Hierarchical Agglomerative Clustering

(https://github.com/learn-co-curriculum/dsc-hierarchical-agglomerative-clustering) (https://github.com/learn-co-curriculum/dsc-hierarchical-agglomerative-clustering/issues/new)

Introduction

In this lesson, we'll learn about another popular class of clustering algorithms -- hierarchical agglomerative clustering!

Objectives

You will be able to:

- Explain the process behind hierarchical agglomerative clustering
- Describe the three different linkage criteria for hierarchical agglomerative clustering
- Define the purpose of a dendrogram
- Compare and contrast k-means and hierarchical agglomerative clustering methodologies

Understanding Hierarchical Clustering

So far, we've worked with a non-hierarchical clustering algorithm, k-means clustering. K-means works by taking a set parameter that tells it how many clusters we think exist in the data, and then uses the Expectation-Maximization (EM) algorithm to iteratively shift each cluster centroid to the best possible position by constantly calculating and recalculating the centroid's position by assigning each point to the cluster centroid they are closest to with each new step, and then moving the centroid to the center of all the points currently assigned to that centroid. With non-hierarchical algorithms, there can be no subgroups -- that is, no clusters within clusters.

This is where agglomerative clustering algorithms come in. In agglomerative clustering, the algorithm starts with n clusters (where n is the number of data points) and proceeds by merging the most similar clusters, until some stopping criterion. in `scikit-learn`, the stopping criterion that is implemented is "number of clusters". If left alone, the algorithm will work until it has merged every cluster into one giant cluster. We can also set the limit, if we want, to stop when there are only [x] clusters remaining.

Linking Similar Clusters Together

Several linkage criteria that have different definitions for "most similar clusters" can be used. The measure is always defined between two existing clusters up until that point, so the later in the algorithms, the bigger the clusters get.

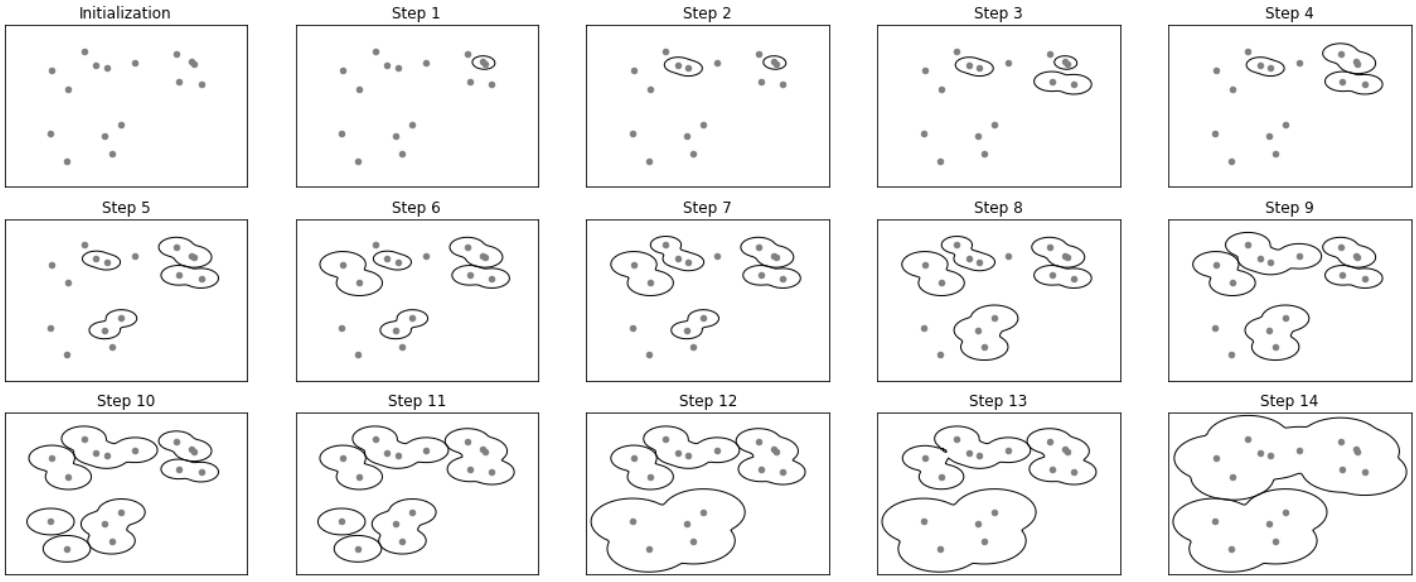
Scikit-learn provides three linkage criteria:

- **ward** (default): picks the two clusters to merge in a way that the variance within all clusters increases the least. Generally, this leads to clusters that are fairly equally sized.
- **average**: merges the two clusters that have the smallest ***average*** distance between all the points.
- **complete** (or maximum linkage): merges the two clusters that have the smallest ***maximum*** distance between their points.

As we'll see in the next lab, these linkage criteria can definitely have an effect on how the clustering algorithm performs. As always seems to be the case, no one of these is "best" -- which one you should use often depends on the structure of your data, and/or your own goals.

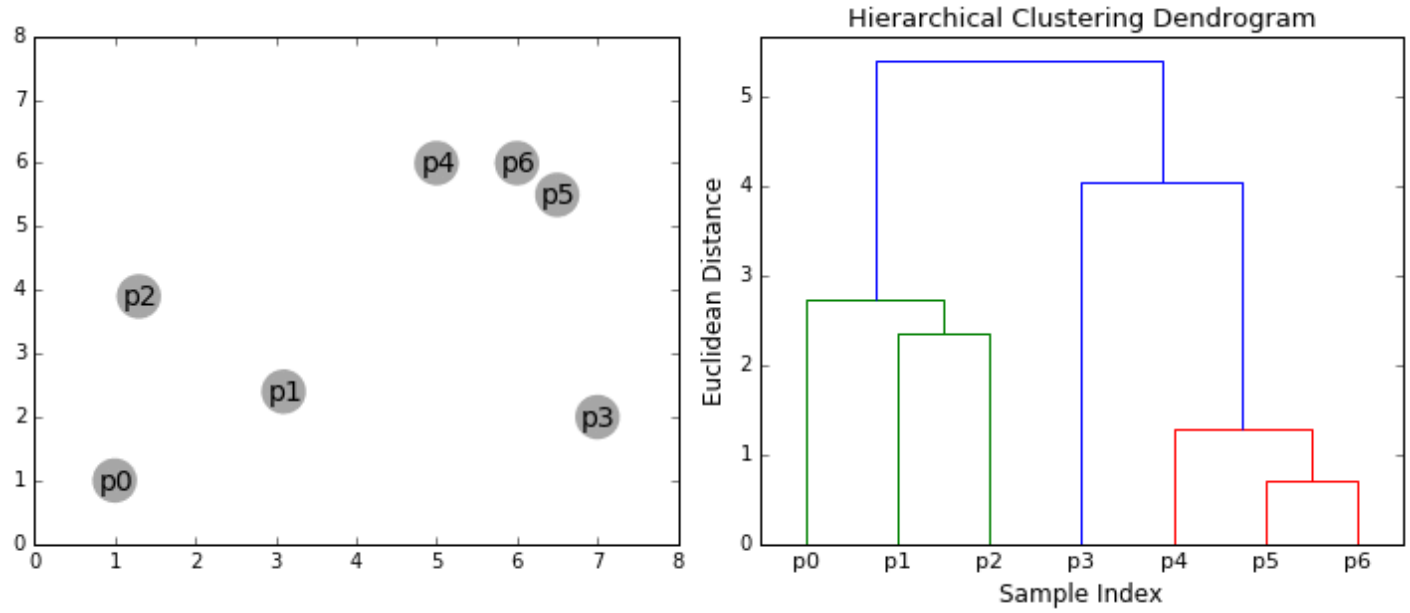
A Visual Example

It's often easier to understand what the HAC algorithm is doing when we look at the decisions it makes at each given step. The following diagram demonstrates the clusters created at each step for a dataset of 16 points. Take a look at the diagram and see if you can figure out what the algorithm is doing at each step as it merges clusters together:



As we can see from the diagram above, in each step, the algorithm takes the two clusters that are closest together (and remember, we define "closest together" according to whichever linkage criteria we choose to use), and then ***merge*** those two clusters together into a single cluster. We don't move the data points or anything like that -- we just consider them as a single unit, as opposed to two separate ones. This works at every stage because in the beginning, we treat each data point as a unique cluster.

This becomes very intuitive when we look at the following gif -- pay attention to the image on the left:

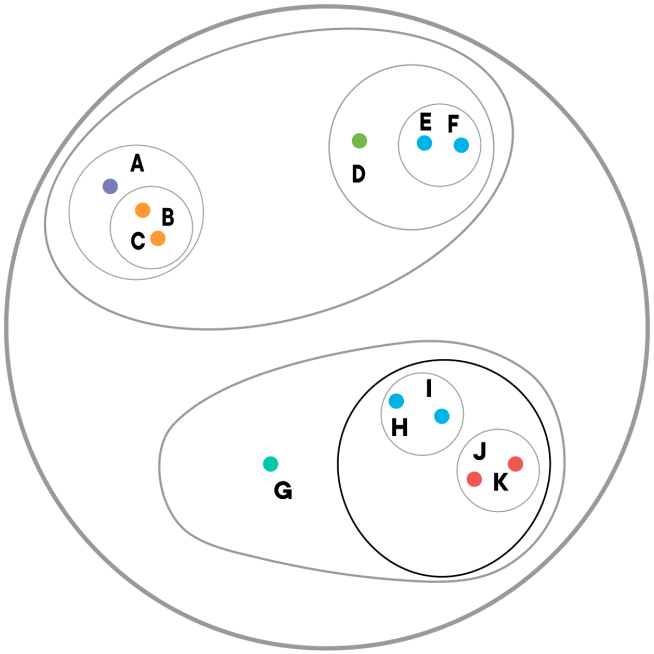


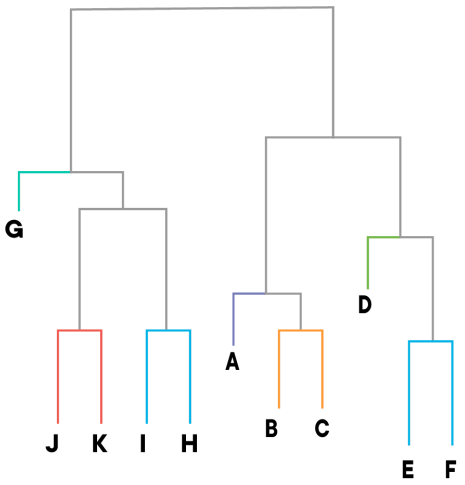
As the dots disappear, the visualization is replacing them with the newly calculated center of that cluster, which will be used for linkage purposes. Now, let's end this lesson by talking about visualizations we can use to interpret results!

Dendrograms and Clustergrams

One advantage of HAC is that we can easily visualize the results ***at any given step*** using visualizations such as ***Dendrograms*** and ***Clustergrams***. Take another look at the gif above, but this time, pay attention to the image on the right. This is a *dendrogram,* which is used to visualize the hierarchical relationship between the various clusters that are computed throughout each step. Dendrograms are very useful to decide how clusters change depending on the euclidean distance. If you decide that your intra-cluster euclidean distance should be smaller than 3, you can draw a horizontal line at euclidean distance 3, and define which points belong to which cluster by looking at the dendrogram. For the gif above, this means that there are three clusters: cluster one contains p_0, p_1 and p_2, cluster two contains p_3, cluster three contains p_4, p_5 and p_6.

We can also visualize the same information by drawing lines representing each cluster at each step to create a *clustergram*. Take a look at the following diagram below, which shows both a dendrogram and clustergram of the same HAC results:





How is HAC used?

HAC algorithms are used in generally the same way that K-means and other clustering algorithms are used: for tasks such as market segmentation, or for gaining a deeper understanding of a dataset through cluster analysis. However, there are special cases of things that fit quite well in a hierarchical agglomerative structure -- one of the most common use cases you'll see for HAC is the way that smartphones naturally sort photos inside their photos app! Take a look at your photos app on your phone, and the albums that it creates for you -- you'll likely see that the albums are sorted in a ***hierarchical*** fashion! Perhaps the phone chooses to group photos by date first, and then by location, or even content! In this way, these can be viewed as natural clusters within clusters, in a way that makes intuitive sense to users. When we browse, we likely want to see photos that were taken around the same time, and then at the same place, and then narrow it down to photos about the same things, to quickly browse and find what we're looking for. This is a great example of HAC being used in the wild!

Summary

In this lesson, we learned about how the HAC algorithm derives its clusters, including different linkage criteria that can be used to determine which clusters should be merged at any given point. We also examined some visualizations of HAC algorithms, in the forms of dendrograms and clustergrams!