Instructors: Parth Shah, Riju Pahwa

Hierarchical Clustering

Big Ideas

Clustering is an unsupervised algorithm that groups data by similarity. Unsupervised simply means after a given step or output from the algorithm you do not feed back an error for it to use to correct its behavior. Of course in the case of clustering this is obvious. If you simply want to group data its hard to pass back an error. This is another way you can think about clustering as an unsupervised algorithm. It will just do what it does with 0 influence from you.

Why is clustering useful? First off it lets you extract relationships you might not have even seen. It also lets you see what data points are similar and different which may help to represent the data in a lower dimension for a classification algorithm. In general, its a great algorithm for recognizing complex relationships in high dimensional data.

To first understand the value of hierarchical clustering it is important to understand the downsides of approaches similar to K-means.

These approaches tend to have the following issues:

1. Convergence time is poor. For example K-means takes worst case exponential number ($2^{\Omega(n)}$ for those familiar with O-notation) of iterations.
2. The final clusters depend heavily on the initialization. Usually a k random points are chosen.
3. The number of clusters is a huge issue. You are forced to specify the number of clusters in the beginning. Note: In K means you are also not guaranteed K clusters. See lecture 2 notes for more details.

Hierarchical clustering solves all these issues and even allows you a metric by which to cluster. Hierarchical clustering is polynomial time, the final clusters are always the same depending on your metric, and the number of clusters is not at all a problem.

The main idea of hierarchical clustering is to not think of clustering as having groups to begin with. The way I think of it is assigning each data point a bubble. Over time these bubble's grow and fuse and continue to grow and fuse until there is one large bubble. Hiearchical clustering's goal is to construct a tree of fusions between clusters so that you can specify the number of clusters and it will give you a snapshot of how the data is assigned for those number of clusters.

A big time advantage with this tree approach is that if you want to optimize over some metric (average distance between points in each cluster plus the number of clusters), you have snapshots for each possible number of clusters. So optimizing simply becomes a task of calculating the metric for each snapshot.

How It Works

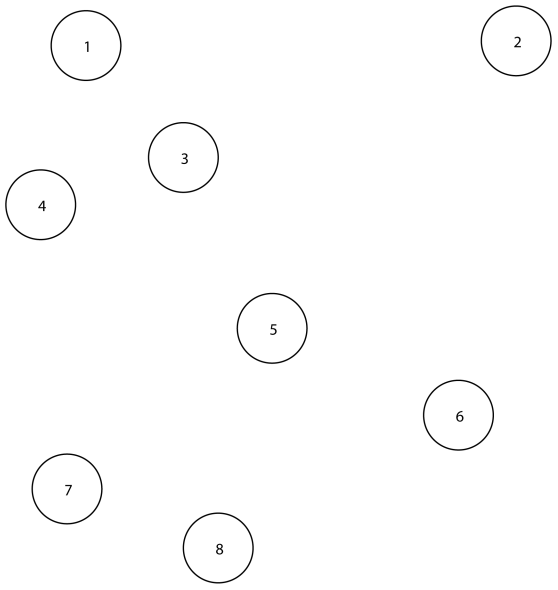
A great way to think about hierarchical clustering is through induction. The idea is if I have k clusters based on my metric it will fuse two clusters to form $k-1$ clusters. To begin with every point is its own cluster. So by induction we have snapshots for n clusters all the way down to 1 cluster.

I have been using some metric often. Now it is time to define "metric". There are many possible metrics but I'll focus on 4 common metrics and use an example to show how they behave.

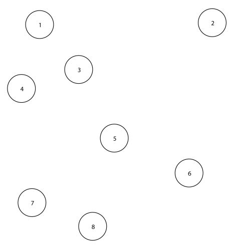
Metrics:

1. **Closest Points Between Clusters**: The distance between to clusters is defined as the smallest distance between a pair of the data points within the clusters. This is also known as *single linkage* [**Victor Lavrenko**].
2. **Farthest Points Between Clusters**: The distance between to clusters is defined as the largest distance between a pair of the data points within the clusters. This is also known as *complete linkage* [**Victor Lavrenko**].
3. **Average Pairwise Distance Between Clusters**: It is what it sounds like. The distance between to clusters is defined by the average distance between a pair of the data points within the clusters.
4. **Ward's Method**: The goal of Ward's method is to minimize the variance within each cluster. This is done by defining distance as the difference in cluster variance by fusing the clusters.

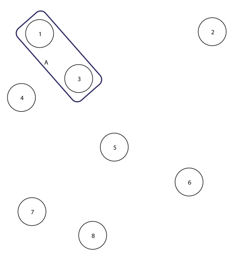
Lets play with the following example:



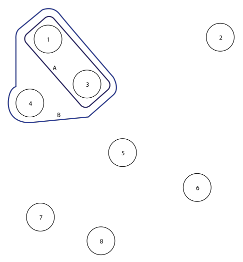
1. Metric: Closest Points: The progression of clusters. The farthest out lines defines the cluster.



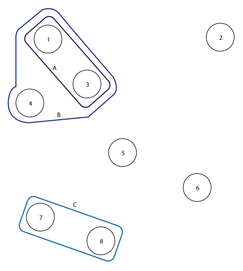
(a) 8 clusters; Initialization



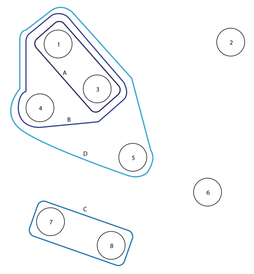
(b) 7 clusters



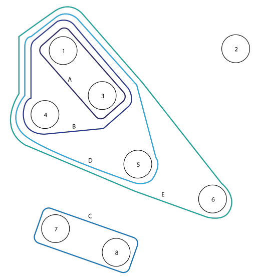
(c) 6 clusters



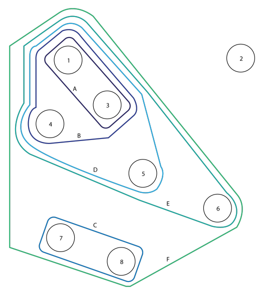
(d) 5 clusters



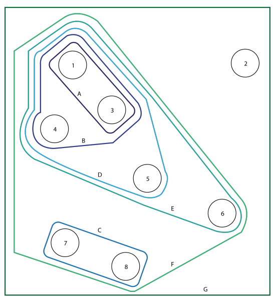
(e) 4 clusters



(f) 3 clusters

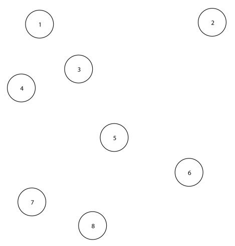


(g) 2 clusters

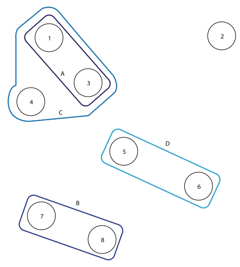


(h) 1 cluster

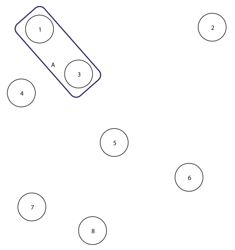
2. Metric: Farthest Points: The progression of clusters. The farthest out lines defines the cluster.



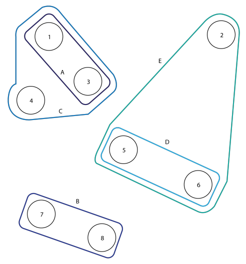
(a) 8 clusters; Initialization



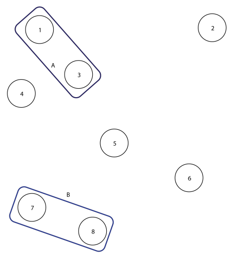
(e) 4 clusters



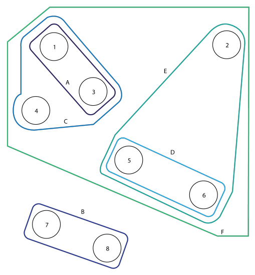
(b) 7 clusters



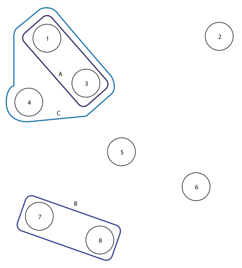
(f) 3 clusters



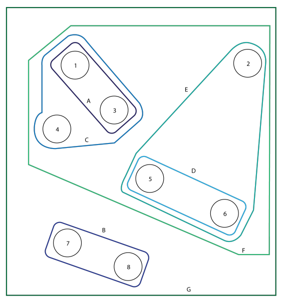
(c) 6 clusters



(g) 2 clusters

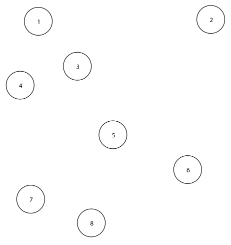


(d) 5 clusters

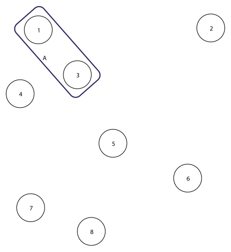


(h) 1 cluster

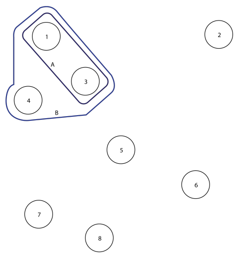
3. Metric: Average Distance: The progression of clusters. The farthest out lines defines the cluster.



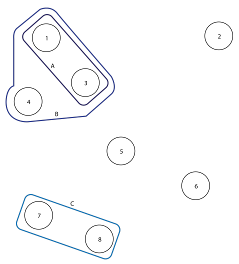
(a) 8 clusters; Initialization



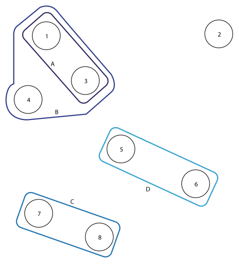
(b) 7 clusters



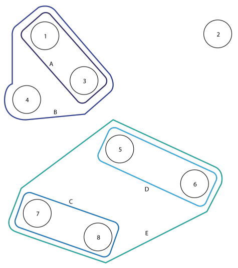
(c) 6 clusters



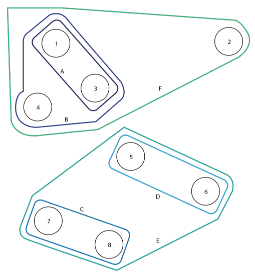
(d) 5 clusters



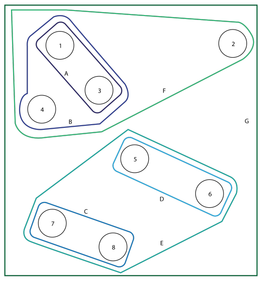
(e) 4 clusters



(f) 3 clusters

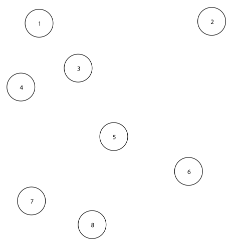


(g) 2 clusters

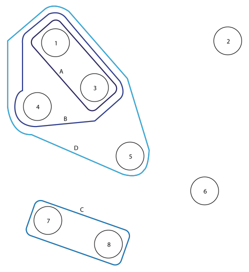


(h) 1 cluster

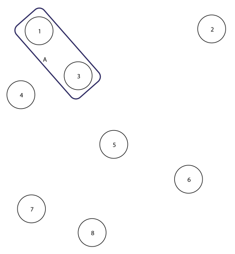
4. Metric: Ward's: The progression of clusters. The farthest out lines defines the cluster.



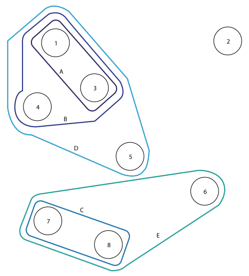
(a) 8 clusters; Initialization



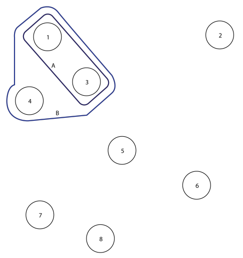
(e) 4 clusters



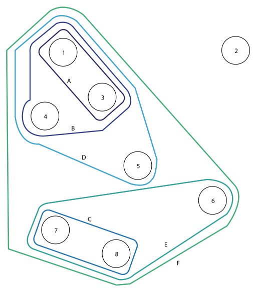
(b) 7 clusters



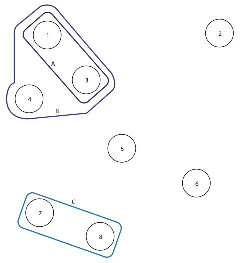
(f) 3 clusters



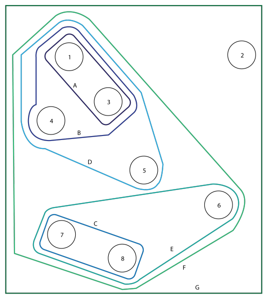
(c) 6 clusters



(g) 2 clusters



(d) 5 clusters



(h) 1 cluster

Lance-Williams Algorithm

So of course you could implement each of these algorithms in the standard way. Simply maintain a priority queue of pairwise distances between clusters and whenever you fuse clusters simply update all the associated pairwise distances associated with those clusters in the queue.

However, the downside of this approach is you would need different functions entirely in code to deal with each type of distance metric. Wouldn't it be nicer to just write one code snippet and have it adjust certain parameters based on the distance metric being used. The Lance-Williams algorithm does just this.

The idea is to update these distances in a recursive manner at each step. The general formula is as follows, where $d(C_a, C_b)$ is the distance defined between cluster a and cluster b [*Gordon, A. D. (1999), Classification, 2nd Edition, Chapman and Hall, Boca Raton.*]:

$$d(C_i \cup C_j, C_k) = \alpha_i d(C_i, C_k) + \alpha_j d(C_j, C_k) + \beta d(C_i, C_j) + \gamma |d(C_i, C_k) - d(C_j, C_k)| \tag{1}$$

For each of the metrics we can compute the parameters α_i, α_j, β, and γ. Note n_i represents the numbers of points in cluster i.

Closest Point/Single Linkage

$$d(C_i \cup C_j, C_k) = 0.5d(C_i, C_k) + 0.5d(C_j, C_k) - 0.5|d(C_i, C_k) - d(C_j, C_k)| \tag{2}$$

Therefore we have:

$$\begin{aligned} \alpha_i &= 0.5 \\ \alpha_j &= 0.5 \\ \beta &= 0 \\ \gamma &= -0.5 \end{aligned}$$

Farthest Point/Complete Linkage

$$d(C_i \cup C_j, C_k) = 0.5d(C_i, C_k) + 0.5d(C_j, C_k) + 0.5|d(C_i, C_k) - d(C_j, C_k)| \tag{3}$$

Therefore we have:

$$\begin{aligned} \alpha_i = 0.5\alpha_j &= 0.5 \\ \beta &= 0 \\ \gamma &= 0.5 \end{aligned}$$

Average Distance

$$d(C_i \cup C_j, C_k) = \frac{n_i}{n_i + n_j} d(C_i, C_k) + \frac{n_j}{n_i + n_j} d(C_j, C_k) \tag{4}$$

Therefore we have:

$$\begin{aligned}\alpha_i &= \frac{n_i}{n_i + n_j} \\ \alpha_j &= \frac{n_j}{n_i + n_j} \\ \beta &= 0 \\ \gamma &= 0\end{aligned}$$

Wards

$$d(C_i \cup C_j, C_k) = \frac{n_i + n_k}{n_i + n_j + n_k} d(C_i, C_k) + \frac{n_j + n_k}{n_i + n_j + n_k} d(C_j, C_k) - \frac{n_k}{n_i + n_j + n_k} d(C_i, C_j) \tag{5}$$

Therefore we have:

$$\begin{aligned}\alpha_i &= \frac{n_i + n_k}{n_i + n_j + n_k} \\ \alpha_j &= \frac{n_j + n_k}{n_i + n_j + n_k} \\ \beta &= -\frac{n_k}{n_i + n_j + n_k} \\ \gamma &= 0\end{aligned}$$