# Hierarchical clustering

For our lab, we want to develop the most efficient way to group data based based on a measure of dissimilarity. We explored the topic of hierarchical clustering which is used in data mining and statistics. It is visualized useing a dendogram to show the organization and relationship among clusters. Hierarchical clustering can be implemented in two ways: divisive and agglomerative. We will elaborate on these two variants in this paper to compare the its algorithmic complexity, or how much resources are needed when computed, and the different types of cluster linkage criterion. Further information on cluster analysis can be found in [**?**].

## Divisive and Agglomerative

Divisive clustering is also known as the "top down" method. This takes in a data set as one large cluster. Then as it moves down the hierarchy, it recusively splits until it reaches the leaves of the tree, i.e. each observation is its own singleton cluster.

Agglomerative clustering can be thought to do the opposite of divisive clustering and is known as a "bottom up" strategy. It takes in a data set and it initially looks at each observation as its own singleton cluster. Then, based on its linkage criterion, it will merge recursively to form new clusters until all the data is in one large cluster.

A dendrogram can be used in both cases to record and visualize the hierarchical nature of the clusters and their distance to each other.

## Cluster Dissimilarity Measurements

In forming clusters, a measurement must be made to determine which clusters split or merge in regards to divisive or agglomerative clustering, respectively. There are many different ways to measure dissimilarity but this paper will explore single linkage, complete linkage, and centroid linkage. Each of the measurements below refer to ways in measuring distances between $k$ number of clusters $c_{i=1...k}$ in relationship with specific points $x_{i=1...k}$ in those clusters.

### Single Linkage

Single linkage takes the distance between the closest observations, or minimum distance, in each cluster and merges the those with the shortest distance. This works well for data sets that desire to cluster together long chained data points. Let $D(c_i, c_j)$ be the distance between two centroids for $i, j \in \{1, 2, ...k\}$, and $D(x_i, x_j)$ be the distance between two data points in $i, j \in \{1, 2, ...n\}$. Then single linkage defines the following relationship:

$$D(c_1, c_2) = \min_{x_1 \in c_1, x_2 \in c_2} D(x_1, x_2)$$

### Complete Linkage

Complete linkage is similar to single linkage. It takes the distance between the farthest observations, or the maximum distance, in each cluster. The clusters that have the shortest measured distance is merged together. This measurement is ideal for data that is sphereically organized relative to each other. The relationship is defined as

$$D(c_1, c_2) = \underset{x_1 \in c_1, x_2 \in c_2}{max} D(x_1, x_2)$$

### Centroid Linkage

Using centroid linkage involves taking the average of all points in a cluster and assigning that value as the mean cenroid. Then the centroids with the shortest distances are merged together. Centroid linkage is defined as

$$D(c_1, c_2) = D((\frac{1}{|c_1|} \sum_{x \in c_1} \vec{x}), (\frac{1}{|c_2|} \sum_{x \in c_2} \vec{x}))$$

## Our Implementation

We developed both hierarchical divisive and agglomerative clustering functions to test which would organize data the fastest. In our particular application, instead of flat clustering, we had to cluster over the curved surface of a sphere. In order to do this, we utilized the SPKmeans function [**?**]. We also chose our linkage criterion to be based on centroid linkage.

Essentially, we developed the divisive clustering by following the intuition mentioned above. For our agglomerative clustering implementation, we did a slightly different variation hoping its run time would be slightly better than the traditional method. In our code, intuitively, we initiated each point as its own cluster with its own centroid. Then we recursively merged two clusters together based on centroid linkage to form a new cluster with its own cluster centroid everytime. This process stops when there is a single point to represent the entire data's average centroid.

We tested with a relatively small, three dimensional data set of 150 points. In both divisive and agglomerative, we input different cluster values into SPKmeans which determines how often the function gets called. In the divisive method, we asked SPKmeans to return two clusters. This means the number of times we call SPKmeans is based on how many nodes we have which is $m = 2n - 1$ where $n$ is the number of data points. For the agglomerative case, we asked SPKmeans to return the $n/2$ clusters. This means means we had to call SPKmeans at every level which is $l = \lceil \log_2 n \rceil$.

As a benchmark, we also used the MATLAB `linkage` function to construct a pairwise-distance based tree (merging nodes together by single-linkage) and compared the results.

## Implementation details

In $k$-means, there is a choice of initial mean initialization. This affects the initial distribution of means and therefore potentially the number of iterations until convergence, but also may be expensive computationally. For instance, a good setting of the initial means will reduce the likelihood of empty clusters found, which is dealt with by splitting the largest cluster and is therefore computationally expensive.

For the agglomerative algorithm, it makes sense to spend time initializing the means (we chose to maximize the distance between the initial means), since the number of calls to SPKMeans is small and the number of clusters for each call is high. For the divisive algorithm, it doesn't make sense to incur this expense when SPKMeans is called many times with only the branching-factor number of clusters (generally a small constant).

This optimization (or lack thereof) substantially reduces the running time for both algorithms.

## Algorithm complexity analysis

Let $n$ be the number of data points, $k$ be the number of clusters (number of means), $d$ be the dimensionality of the data, and $i$ be the number of iterations until convergence. The runtime of the $k$-means algorithm (including spherical k-means) is $O(nkdi)$.

Let us assume that the cost of initializing the means and splitting empty clusters is negligible, i.e. $O(1)$. In practice, the two are related: we find that explicitly picking good starting means for the agglomerative algorithm (e.g. randomly, or by maximizing distance between the means) makes the probability of empty clusters small, whereas it becomes excessive and unnecessary for the divisive algorithm. See the **Implementation details** section for more discussion.

We will also assume that $d$ and $i$ are constants. Though in degenerate data sets $i$ can be an exponential function of $n$, in practice $i$ is a small constant on datasets with clustering structuring.

Let us also assume that we choose a branching factor of 2, i.e. a binary hierarchical tree structure. The following analysis does not depend on the branching factor, assuming it is constant.

**Theorem.** ***Agglomerative complexity***

$$T_{aggl}(n) = \Theta(n^2 di)$$

**Proof.** Our agglomerative algorithm has a recurrence relation of the following form:

$$T_{aggl}(n) = T(\frac{n}{2}) + nkdi$$

for some constant $c$, since we divide the problem size into two for every recursive call. But we choose $k = \frac{n}{2}$ each time, since the number of means we choose to cluster with is also half of $n$.

| Master theorem variable | Value |
|---|---|
| $a$ | 1 |
| $b$ | 2 |
| $f(n)$ | $\frac{n^2 di}{2}$ |
| $c$ | 2 |
| $log_b(a)$ | 0 |

We use the master theorem[**?**] to solve this recurrence relation:
This satisfies case 3 of the master theorem:

$$f(n) \in \Omega(n^c) \text{ s.t. } c > log_b(a)$$

since $f(n) \in \Omega(n^2)$ s.t. $c = 2 > log_b(a) = 0$, and

$$af(\frac{n}{b}) \leq k_0 f(n) \text{ for some } k_0 < 1 \text{ and sufficiently large } n$$

since when $k_0 = \frac{1}{4}$, $\left\{ f(\frac{n}{2}) = (\frac{1}{4})\frac{n^2 di}{2} \right\} \leq \left\{ k_0 f(n) = (\frac{1}{4})\frac{n^2 di}{2} \right\}$
and so, by the master theorem: $T_{aggl}(n) = \Theta(n^2 di)$. □

**Theorem.** ***Divisive complexity***

$$T_{div}(n) = \Theta(n \log(n) di)$$

**Proof.** Our divisive algorithm has a recurrence relation of the following form:

$$T_{div}(n) = 2T(\frac{n}{2}) + nkdi$$

for some constant $c$, since we divide the problem size into two for every recursive call and we have two subproblems each time. But we choose $k = 2$ each time, since the number of means we choose to cluster with is always the branching factor.

We use the master theorem[**?**] to solve this recurrence relation:

| Master theorem variable | Value |
|---|---|
| $a$ | 2 |
| $b$ | 2 |
| $f(n)$ | $n(2)di$ |
| $c$ | 1 |
| $log_b(a)$ | 1 |

This satisfies case 2 of the master theorem:

$$f(n) \in \Theta(n^c \log_0^k n) \text{ s.t. } c = log_b(a)$$

by letting $k_0 = 0$, since $f(n \log^0(n)) = f(n) \in \Theta(n)$ s.t. $c = 1 = log_b(a) = 1$.
and so, by the master theorem: $T_{div}(n) = \Theta(n \log(n) di)$. □