Chapter 12
More Clustering Methods

Slides (pdf) Video (Panopto)

Recall in the last chapter that we introduced the concept of a **cluster**, a centroid with a spread of a specific shape and size. We also went over some limitations of the k-means algorithm. In this chapter we will look at a more diverse set of clustering techniques that allow us to overcome some of the limitations of k-means.

12.1 Mixture Models

One example of a mode of clustering that can generalize better than the k-means algorithm is a mixture model. Each cluster can have its own probability distribution. One example of a probability distribution for clusters is **Gaussian Mixtures**. The best shape and size for each cluster is learned via a technique called **Expectation Maximization**, though we will not go in depth into it. Since Guassian Mixtures allow for different shapes and sizes of clusters, we can allow **soft assignments** to clusters. This means that every datapoint has a probability associated with how much it belongs to a cluster.

Example(s)

A **soft assignment** may be useful for categorizing news articles. A news article could have a 54% chance it is about world news, 45% chance it is about science, and 1% chance it is a conspiracy theory.

Therefore, this leaves us with two types of clustering.

12.1.1 Hard Clustering

Hard clustering is clustering that has no overlaps. An example of hard clustering is K-means, because every datapoint belongs to at most one cluster. Hard clustering is a subset of soft clustering.

12.1.2 Soft Clustering

Soft clustering does allow for overlap, though clusters do not necessarily have to overlap. As Gaussian Mixtures use probabilities to cluster data, clusters may overlap.

When using a Gaussian Distribution to model our clusters, each cluster is defined by a mean μ and a covariance σ. Hence, clusters can be unique in shape and size.

12.2 Hierarchical Clustering

Hierarchical Clustering takes advantage data that might have a hierarchical structure.

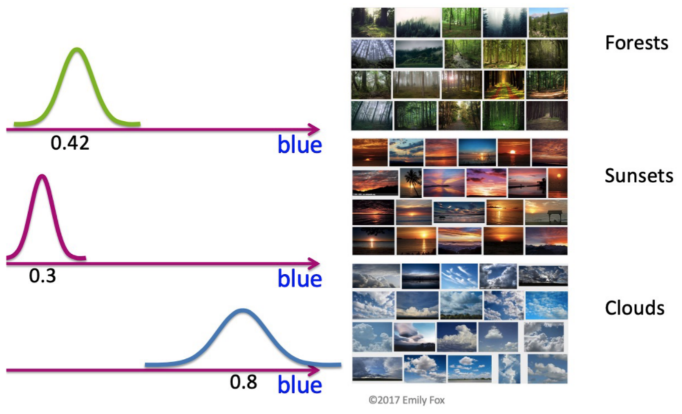


Figure 12.1: Taking a look at how "blue" each pixel of an image is, we have soft clusters resulting from a Guassian distribution.

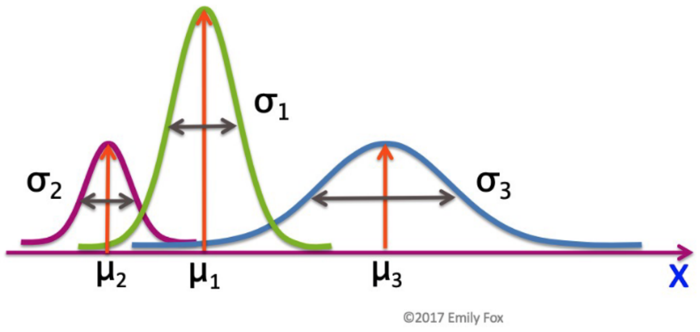


Figure 12.2: Gaussian mixture components represent unique clusters specified by mean and covariance.

Definition 12.1: Dendrogram

A **dendrogram** is a type of tree diagram that is used for a hierarchical ordering of things.

We can organize data into a hierarchical structure using one of the two approaches:

Definition 12.2: Divisive

The **divisive** algorithm, a.k.a. **top-down** starts with all the data in one big cluster and then recursively splits the data into smaller clusters.

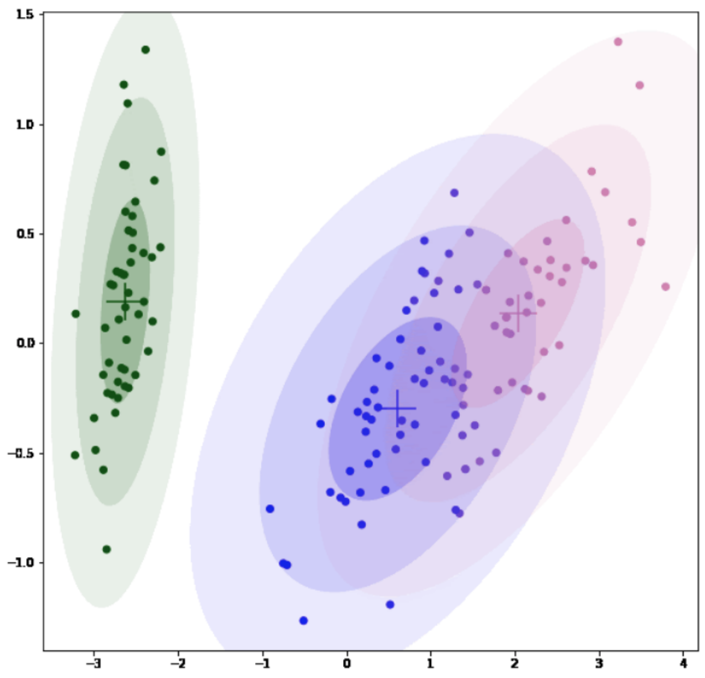


Figure 12.3: The distribution of each cluster gives it a unique shape.

Definition 12.3: Agglomerative

The **agglomerative** algorithm, a.k.a. **bottom-up** starts with each datapoint in its own cluster and merges clusters until all points are in one big cluster.

12.2.1 Divisive Clustering

To structure our data into a dendrogram, we might adapt the k-means algorithm to do divisive clustering. We could run the k-means algorithm on one big cluster, and then running it on each of those smaller clusters, and keeping going until we have the desired number of sub-clusters. This is one option out of many divisive clustering techniques as there are a number of choices we must consider:

- **Algorithm:** K-means? Or something else?
- **Clusters per split:** How many subclusters result from splitting one cluster?
- **Stopping:** When do we stop splitting? It could be when we reach a **max cluster size**, or a **max cluster radius**, or a **specified number of clusters.**

12.2.2 Agglomerative Clustering

Agglomerative clustering can be summarized by the following algorithm:

1. Initialize each point in its own cluster.

2. Define a distance metric between clusters.

3. While there is more than one cluster present, merge the two closest clusters.

With agglomerative clustering, we have to make the choice of which distance metric to use. As an example, we can use the **single linkage** metric, which determines the distance between two clusters as the distance between the closest two points of those clusters:

$$distance(C_1, C_2) = min_{x_i \in C_1, x_j \in C_2} d(x_i, x_j) \tag{12.1}$$

The above equation states that the distance between Cluster 1 and Cluster 2 is some x_i in Cluster 1 and some x_j in Cluster 2 where those two points are the closest possible pair between the two clusters.

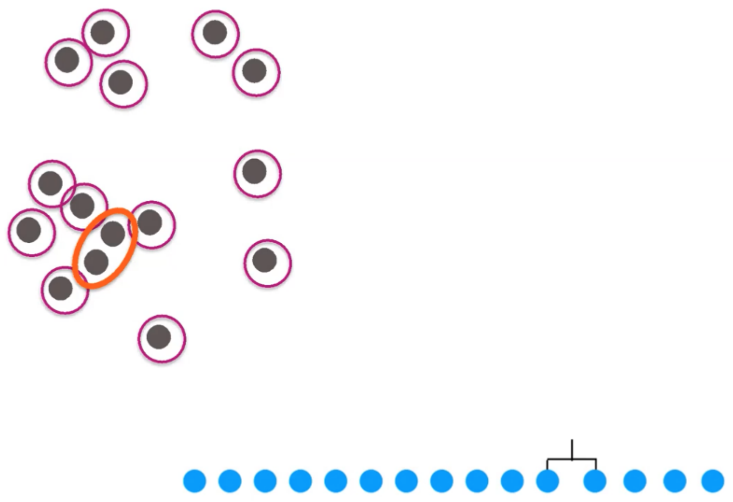


Figure 12.4: In the above figure, we have a set of datapoints in the top left. We want to organize them into a dendrogram (in blue) using agglomerative clustering, so we will group datapoints together in the dendrogram (bottom right) as we merge clusters (top left). To start off, every cluster has one single datapoint. After merging two of the closest clusters together, notice how one cluster (in red) now has two datapoints. A branch on the dendrogram connecting these datapoints has been drawn to reflect this change.

Since the single linkage distance metric compares the two closest points between two clusters, we are able to draw complex cluster shapes.

Now, to figure out which two clusters to merge, we have to compare the closest datapoint pairs between every combination of clusters. Once we find the closest datapoint pairs between the two closest clusters, we merge these clusters and repeat until there is only one cluster. As we keep merging until we get one cluster, we complete our dendrogram. One thing to notice is that the higher the connecting line between two point is in the dendrogram, the further away they are in the cluster.

This distance measure is important because it allows us to mark specific heights within our dendrogram that specify cluster separations, if we want to preserve some clusters.

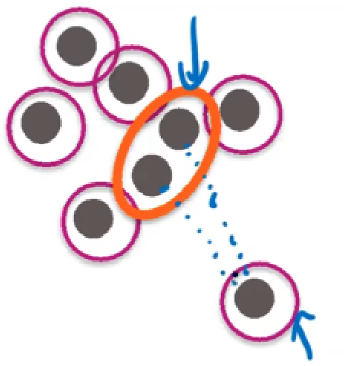


Figure 12.5: Now that one of our clusters has more than one datapoint in it, we can utilize the single linkage distance metric. If we want to measure the distance between the red cluster A and the cluster at the bottom-right B, we have two points to compare. Clearly, the datapoint at the bottom of the red cluster A is closer to cluster B.

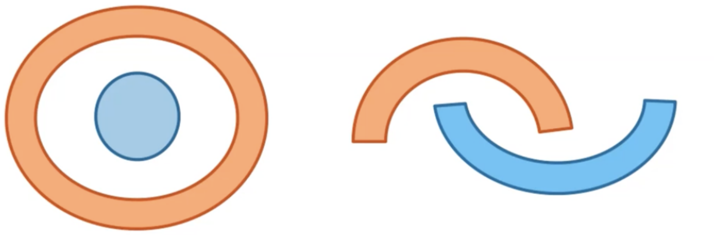
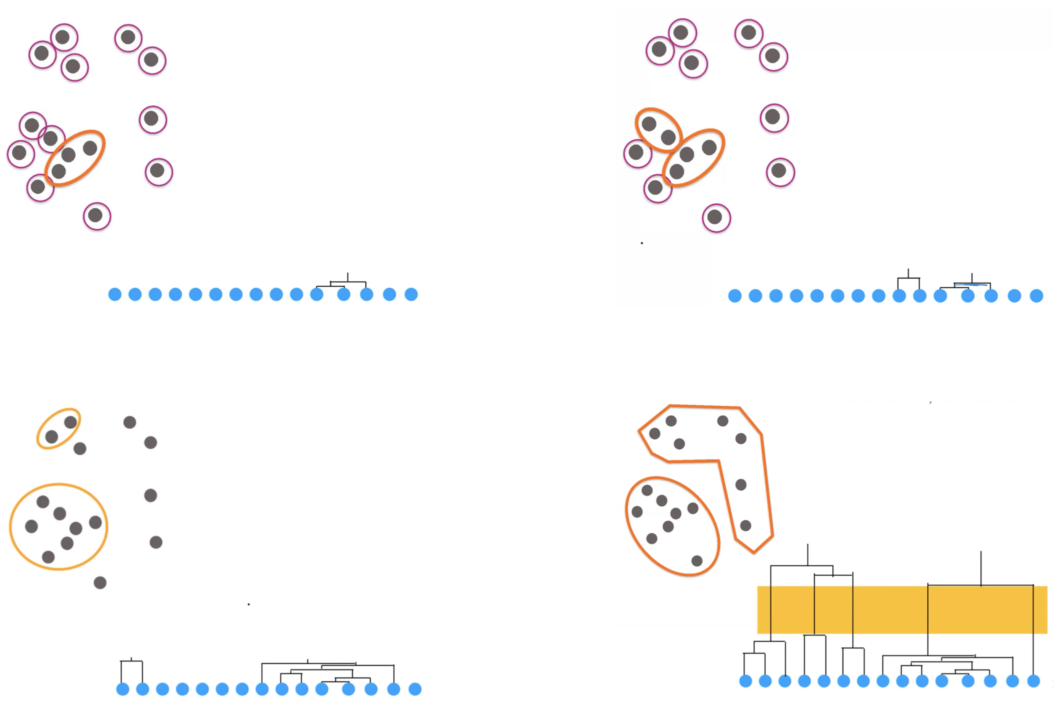


Figure 12.6: A doughnut and a crescent clustering achieved by the single-linkage distance metric in agglomerative clustering. Due to single linkage, before every clustering, we look at the two closest points between the two closest clusters. Because of this, the ring that is the doughnut shape can be formed distinctly from the doughnut's center, as every time we add a point to the ring, it happens to lie on the circumference of the ring and is closer to the ring than the center of the doughnut is to any of the ring points.

12.2.2.1 Final Notes on Agglomerative Clustering

There are a number of choices we must make when considering the practical applications of agglomerative clustering. Firstly, the single linkage distance metric isn't the only metric there is. We could also use a **complete linkage** or a **centroid linkage**.The complete linkage computes the distances of clusters via the *farthest* points of each cluster away from each other, and the centroid linkage uses the center point of each cluster. We also have to think about how to define our D^*, or where to cut the dendrogram, i.e. how many clusters to preserve. In general, fewer clusters are more interpretable, but if we only have one cluster then we might as well not have clustered anything at all. This is why we cut the dendrogram, so that we can highlight key categories of the data. Finally, we cannot forget about outliers. We might set a distance threshold to pick out outliers, or come up with a more complex algorithm all together. The key takeaway from all of these choices we must make is that in the end there is no one correct answer, but rather what yields the most practical and interpretable results given our data and purposes.

The runtime of the algorithm we described above is $O(n^2 log(n))$. In this class you do not need to fully



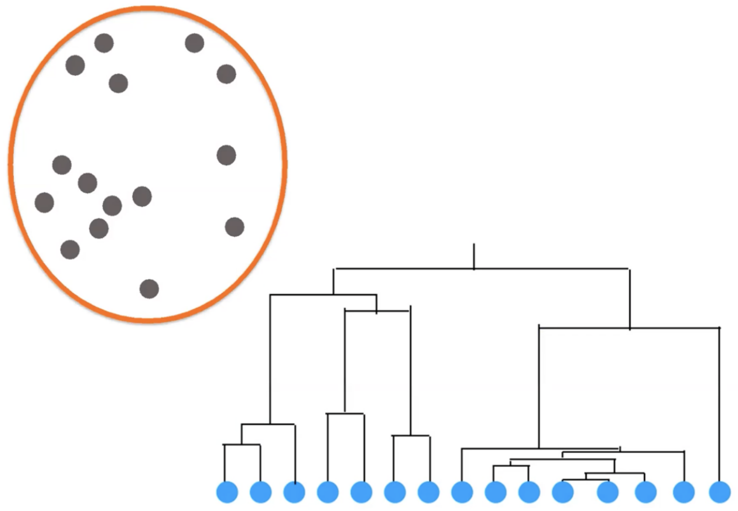


Figure 12.7: The final cluster, with the completed dendrogram.

understand runtime analysis, but this runtime comes from the fact that this algorithm compares every single point to every single other point, and sorts each of those comparisons. The **triangle inequality**, a theorem that states that "for any triangle, the sum of the lengths of any two sides must be greater than or equal to the length of the remaining side", allows us to bypass some of these comparisons, making the best known optimal runtime for agglomerative clustering $O(n^2)$.

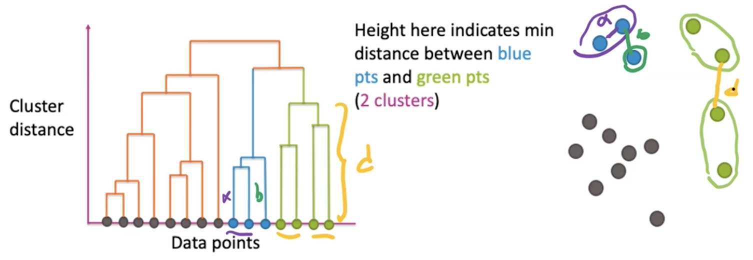

Cluster distance
Data points
Height here indicates min distance between blue pts and green pts (2 clusters)
α
b
d

Figure 12.8: The height of the branches that connect datapoints gives is proportional to the distance between clusters. See distances of α, b, d.

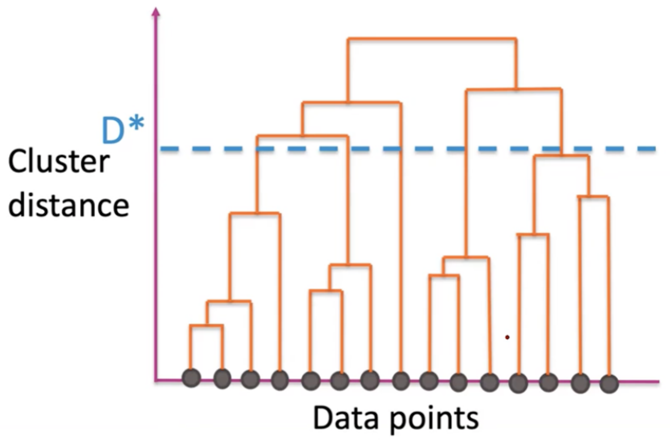


Figure 12.9: If we want to preserve some clusters, we can "cut" the dendrogram at a specific height D^* depending on how many clusters we want. The above example would preserve 5 unique clusters.

12.3 Missing Data

In all types of data analysis, the problem of missing data is extremely common. In this section we will discuss our options in the face of missing data.

12.3.1 Strategy 1: Skip

We can simply ignore missing data by discarding data with missing elements in it.
Pros:

- Very easy to understand.
- Can be applied to any model.

Cons:

- We could be removing useful information.

- We don't know if we should remove one datapoint with a missing feature, or remove that feature from the dataset entirely to keep the datapoint.

12.3.2 Strategy 2: Sentinel Values

Instead of cutting out missing data, we can replace it with a value that indicates it is missing.
Pros:

- Very easy to understand.
- Works well with categorical data, as "missing" becomes its own category.

Cons:

- Does not work well with numeric features.

12.3.3 Strategy 3: Imputation

With imputation, we utilize some heuristic to "predict' what a missing value may be. A simple approach for categorical data could be simply just using the most popular category as the value for the missing data, and for numeric data it could be using the median or the mode. A more complex approach could be using machine learning to estimate a prediction of what that missing value should be based on the relationship between other features in the data.

Pros:

- Simple to implement if using the simple approach.
- Can be applied to any model.

Cons:

- May result in systematic errors. Maybe there is a deeper-rooted issue for why the values are missing, and we shouldn't just fill them in as if nothing happened.

12.3.4 Strategy 4: Modify Algorithm

We can completely redesign the model so that it accounts for missing values. For example, we could recreate a tree such that it has a special branch for missing values.

Pros:

- This is very similar to the sentinel value approach so it is likewise simple.
- It works for numeric features.
- It can yield more accurate predictions.

Cons:

- You have to implement an entirely new model. This could be pretty difficult for some model types.