Lecture 10: Semantic Segmentation and Clustering

Vineet Kosaraju, Davy Ragland, Adrien Truong, Effie Nehoran, Maneekwan Toyungyernsub
Department of Computer Science
Stanford University
Stanford, CA 94305
{vineetk, dragland, aqtruong, effie, maneekwt}@cs.stanford.edu

1 Clustering and Segmentation

One task in computer vision is image segmentation. The goal of image segmentation is to identify groups of pixels that go together. Pixels can be similar to each other in some aspect, whether it be texture, color, or some other feature. An example of image segmentation is pictured below. In figure 1, we'd like to be able to group all the pixels that make up the tiger, all the pixels that make up the grass, all the pixels that make up the sky, and all the pixels that make up the sand. We can see the resulting segmentation in figure 2.

Figure 1: Input image. Source: lecture 12, slide 4

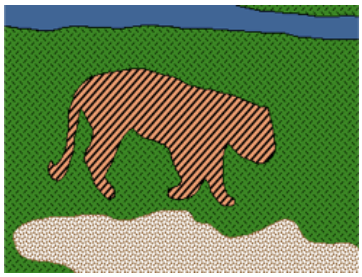

Figure 2: Output segmentation. Source: lecture 12, slide 4

This is one example of image segmentation. Other examples include grouping video frames into shots and determining what is a figure and what is the ground/background. By identifying groups of pixels that go together, we can separate an image into distinct objects. Beyond the immediate usefulness of recognizing objects, this can also allow for greater efficiency in any further processing we do.

2 Gestalt School and Factors

A lot of computer vision algorithms draw from areas outside of computer science, and image segmentation is no different. Computer vision researchers drew inspiration from the field of psychology, specifically the Gestalt Theory of Visual Perception. At a very high level, this theory states that "the whole is greater than the sum of its parts." The relationships between parts can yield new properties and features. In the theory, there are Gestalt factors. These factors are properties that can define groups. Below are examples of factors.

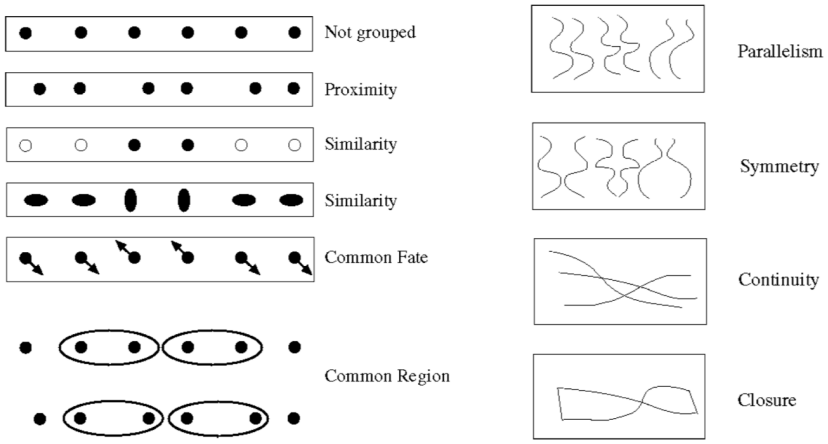

Figure 3: Examples of gestalt factors. Source: Forsyth & Ponce [1]

There are a few other factors that cannot be explained with just a picture. Take a look at the following image:

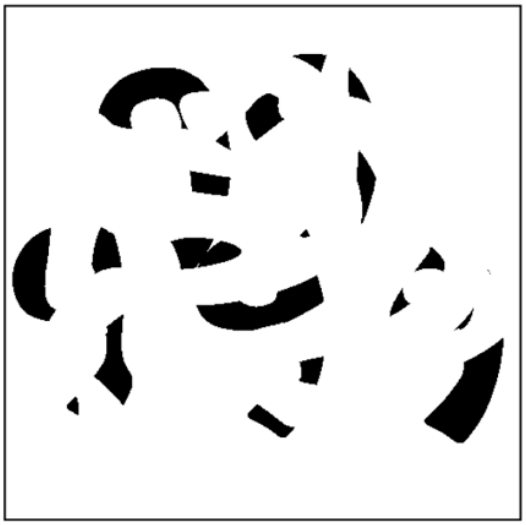

Figure 4: Source: Forsyth & Ponce [1]

Can you make out anything in this picture? Probably not. But how about now:

Figure 5: Source: Forsyth & Ponce [1]

Now, we can clearly see the image is a bunch of 9's occluded by random squiggly lines. This is an example of a continuity through occlusion cue. The grey squiggly lines give us a cue that the black pixels are not separate and should in fact be grouped together. And by grouping the black pixels together and not perceiving them as separate objects, our brain is able to recognize that this picture is a bunch of 9's being occluded by grey lines.

Here's another image:

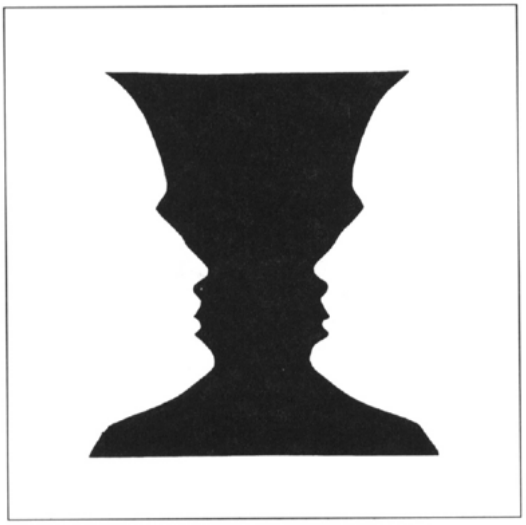

Figure 6: Source: Forsyth & Ponce [1]

What do you see? Do you see a cup or do you see the side of 2 heads facing each other? Either option is correct, depending on your perspective. This variation in perception is due to what we identify as the foreground and what we identify as the background. If we identify the black pixels as the foreground, we see a cup. But if we identify the white pixels as the foreground, we see 2 faces.

This is an overview of some of the factors we draw inspiration from for grouping. One thing to note however is that while these factors make intuitive sense, it's very hard to translate these factors into working algorithms.

So now the question is, how do we perform image segmentation? What algorithms do we use? One way to think about segmentation is clustering. We have a bunch of pixels and we want to assign each of them to a cluster. In the following sections, we will explore different methods of clustering.

3 Agglomerative Clustering

In general, clustering is an unsupervised learning technique where we have several items, $x_1, ..., x_n$, each of which are in R^D, and we are attempting to group them into clusters, without knowing what the correct answer is. Within clustering there are several kinds of algorithms that are commonly used. Out of these, one of the more common ones is called **agglomerative clustering**.

The general idea behind agglomerative clustering is to look at similarities between points to decide how these points should be grouped in a sensible manner. Before we discuss the details of the algorithm, we must first decide how we determine similarity.

3.1 Distance Measures

We approach the problem of determining similarity between objects as determining the distance between the objects: the smaller the distance, the more similar they are. Although there are several potential distance functions we can work with, in general, it's hard to know what makes a good distance metric, so people try to choose a standard, well-researched distance metric, such as the two examples below.

3.1.1 Euclidean Distance

One common measure of distance is the Euclidean distance, which measures distances between two data points, x and x' by taking into account the angle and the magnitude. We can write the Euclidean distance as the following (which is equivalent to the dot product):

$$sim(x, x') = x^T x' \tag{1}$$

Note that this distance measure doesn't normalize the vectors, so the magnitude of the vectors is factored into the measure of similarity.

3.1.2 Cosine Distance Measure

Another common measure of distance is the Cosine distance measure, which only accounts for the angle between two data points, x and x'. As its name implies, this measure relies on the cosine between the two points, as found by:

$$sim(x, x') = cos(\theta) \tag{2}$$

$$= \frac{x^T x'}{||x|| \cdot ||x'||} \tag{3}$$

$$= \frac{x^T x'}{\sqrt{x^T x}\sqrt{x'^T x'}} \tag{4}$$

Note that by dividing by the magnitudes of the vectors, we normalize the distance metric, and ensure it is only dependent on the angle between our objects.

3.2 Desirable Clustering Properties

Now that we've defined potential distance metrics, the next step is to choose a clustering technique. There are various properties of clustering methods that we might want to consider when choosing specific techniques:

1. **Scalable** - in terms of compute power & memory
2. **Different data types** - algorithm should support arbitrary data being in R^d for all d
3. **Input parameters** - many algorithms will have input parameters, and we shouldn't have to know a lot about the data to come up with good input parameters. The more we can isolate understanding about the data from the actual algorithm, the better. In other words, our algorithm should be useful even if we don't know much about the data.

4. **Interpretable** - we should be able to interpret the results.
5. **Constraints** - if we have predefined constraints (i.e: we know two points should be in the same cluster, or shouldn't belong together), it's sometimes helpful to have an algorithm that uses these constraints effectively

After discussing its implementation, we'll discuss the various benefits and drawbacks of agglomerative clustering, with regards to these criteria.

3.3 Agglomerative Clustering Implementation

The general idea behind agglomerative clustering is that we want to measure the similarities among data points and use those to group close points together. Once we start creating groups, we'll then merge groups that are close distance-wise together, and keep repeating this process until we have one remaining group. This resulting grouping creates a hierarchy, where some points are grouped together before other points, which is best viewed as a **dendrogram**.

We can visualize this in the following diagram, which shows data points, and the results of an agglomerative clustering algorithm.

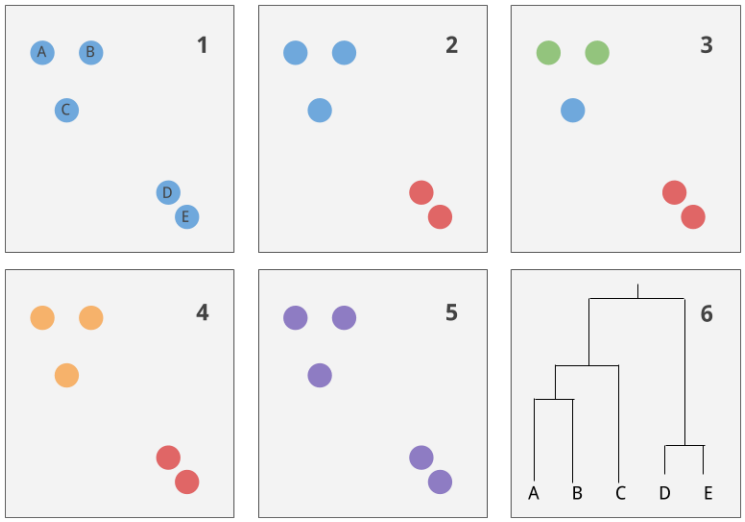

Figure 7: Agglomerative clustering on sample input, and resulting dendrogram

The first picture shows all the datapoints, and pictures 2 to 5 show various steps in the clustering algorithm. Step 2 groups the two red points together, step 3 groups the two green points together, step 4 groups the previous green group and the nearby blue point into a new orange group, and step 5 groups all the points together into one large group. This creates the dendrogram in picture 6.

3.3.1 Algorithm

Our general algorithm is based off our intuition and has four main steps:

1. Initialize each point to its own cluster
2. Find the most similar pair of clusters
3. Merge the similar pair of clusters into a *parent* cluster
4. Repeat steps 2 & 3 until we have 1 cluster.

3.3.2 Questions

Although agglomerative clustering is a powerful technique, there's still various factors we should consider when implementing it. For instance:

1. *How do we define similarity between clusters? How do we measure the distance between two clusters?*

We can measure the distance between two clusters in a variety of different ways, including looking at the average distance between points, the minimum distance between points in the clusters, the distance between the means of each cluster, and the maximum distance between points in the clusters. Each way we measure the distance between clusters can highly vary the results, as we'll see later.

2. *How many clusters do we chose?*

   When we create the dendrogram, we can decide how many clusters we want based off a distance threshold. Alternatively, we can just cut the dendrogram horizontally at some point to see how many clusters that creates.

3.4 Different measures of nearest clusters

There are three main models we can use to determine the distance between cluster points as we segment our dataset: single, complete, and average.

1. Single link:
   Distance is calculated with the formula:

$$d(C_i, C_j) = min_{x \in C_i, x' \in C_j} d(x, x') \tag{5}$$

   With single linkage, we cluster by utilizing the minimum distance between points in the two clusters.
   This is also known as a minimum spanning tree.
   We can stop clustering once we pass a threshold for the acceptable distance between clusters.
   This algorithm tends to produces long, skinny clusters (since it is very easy to link far away points to be in the same cluster as we only care about the point in that cluster with the minimum distance).

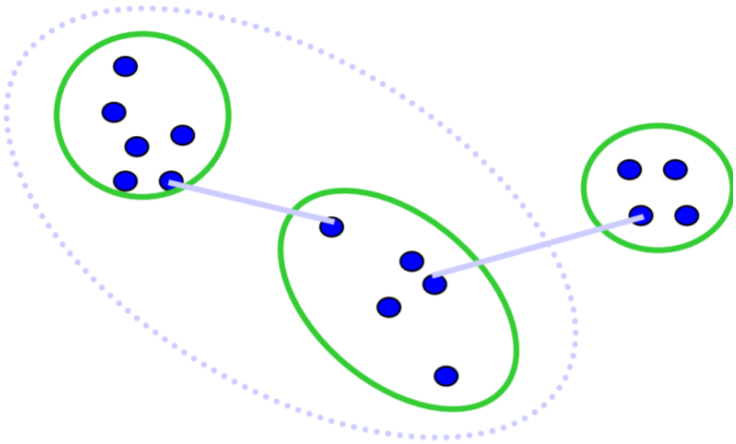

Figure 8: Image segmentation example using single link measurement of nearest clusters.
Source: lecture 12, slide 46

2. Complete link:
   Distance is calculated with the formula:

$$d(C_i, C_j) = max_{x \in C_i, x' \in C_j} d(x, x') \tag{6}$$

   With complete linkage, we cluster by utilizing the maximum distance between points in the two clusters.
   This algorithm tends to produces compact, tight clusters of roughly equal diameter (since it favors having all the points close together).

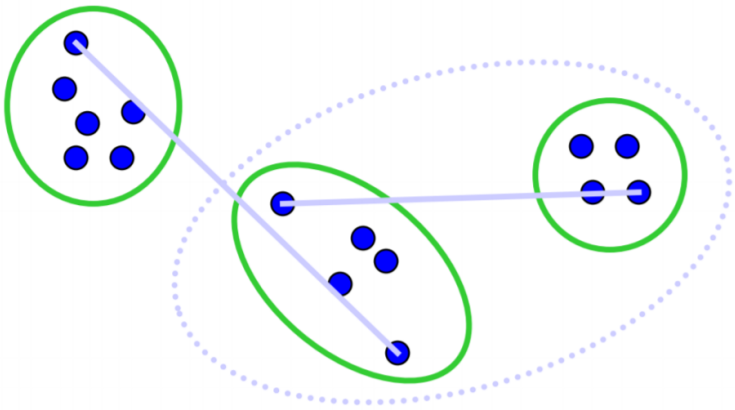

Figure 9: Image segmentation example using complete link measurement of nearest clusters.
Source: lecture 12, slide 47

3. Average link:
   Distance is calculated with the formula:

$$d(C_i, C_j) = \frac{\sum x \in C_i, x' \in C_j d(x, x')}{|C_i| \cdot |C_j|} \tag{7}$$

   With average linkage, we cluster by utilizing the average distance between points in the two clusters.
   This model is robust against noise, as the distance does not depend on single pair of points unlike single link and complete link, where points can be affected by artifacts in the data.

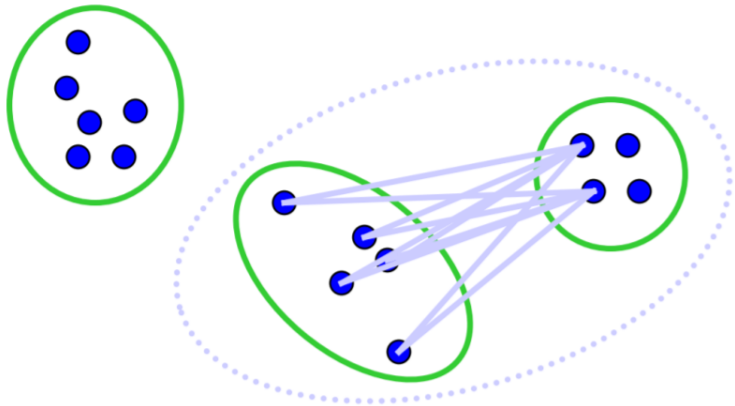

Figure 10: Image segmentation example using average link measurement of nearest clusters.
Source: lecture 12, slide 48

3.5 Agglomerative clustering conclusions

As with any algorithm, agglomerative clustering has its pros and cons when it comes to clustering.

In terms of its positive characteristics;

1. Simple to implement and apply
2. Cluster shape adapts to dataset
3. Results in a hierarchy of clusters
4. No need to specify number of clusters at initialization

In terms of its negative characteristics;

1. Can return imbalanced clusters

2. Threshold value for number of clusters must be specified

3. Does not scale well with a runtime of $O(n^3)$

4. Greedy merging can get stuck at local minima

4 K-Means Clustering

Another clustering algorithm is called k-means clustering. K-means identifies a fixed number of cluster "centers" as representatives of their clusters, and labels each point according to the center it is closest to. A major difference between k-means and agglomerative clustering is that k-means requires the input of a target number of clusters to run the algorithm.

4.1 Image Segmentation Example

At the top left of figure 11, we have an image with three distinct color regions, so segmenting the image using color intensity can be done by assigning each color intensity, shown on the top right, to a different cluster. In the bottom left image, however, the image is cluttered with noise. To segment the image, we can use k-means.

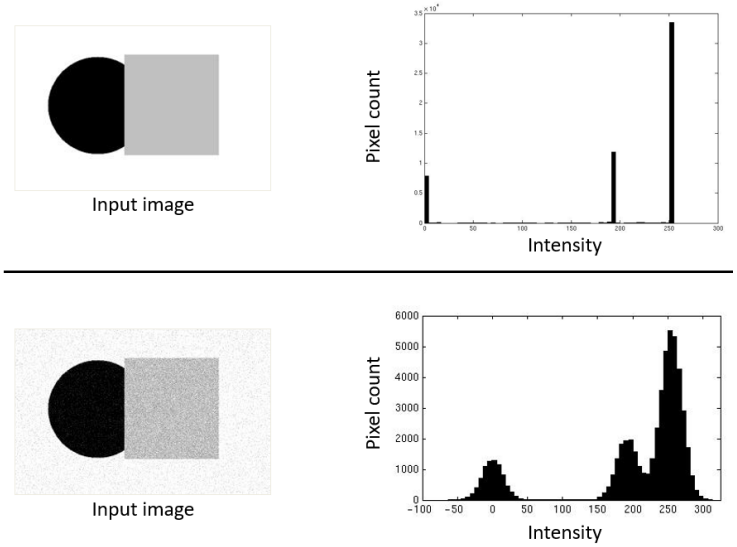

Figure 11: Image segmentation example using k-means. The picture on the top left has three distinct colors, but the bottom picture has Gaussian noise. Source: lecture 11, slide 8, slide credit: Kristen Grauman

Using k-means, the objective here is to identify three cluster centers as the representative intensities, and label each pixel according to its closest center. The best cluster centers are those that minimize Sum of Square Distance between all points and their nearest cluster center c_i:

$$SSD = \sum_{clusteri} \sum_{x \in clusteri} (x - c_i)^2 \tag{8}$$

When we are using k-means to summarize a dataset, the goal is to minimize the variance in the data points that are assigned to each cluster. We would like to preserve as much informa-

tion as possible given a certain number of clusters. This is demonstrated by the equation below.

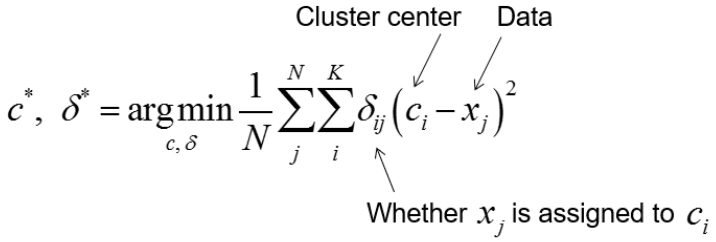

Figure 12: Clustering for summarization. Source: lecture 11, slide 11

4.2 Algorithm

Finding the cluster centers and group memberships of points can be thought of as a "chicken and egg" problem. If we knew the cluster centers, we could allocate points to groups by assigning each point to the closest center. On the other hand, if we knew the group memberships, we could find the centers by computing the mean of each group. Therefore, we alternate between the tasks.

In order to find the centers and group memberships, we start by initializing k cluster centers, usually by assigning them randomly. We then run through an iterative process that computes group memberships and cluster centers for a certain number of iterations or until the values of the cluster centers converge. The process is outlined below:

1. Initialize cluster centers $c_1, ..., c_K$. Usually, these centers are randomly chosen data points.
2. Assign each point in the dataset to the closest center. As in agglomerative clustering, we can use the Euclidean distance or the cosine distance measure to compute the distance to each center.
3. Update the cluster centers to be the mean of the points in the cluster's group.
4. Repeat Steps 2-3 until the value of the cluster centers stops changing or the algorithm has reached the maximum number of iterations.

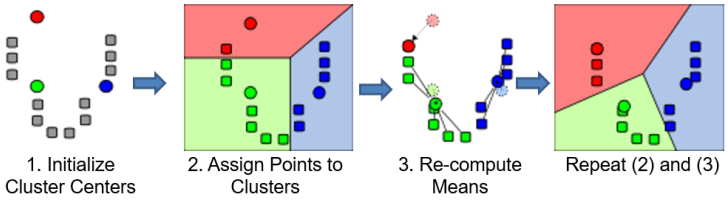

Figure 13: Visualization of k-means clustering. Source: lecture 11, slide 15

4.3 Output

Each time it is run, k-means converges to a local minimum solution. Additionally, since the centers are initialized randomly, each run of the algorithm may return a different result. Therefore, initializing multiple runs of k-means and then choosing the most representative clustering yields the best results. The best clustering can be measured by minimizing the sum of the square distances to the centers or the variance of each cluster. K-means works best with spherical data.

4.4 Segmentation as Clustering

As with the example in section 4.1, clustering can be used to segment an image. While color intensity alone can be effective in situations like figure 11, images like the one below (figure 14) may require us to define a feature space, in which we choose which features of pixels should affect the clustering.

Figure 14: Image of a panda. Source: lecture 11, slide 19

In addition to pixel intensity, examples of pixel groupings using an image feature space include RGB color similarities, texture similarities, and pixel positions. Clustering based on color similarities can be modeled using separate features for red, green, and blue. Texture can be measured by the similarities of pixels after applying specific filters. Position features include the coordinates of pixels within an image. Both intensity and position can be used together to group pixels based on similarity and proximity.

4.5 K-Means++

K-means method is appealing due to its speed and simplicity but not its accuracy. By augmentation with a variant on choosing the initial seeds for the k-means clustering problems, arbitrarily bad clusterings that are sometimes a result of k-means clustering may be avoided. The algorithm for choosing the initial seeds for k-means++ is outlined as following:

1. Randomly choose a starting center from the data points
2. Compute a distance $D(X)$, which is a distance between each data point x to the center that has been chosen. By using a weighted probability distribution, a new data point is chosen as a new center based on a probability that is proportional to $D(x)^2$
3. Repeat the previous step until k centers have been chosen and then proceed with the usual k-means clustering process as the initial seeds have been selected

It has been shown that k-means++ is Olog(K) competitive.

4.6 Evaluation of clusters

There are various measures that can be taken to evaluate the clustering results. For example, there is an $internal$ evaluation measure, which involves giving a single quality score the results. $External$ evaluation, on the other hand, compares the clustering results to an existing true classification. More qualitatively, we can evaluate the results of clustering based on its $generative$ measure: how well is the reconstruction of points from the clusters or is the center of the cluster a good representation of the data. Another evaluation method is a $discriminative$ method where we evaluate how well clusters correspond to the labels. We check if the clusters are able to separate things that should be separated. This measure can only be worked with supervised learning as there are no labels associated with unsupervised learning.

4.7 Pros & Cons

There are advantages and disadvantages associated with k-means clustering technique:

Pros

1. Simple and easy to implement
2. Fast for low-dimensional data
3. Good representation of the data (cluster centers minimize the conditional variance)

Cons

1. Doesn't identify outliers
2. Need to specify k value, which is unknown
3. Cannot handle non-globular data of different sizes and densities
4. Restricted to data which has the notion of a center
5. Converge to a local minimum of the objective function instead and is not guaranteed to converge to the global minimum of the objective function

In order to choose the number of clusters or the k value, the objective function can be plotted against different k values. The abrupt change in the objective function at a certain k value is suggestive of that specific number of clusters in the data. This technique is called "knee-finding" or "elbow-finding"

5 Mean-shift Clustering

Mean-shift clustering is yet another clustering algorithm. At its essence, mean-shift clustering is about finding the densest areas in our feature space. This algorithm has four main steps:

1. Initialize random seed, and window W
2. Calculate center of gravity (the "mean) of W: $\sum_{x \in W} xH(x)$
3. Shift the search window to the mean
4. Repeat step 2 until convergence

One way to mentally visualize this algorithm is to picture each data point as a marble. If each marble is attracted to areas of high density, all the marbles will eventually converge onto 1 or more centers.

We can also try to visualize the algorithm via this picture:

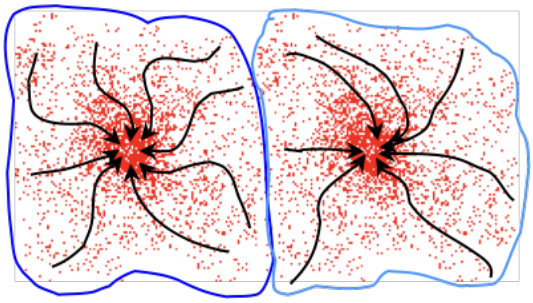

Figure 15: Results of mean-shift clustering.
Source: Y. Ukrainitz & B. Sarel

In this picture, we see the algorithm will generate 2 clusters. All the data points on the left converge onto one center and all the data points on the right converge onto a different center.

To learn more about mean-shift clustering, check out the next set of notes.

References

[1] David Forsyth and Jean Ponce. *Computer vision: a modern approach*. Upper Saddle River, NJ; London: Prentice Hall, 2011.