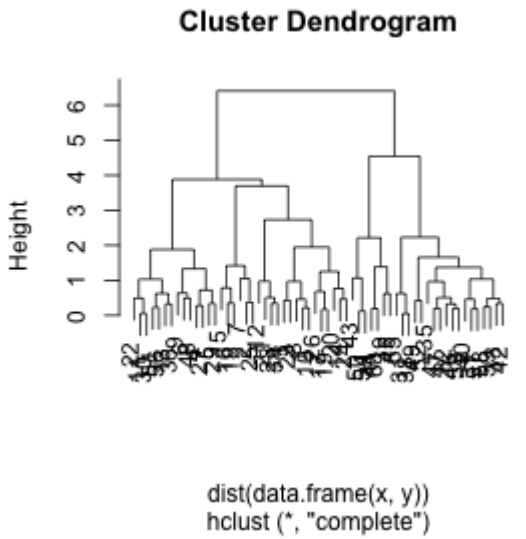
1. Begin with n observations and a measure (such as Euclidean distance) of all the $\binom{n}{2} = n(n-1)/2$ pairwise dissimilarities. Treat each observation as its own cluster.
2. For $i = n, n-1, \ldots, 2$:
   (a) Examine all pairwise inter-cluster dissimilarities among the i clusters and identify the pair of clusters that are least dissimilar (that is, most similar). Fuse these two clusters. The dissimilarity between these two clusters indicates the height in the dendrogram at which the fusion should be placed.
   (b) Compute the new pairwise inter-cluster dissimilarities among the $i-1$ remaining clusters.

Note the "[t]reat each observation as its own cluster." This is specifically "bottom-up" or *agglomerative* clustering. A primary limitation of agglomerative clustering is that all clusters lie within other clusters (hence the name "hierarchical").

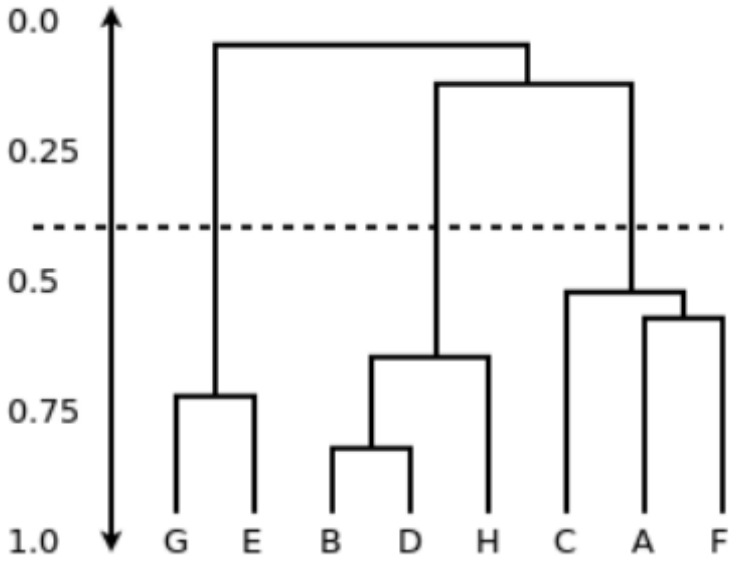
Hierarchical Clustering: Example

Here's an example of a dendrogram for complete-linkage hierarchical clustering. The "height" along the vertical axis at which two clusters fuse indicates dissimilarity...the higher the merge, the greater the dissimilarity between clusters. One extracts cluster members using, e.g., the `cutree()` function and processing its output.

```
hc.out = hclust(dist(data.frame(x,y)),method="complete")
plot(hc.out)
```



Hierarchical Clustering: Tree Cutting



(Credit goes to Erik Velldal for the above image.)

Hierarchical Clustering: Linkage

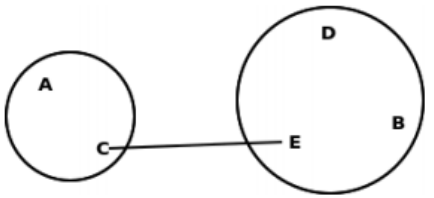
In agglomerative clustering, clusters are built up piece by piece by linking them together. There is no unique algorithm for how that linking is done! Note the descriptions below, and note that average and complete linkage are the algorithms most commonly used.

Linkage	*Description*
Complete	Maximal intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the *largest* of these dissimilarities.
Single	Minimal intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the *smallest* of these dissimilarities. Single linkage can result in extended, trailing clusters in which single observations are fused one-at-a-time.
Average	Mean intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the *average* of these dissimilarities.
Centroid	Dissimilarity between the centroid for cluster A (a mean vector of length p) and the centroid for cluster B. Centroid linkage can result in undesirable *inversions*.

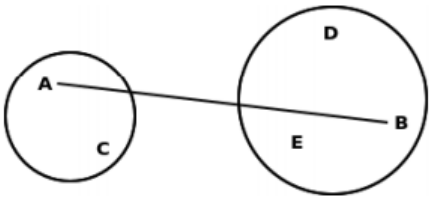
The concept of "inversion" is shown on the next page. Note that the first three listed methods are "monotonic," meaning they cannot generate trees with inversions.

Hierarchical Clustering: Linkage

Single linkage (also called the "friend-of-friends" algorithm):

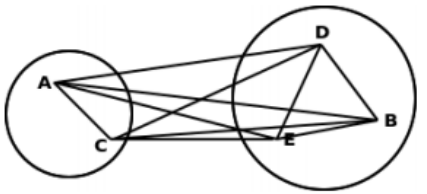


Complete linkage:

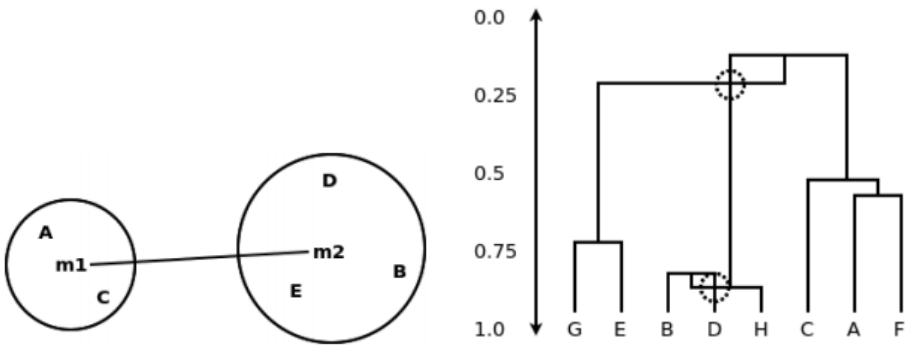


Hierarchical Clustering: Linkage

Average linkage:



Centroid linkage (plus an example of "inversion"):



(Credit goes to Erik Velldal for the images on this page.)

K-Means and Hierarchical Clustering: Wrap-Up

So: which of these should you use? Quoting ISLR: "we recommend performing clustering with different choices of [methods and parameters], and looking at the full set of results in order to see what patterns consistently emerge."

That said:

- In K-means, you specify the number of clusters before running the algorithm, as opposed to hierarchical clustering, where you specify the number of clusters afterwards by cutting across a dendrogram.
- Note that dendrograms can be really hard to read when the sample size is large.
- Also note that all data are assigned to clusters in these algorithms. Maybe that shouldn't be the case. Another algorithm we do not cover is the *Gaussian Mixture Model* (GMM), which can be used to specify probabilities that an observation belongs to a particular group/cluster. Then you can decide how to assign observations. (See, e.g., the `GMM` function in the `ClusterR` package.)