Lecture 16: Unsupervised Learning: Clustering

Data Science I

Elizabeth Sweeney [ems4003@med.cornell.edu]

Data Challenge #4

Data Challenge #4

- The assignment is due on Monday, 11/30 @ 6:00 pm on Canvas
- A rubric for grading of the data challenge is provide on the assignment
- When you submit the data challenge you will submit it as an html that contains a link to the GitHub repository with the code for your data challenge
- Make sure your repo is private. Make sure you add your TAs (Nick and Anjile) and I as collaborators to your repo. There are instructions for this in Lecture 2. You will lose points if you do not.
- Make sure the html file shows your code.

Lab #4

- Lab #4 will be due on Monday, 11/30 @ 6:00 pm on Canvas
- Lab #4 is graded for correctness and is worth 5% of you grade!
- When you submit the lab you will submit it as an html that contains a link to the GitHub repository with the code for your lab
- Make sure your repo is private. Make sure you add your TAs (Nick and Anjile) and I as collaborators to your repo. There are instructions for this in Lecture 2. You will lose points if you do not.
- New thing! There will be 5 participation points for the lab!

Thanksgiving Break

No class on Wednesday 11/25

Quiz

Required Reading

Chapter 10.3 from An Introduction to Statistical Learning (http://faculty.marshall.usc.edu/gareth-james/ISL/)

Chapter 5, In Depth: k-Means Clustering from the Python Data Science Handbook

Unsupervised Learning

Unsupervised vs Supervised Learning:

- In Data Science II you will learn about supervised learning methods such as regression and classification.
- In that setting you will observe both a set of features $X_1, X_2, \cdots, X_p$ for each object, as well as a response or outcome variable Y . The goal is then to predict Y using $X_1, X_2, \ldots, X_p$.
- In Data Science I, we focus on **unsupervised learning**, where we observe only the features $X_1, X_2, \ldots, X_p$. We are not interested in prediction, because we do not have an associated response variable Y .

Goals and Challenges of Unsupervised Learning

Goals:

- The goal is to discover interesting things about the measurements: is there an informative way to visualize the data? Can we discover subgroups among the variables or among the observations?

Challenges:

- Unsupervised learning is more subjective than supervised learning, as there is no simple goal for the analysis, such as prediction of a response.

Some more advantages

- But techniques for unsupervised learning are of growing importance in a number of fields:
  - subgroups of breast cancer patients grouped by their gene expression measurements,
  - regions of the brain (through fMRI) that define subtypes of mental health disorders.
  - movies grouped by the ratings assigned by movie viewers.
- It is often easier to obtain **unlabeled data** – from a lab instrument or a computer – than labeled data, which can require human intervention.
- For example it is difficult to automatically assess the overall sentiment of a movie review: is it favorable or not?

Clustering

- Clustering refers to a very broad set of methods for finding subgroups, or clusters, in a data set.
- We seek a partition of the data into distinct groups so that the observations within each group are quite similar to each other.
- To make this concrete, we must define what it means for two or more observations to be similar or different.
- Indeed, this is often a domain-specific consideration that must be made based on knowledge of the data being studied.

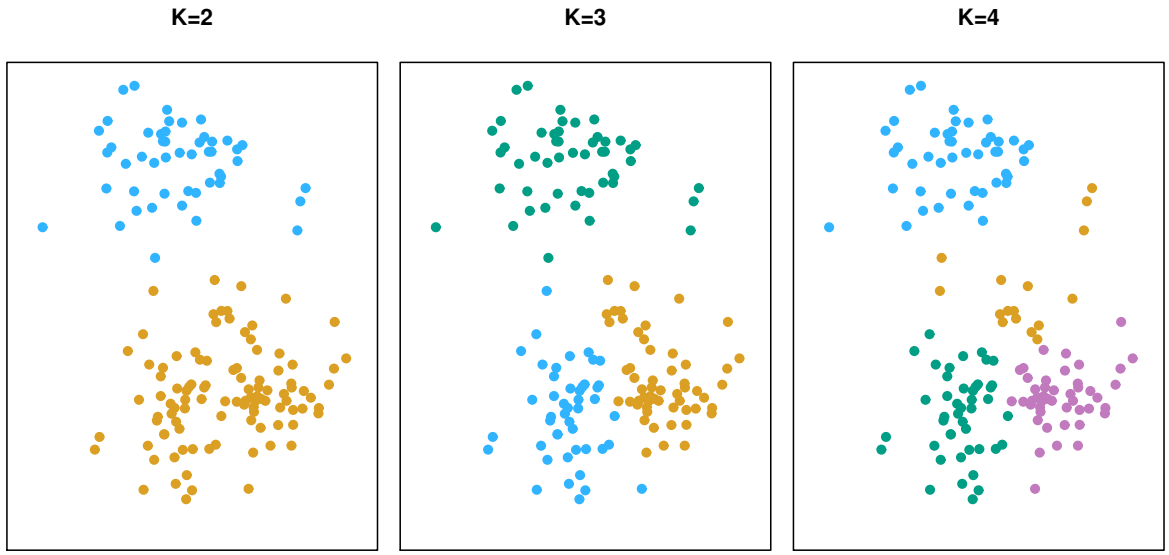
Example: Clustering for Market Segmentation

- Suppose we have access to a large number of measurements (e.g. median household income, occupation, distance from nearest urban area, and so forth) for a large number of people.
- Our goal is to perform **market segmentation** by identifying subgroups of people who might be more receptive to a particular form of advertising, or more likely to purchase a particular product.
- The task of performing market segmentation amounts to **clustering** the people in the data set into specific sub-groups.

Two clustering methods

- In **K-means clustering**, we seek to partition the observations into a pre-specified number of clusters.
- In **hierarchical clustering**, we do not know in advance how many clusters we want; in fact, we end up with a tree-like visual representation of the observations, called a **dendrogram**, that allows us to view at once the clusterings obtained for each possible number of clusters, from 1 to n.

K-means clustering



A simulated data set with 150 observations in 2-dimensional space. Panels show the results of applying K-means clustering with different values of K, the number of clusters. The color of each observation indicates the cluster to which it was assigned using the K-means clustering algorithm. Note that there is no ordering of the clusters, so the cluster coloring is arbitrary. These cluster labels were not used in clustering; instead, they are the outputs of the clustering procedure.

Details of K-means clustering

Let $C_1, \ldots, C_K$ denote sets containing the indices of the observations in each cluster. These sets satisfy two properties:

1. $C_1 \cup C_2 \cup \ldots \cup C_K = \{1, \ldots, n\}$. In other words, each observation belongs to at least one of the K clusters.
2. $C_k \cap C_{k'} = \emptyset$ for all $k \neq k'$. In other words, the clusters are non-overlapping: no observation belongs to more than one cluster.

For instance, if the i^{th} observation is in the k^{th} cluster, then $i \in C_k$.

Details of K-means clustering: continued

- The idea behind K-means clustering is that a *good* clustering is one for which the **within-cluster variation** is as small as possible.
- The within-cluster variation for cluster C_k is a measure $WCV(C_k)$ of the amount by which the observations within a cluster differ from each other.
- Hence we want to solve the problem

$$\underset{C_1,\ldots,C_k}{\text{minimize}} \left\{ \sum_{k=1}^{K} WCV(C_k) \right\}$$

- In words, this formula says that we want to partition the observations into K clusters such that the total within-cluster variation, summed over all K clusters, is as small as possible.

How to define within-cluster variation?

- Typically we use **Euclidean distance**

$$WCV(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

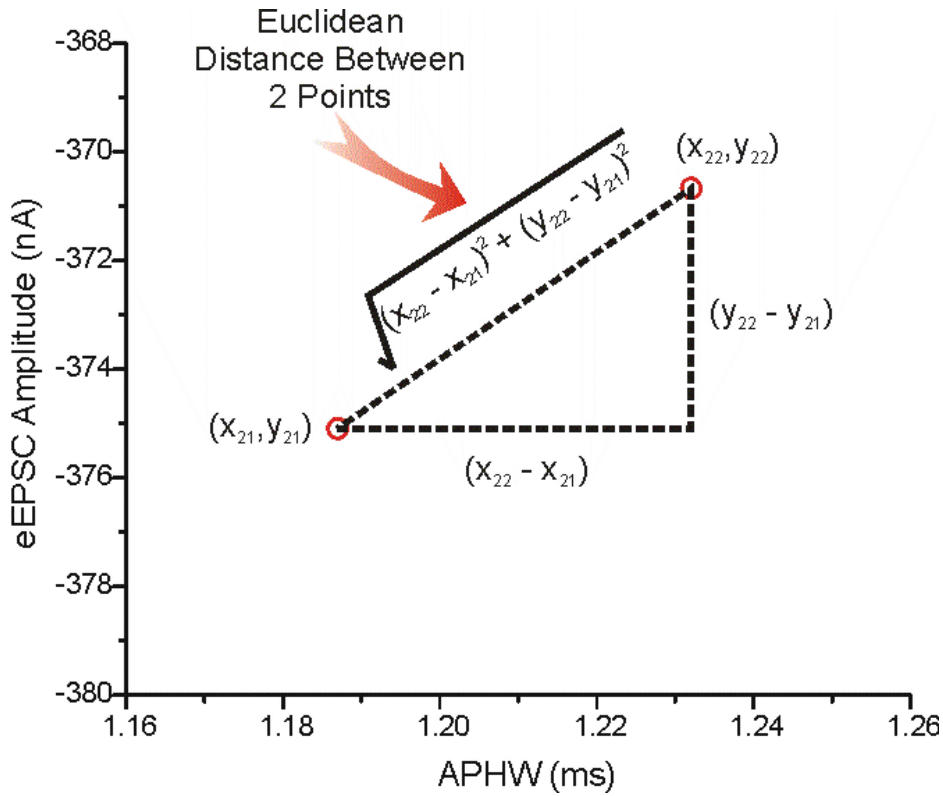
where $|C_k|$ denotes the number of observations in the k^{th} cluster.

- Combining (1) and (2) gives the optimization problem that defines K-means clustering,

$$\underset{C_1,\ldots,C_k}{\text{minimize}} \left\{ \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \right\}$$

Euclidean Distance

Euclidean distance between two points in the feature space. In a two-dimensional space this is the following:



The Euclidean distance in a p-dimensional feature space between two points a and b is $\sqrt{\sum_i^p (a_i - b_i)^2}$

K-Means Clustering Algorithm

1. Randomly assign a number, from 1 to K, to each of the observations. These serve as initial cluster assignments for the observations.
2. Iterate until the cluster assignments stop changing:
   - 2.1 For each of the K clusters, compute the cluster **centroid**. The k^{th} cluster centroid is the vector of the p feature means for the observations in the k^{th} cluster.
   - 2.2 Assign each observation to the cluster whose centroid is **closest** (where closest is defined using Euclidean distance).

Note: K needs to be prespecified for K-means algorithm. This is a drawback of the K-means method.

Properties of the Algorithm

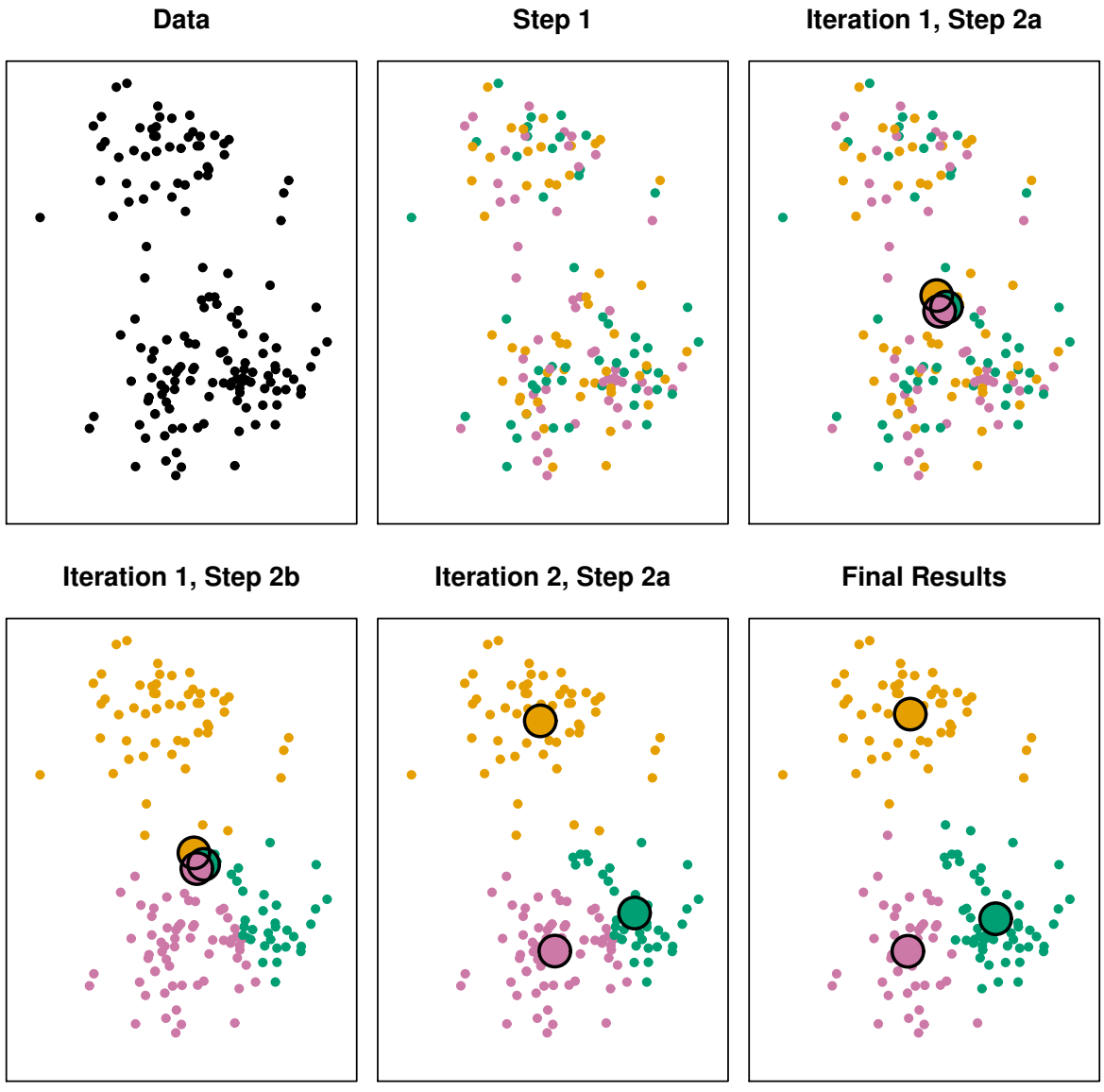
- This algorithm is guaranteed to decrease the value of the objective (4) at each step. **Why**? Note that

$$\frac{1}{|C_k|}\sum_{i,i'\in C_k}\sum_{j=1}^{p}(x_{ij}-x_{i'j})^2 = 2\sum_{i\in C_k}\sum_{j=1}^{p}(x_{ij}-\bar{x}_{kj})^2$$

where $\bar{x}_{kj} = \frac{1}{|C_k|}\sum_{i\in C_k} x_{ij}$ is the mean of the feature j in cluster C_k.

- There are two minimization criteria (see Figure 1)
  - given the centroid, you want to reclassify observations or change the C_i's such that the overall variation reduces
  - given a classification or a set of C_i's, update the centroid and we know that sum of squares are minimized at means.
  - therefore, the algorithm guarantees a reduction of variation at each step.
- however it is not guaranteed to give the global minimum. **Why not**?

Example


Data
Step 1
Iteration 1, Step 2a
Iteration 1, Step 2b
Iteration 2, Step 2a
Final Results

Details of Figure 1

The progress of the K-means algorithm with $K = 3$.

- **Top left**: The observations are shown.
- **Top center**: In Step 1 of the algorithm, each observation is randomly assigned to a cluster.
- **Top right**: In Step 2(a), the cluster centroids are computed. These are shown as large colored disks. Initially the centroids are almost completely overlapping because the initial cluster assignments were chosen at random.
- **Bottom left**: In Step 2(b), each observation is assigned to the nearest centroid.
- **Bottom center**: Step 2(a) is once again performed, leading to new cluster centroids.
- **Bottom right**: The results obtained after 10 iterations.

Example: different starting values

Details of Figure 2

- K-means clustering performed six times on the data from Figure 2 with $K = 3$, each time with a different random assignment of the observations in Step 1 of the K-means algorithm.
- Above each plot is the value of the objective (3).
- Three different local optima were obtained, one of which resulted in a smaller value of the objective and provides better separation between the clusters.
- Those labeled in red all achieved the same best solution, with an objective value of 235.8

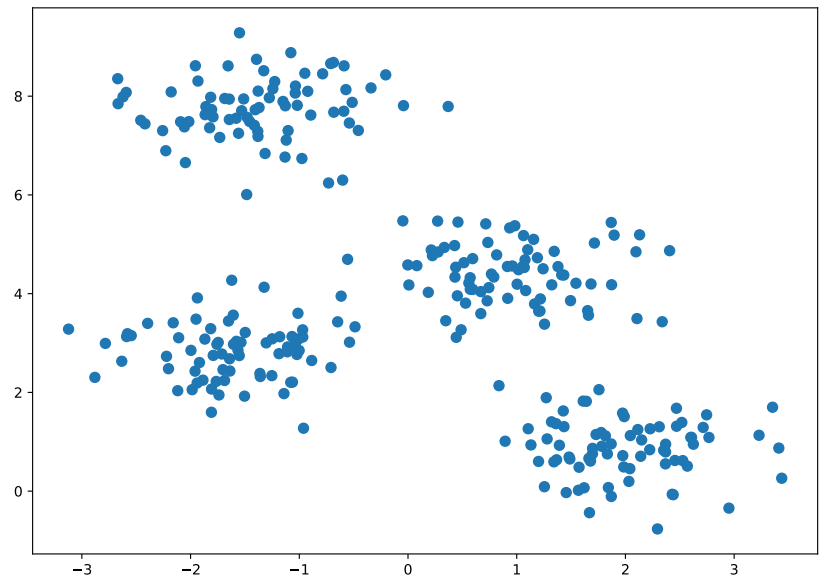
Example in Python

```
from sklearn.datasets.samples_generator import make_blobs
import matplotlib.pyplot as plt
import numpy as np

X, y_true = make_blobs(n_samples = 300,
                       centers = 4,
                       cluster_std = 0.60,
                       random_state = 0)

X
```

```
## array([[ 8.36856841e-01,  2.13635938e+00],
##        [-1.41365810e+00,  7.40962324e+00],
##        [ 1.15521298e+00,  5.09961887e+00],
##        [-1.01861632e+00,  7.81491465e+00],
##        [ 1.27135141e+00,  1.89254207e+00],
##        [ 3.43761754e+00,  2.61654166e-01],
##        [-1.80822253e+00,  1.59701749e+00],
##        [ 1.41372442e+00,  4.38117707e+00],
##        [-2.04932168e-01,  8.43209665e+00],
##        [-7.11099611e-01,  8.66043846e+00],
##        [-1.71237268e+00,  2.77780226e+00],
##        [-2.67000792e+00,  8.35389140e+00],
##        [ 1.24258802e+00,  4.50399192e+00],
##        [-2.22783649e+00,  6.89479938e+00],
##        [ 1.45513831e+00, -2.91989981e-02],
```

Example in Python

```
plt.scatter(X[:,0], X[:,1], s = 50)
```



Example in Python

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters = 4)
kmeans.fit(X)
```

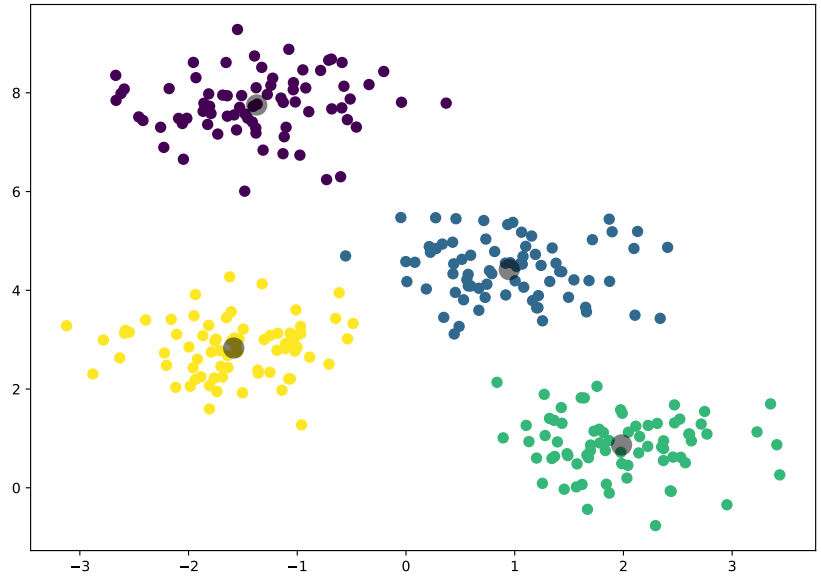
```
## KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
##     n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
##     random_state=None, tol=0.0001, verbose=0)
```

```
y_kmeans = kmeans.predict(X)

y_kmeans
```

```
## array([2, 0, 1, 0, 2, 2, 3, 1, 0, 0, 3, 0, 1, 0, 2, 1, 1, 2, 3, 3, 2, 2,
##        1, 3, 3, 1, 2, 1, 3, 1, 0, 0, 1, 0, 0, 0, 0, 0, 3, 2, 1, 3, 1, 1,
##        3, 3, 0, 3, 0, 2, 3, 2, 0, 2, 2, 3, 0, 3, 0, 2, 0, 1, 0, 3, 3, 3,
##        0, 2, 0, 3, 1, 3, 0, 3, 3, 0, 3, 1, 2, 0, 2, 1, 2, 2, 0, 1, 2, 1,
##        0, 0, 1, 2, 0, 3, 3, 1, 2, 2, 1, 3, 0, 2, 0, 2, 1, 2, 2, 1, 0, 1,
##        3, 3, 2, 0, 2, 1, 0, 2, 2, 1, 3, 2, 3, 2, 2, 2, 2, 3, 2, 3, 0, 3,
##        3, 2, 0, 3, 3, 0, 1, 0, 0, 3, 1, 3, 1, 3, 0, 1, 0, 0, 0, 1, 0, 1,
##        2, 3, 0, 3, 2, 1, 0, 1, 1, 2, 1, 3, 3, 1, 2, 1, 1, 0, 2, 1, 3, 0,
##        2, 2, 1, 3, 2, 1, 3, 3, 1, 1, 1, 1, 2, 0, 1, 3, 1, 1, 3, 3, 3, 1,
##        3, 0, 1, 3, 2, 3, 1, 0, 3, 0, 1, 0, 1, 3, 1, 1, 0, 3, 3, 2, 2, 1,
##        0, 2, 2, 3, 2, 3, 1, 0, 0, 1, 1, 0, 1, 2, 3, 1, 2, 3, 0, 3, 2, 1,
##        2, 0, 0, 0, 0, 3, 3, 0, 1, 3, 2, 1, 3, 3, 3, 2, 2, 0, 1, 1, 3, 2,
```

Example in Python

```
plt.scatter(X[:,0], X[:,1], c = y_kmeans, s = 50, cmap = 'viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:,0], centers[:,1], c = 'black', s = 200, alpha = 0.5)
```



Example in Python

What if we thought there were 5 clusters?

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters = 5)
kmeans.fit(X)
```

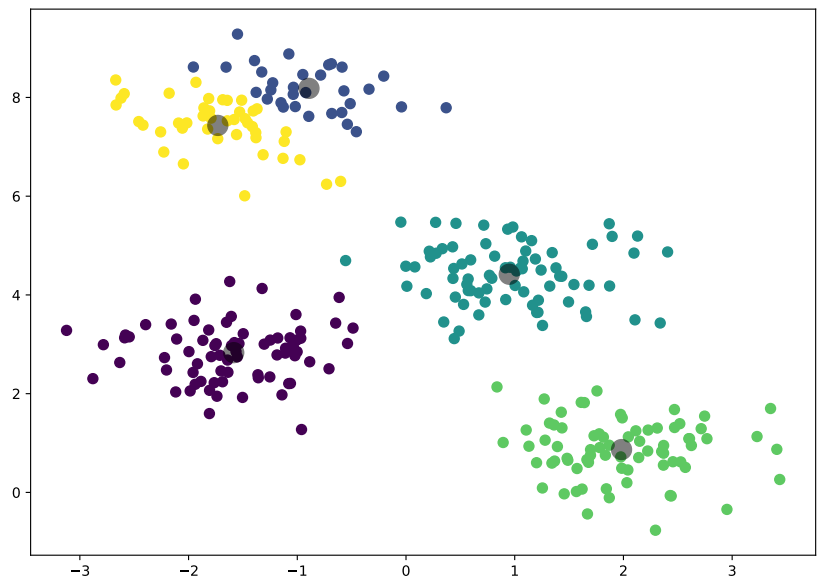
```
## KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
##     n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
##     random_state=None, tol=0.0001, verbose=0)
```

```
y_kmeans = kmeans.predict(X)
```

Example in Python

What if we thought there were 5 clusters?

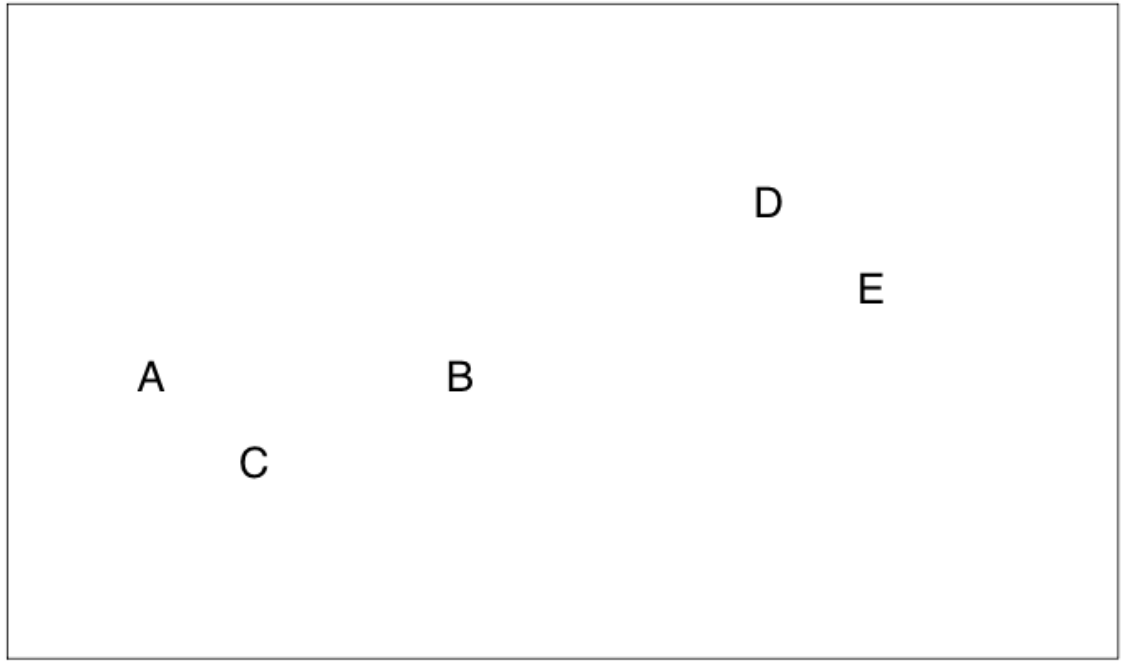
```
plt.scatter(X[:,0], X[:,1], c = y_kmeans, s = 50, cmap = 'viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:,0], centers[:,1], c = 'black', s = 200, alpha = 0.5)
```



Hierarchical Clustering

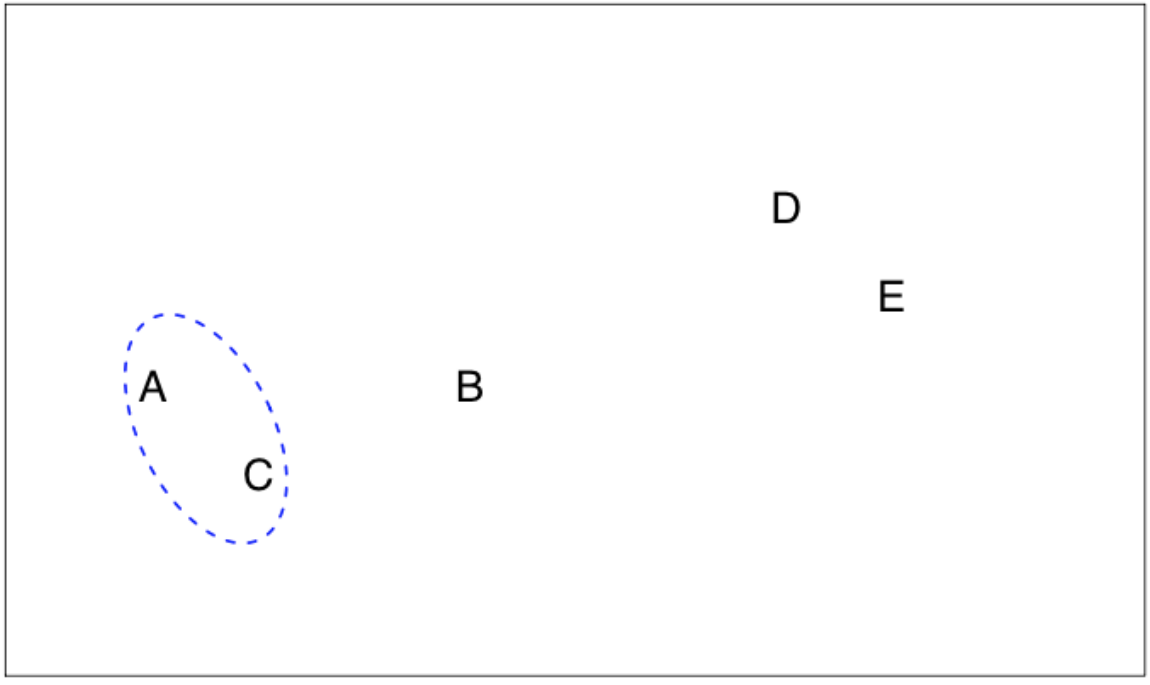
- K-means clustering requires us to pre-specify the number of clusters K. This can be a disadvantage (later we discuss strategies for choosing K)
- **Hierarchical clustering** is an alternative approach which does not require that we commit to a particular choice of K.
- In this section, we describe **bottom-up** or **agglomerative clustering**. This is the most common type of hierarchical clustering, and refers to the fact that a dendrogram is built starting from the leaves and combining clusters up to the trunk.

Hierarchical Clustering: the idea



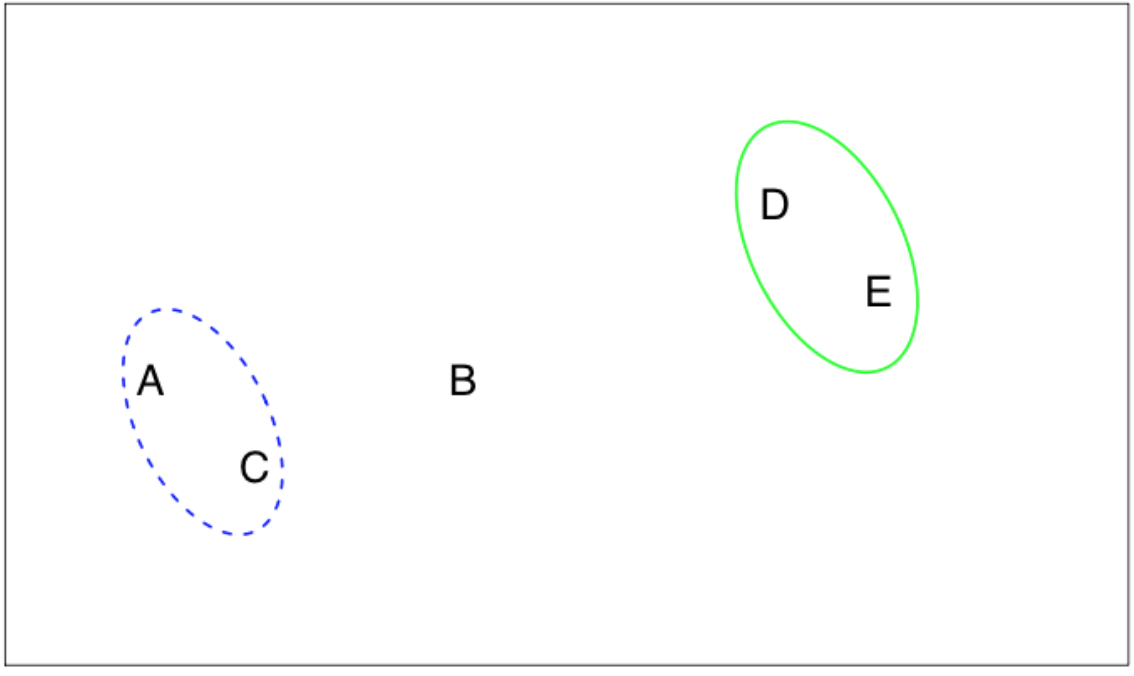
Builds a hierarchy in a **bottom-up** fashion.

Hierarchical Clustering: the idea



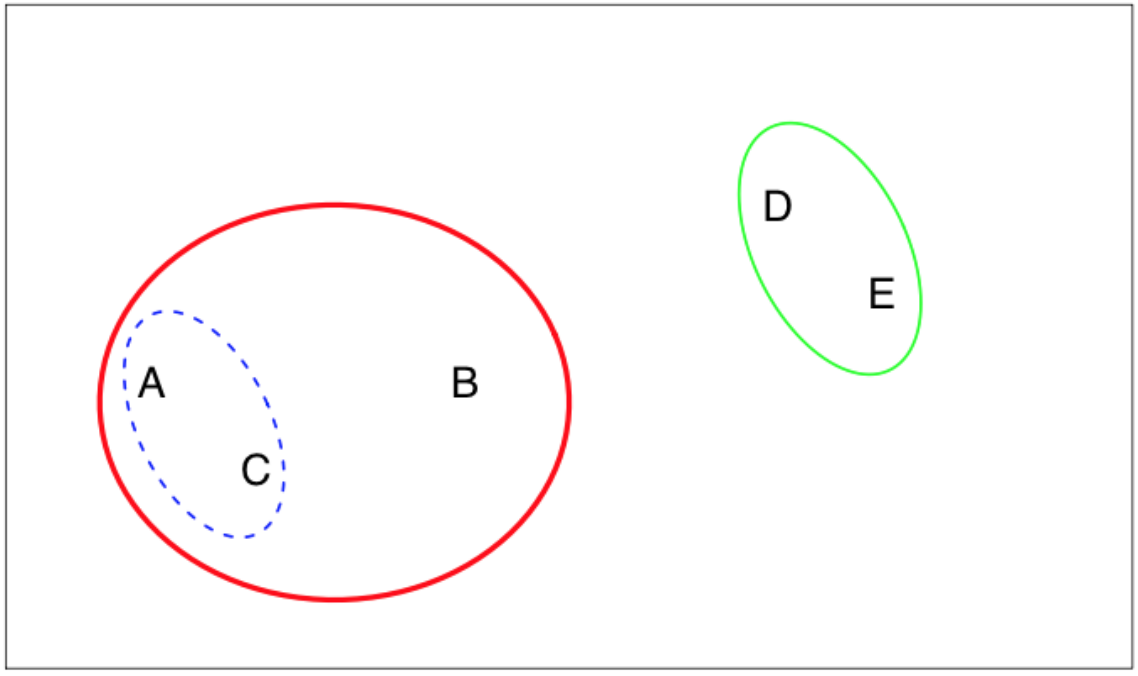
Builds a hierarchy in a **bottom-up** fashion.

Hierarchical Clustering: the idea



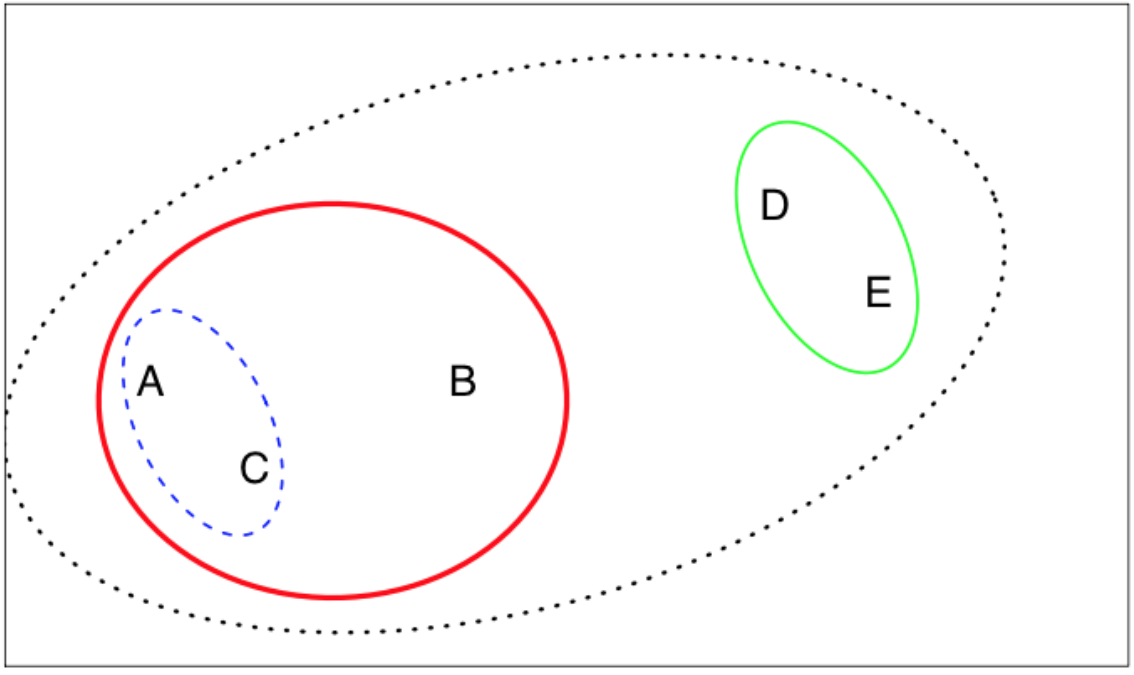
Builds a hierarchy in a **bottom-up** fashion.

Hierarchical Clustering: the idea



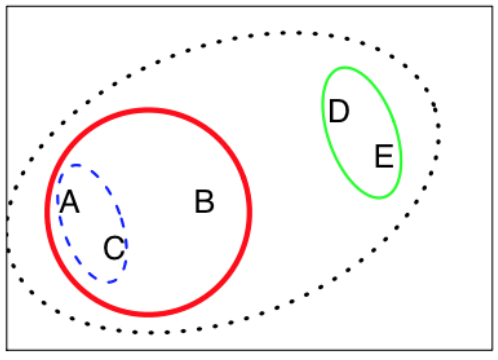
Builds a hierarchy in a **bottom-up** fashion.

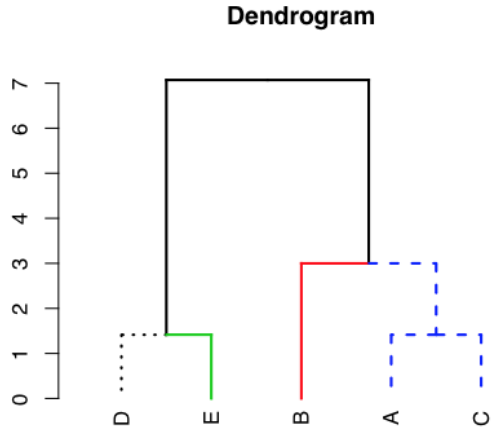
Hierarchical Clustering: the idea



Builds a hierarchy in a **bottom-up** fashion.

Hierarchical Clustering Algorithm





- Start with each point in its own cluster.
- Identify the closest two clusters and merge them.
- Repeat.
- Ends when all points are in a single cluster.

How Similar are Two Clusters?

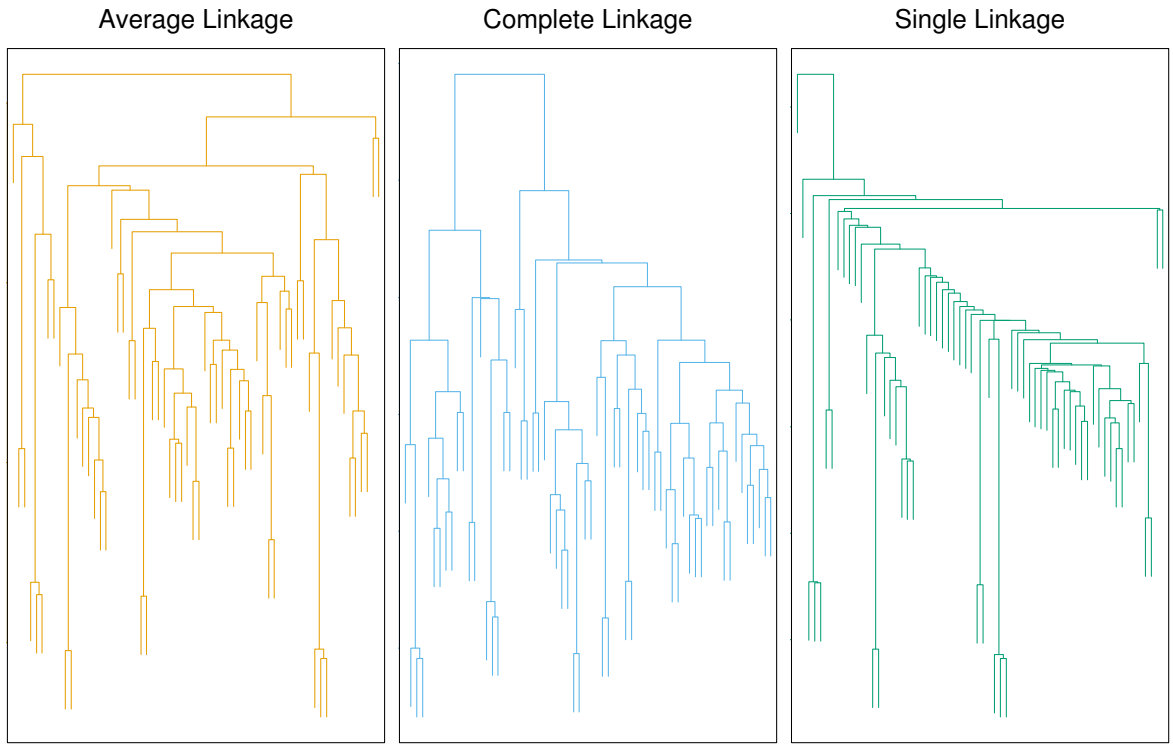
- We first must define a 'disimilarity' metric between points. We will use Euclidean distance, but other distances may be used.
- As we start to merge points together we will need a notion of how dissimilar two clusters are.
- This brings us to the notion of 'linkage', which defines the dissimilarity between two groups of observations
- This linkage is what is plotted along the y-axis of the dendogram.

Types of Linkage

The table below shows the most common types of linkage:

Liknage	**Description**
Complete	Maximal inter-cluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the **largest** of these dissimilarities.
Single	Minimal inter-cluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the **smallest** of these dissimilarities.
Average	Mean inter-cluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the **average** of these dissimilarities.
Centroid	Dissimilarity between the centroid for cluster A (a mean vector of length p) and the centroid for cluster B. Centroid linkage can result in undesirable **inversions** (whereby two clusters are fused at a height *below* either of the individual clusters in the dendrogram.)

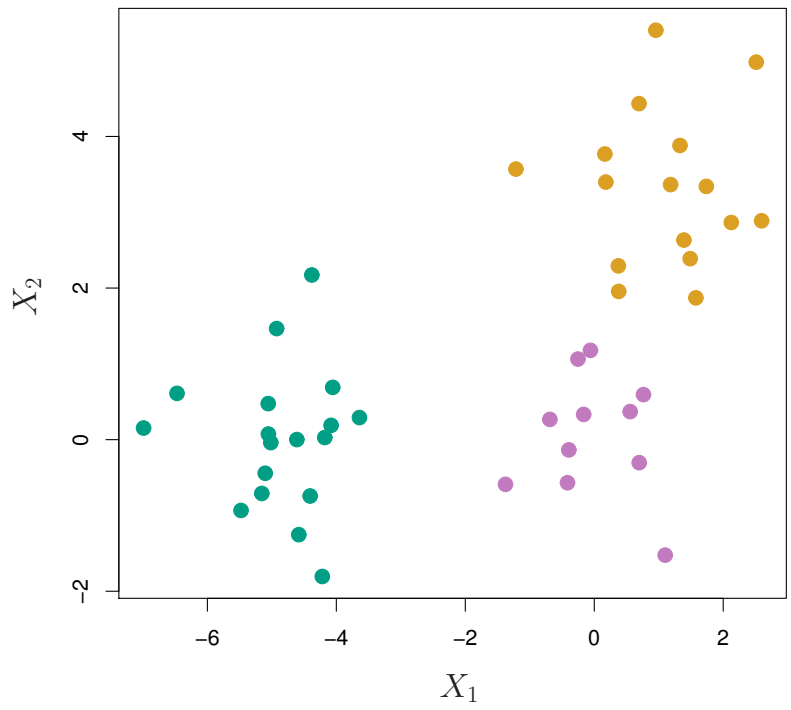
Types of Linkage



Average, complete, and single linkage applied to an example data set.
Average and complete linkage tend to yield more balanced clusters.

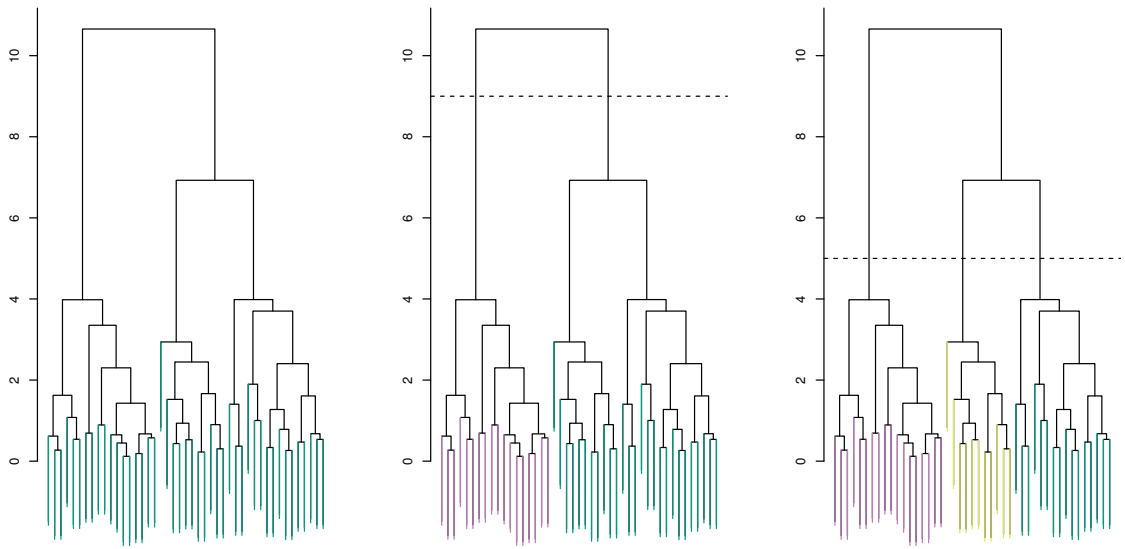
Picking the Number of Clusters

Forty-five observations generated in two-dimensional space. In reality there are three distinct classes, shown in separate colors. However, we will treat these class labels as unknown and will seek to cluster the observations in order to discover the classes from the data.

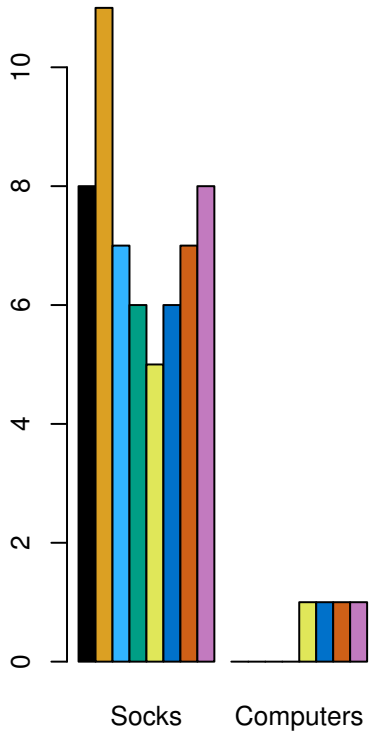


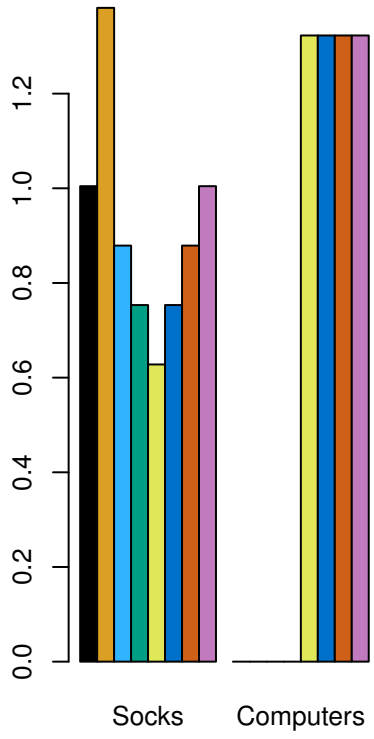
Picking the Number of Clusters

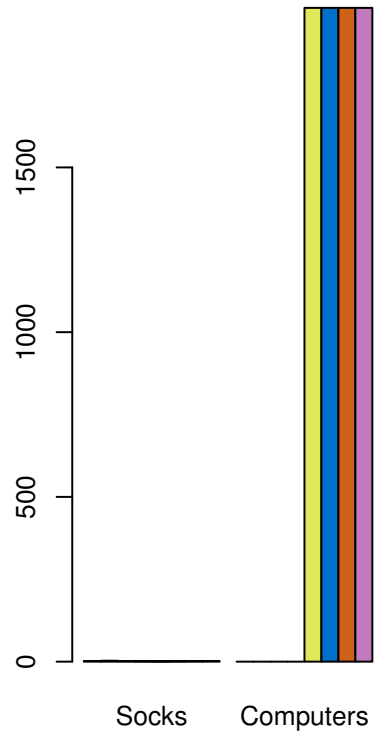
Left: dendrogram obtained from hierarchically clustering the data from Figure 10.8 with complete linkage and Euclidean distance. Center: the den- drogram from the left-hand panel, cut at a height of nine (indicated by the dashed line). This cut results in two distinct clusters, shown in different colors. Right: the dendrogram from the left-hand panel, now cut at a height of five. This cut results in three distinct clusters, shown in different colors.



A Note on Scaling Variables: Sale of socks and computer by an online retailers







Details of Figure 7 (Scaling of variables matter!)

- **Left**: Sale of socks pairs, and computers, purchased by eight online shoppers (different color) is displayed. If inter-observation dissimilarities are computed using Euclidean distance on the raw variables, then the number of socks purchased by an individual will drive the dissimilarities obtained, and the number of computers purchased will have little effect. This might be undesirable, since
  - computers are more expensive than socks and so the online retailer may be more interested in encouraging shoppers to buy computers than socks, and
  - a large difference in the number of socks purchased by two shoppers may be less informative about the shoppers' overall shopping preferences than a small difference in the number of computers purchased.
- **Center**: the same data is shown, after scaling each variable by its standard deviation. Now the number of computers purchased will have a much greater effect on the inter-observation dissimilarities obtained.
- **Right**: the same data are displayed, but now the y-axis represents the number of dollars spent by each online shopper on socks and on computers. Since computers are much more expensive than socks, now computer purchase history will drive the inter-observation dissimilarities obtained.

Scaling in Python

```
np.mean(X, axis = 0)
```

```
## array([-0.00632763,  3.96782089])
```

```
np.std(X, axis = 0)
```

```
## array([1.63327104, 2.58963659])
```

Scaling in Python

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
```

```
## StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
standardized_X = scaler.transform(X)

np.mean(standardized_X, axis = 0)
```

```
## array([4.14483263e-17, 1.18701345e-15])
```

```
np.std(standardized_X, axis = 0)
```

```
## array([1., 1.])
```

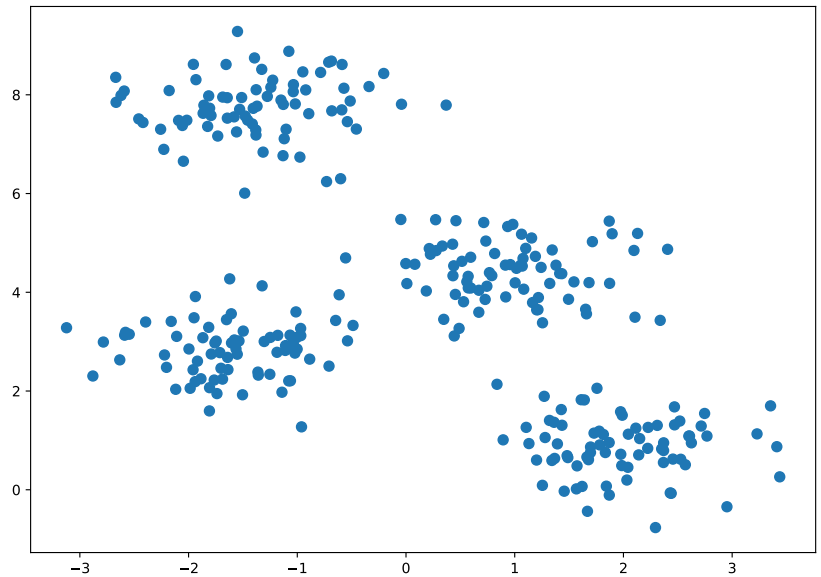
Practical issues

- Should the observations or features first be standardized in some way? For instance, maybe the variables should be centered to have mean zero and scaled to have standard deviation one.
- In the case of hierarchical clustering,
  - What dissimilarity measure should be used?
  - What type of linkage should be used?
- How many clusters to choose? (in both K-means or hierarchical clustering). Difficult problem. No agreed-upon method.

Example in Python

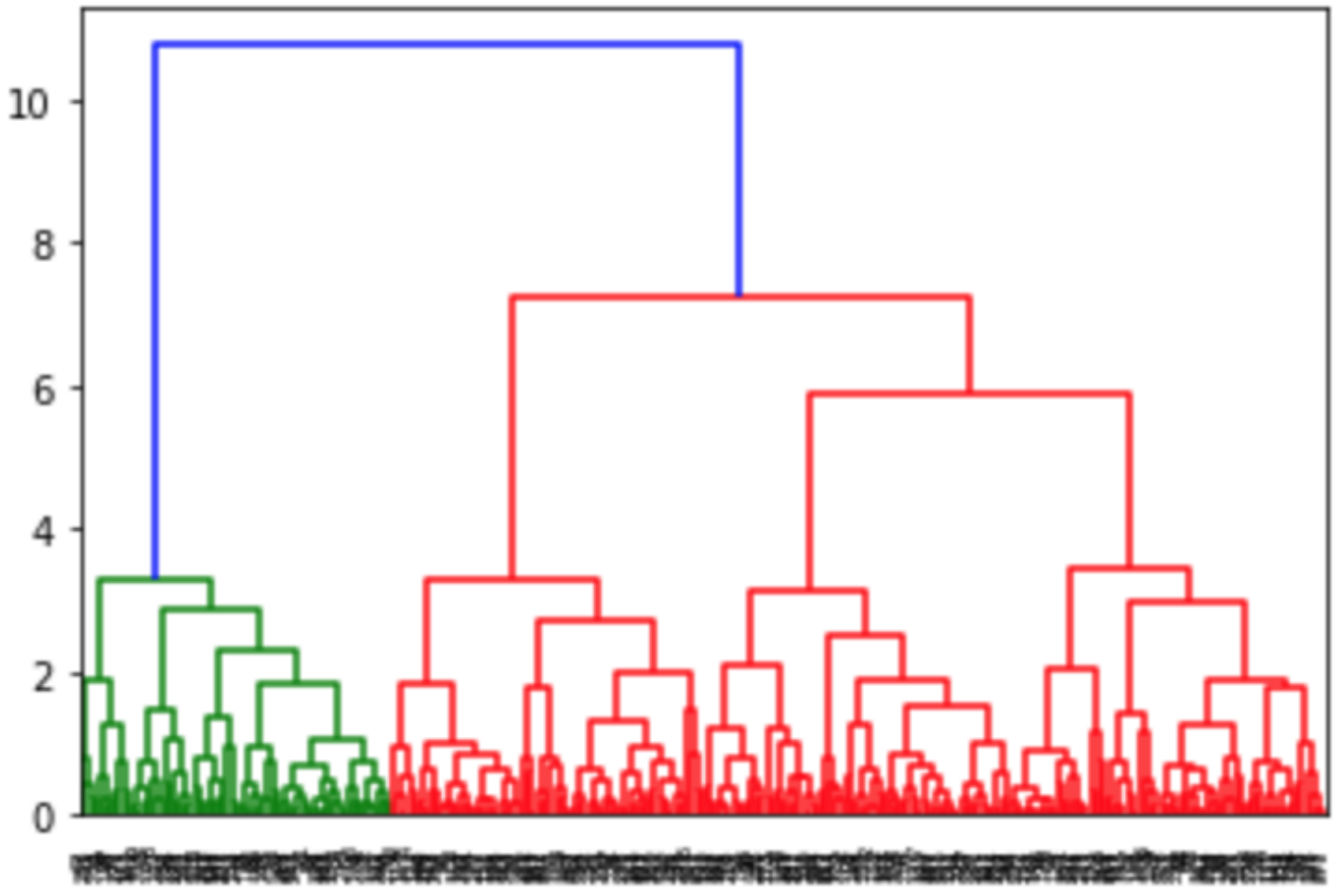
Let's return to our example from before

```
plt.scatter(X[:,0], X[:,1], s = 50)
```



Example in Python

```
import scipy.cluster.hierarchy as shc
dend = shc.dendrogram(shc.linkage(X, method='complete'))
```



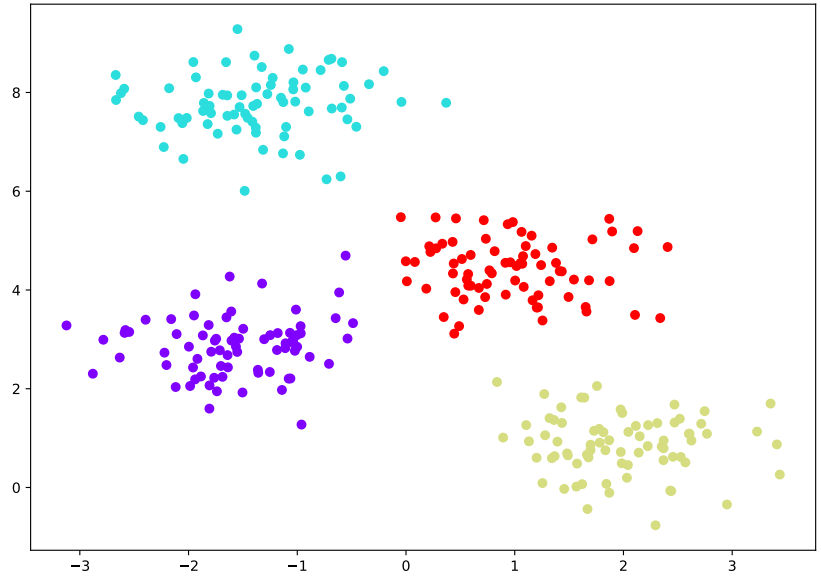
Example in Python

```
from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n_clusters=4,
                                  affinity='euclidean',
                                  linkage='complete')
cluster.fit_predict(X)
```

```
## array([2, 1, 3, 1, 2, 2, 0, 3, 1, 1, 0, 1, 3, 1, 2, 3, 3, 2, 0, 0, 2, 2,
##        3, 0, 0, 0, 2, 3, 0, 3, 1, 1, 3, 1, 1, 1, 1, 1, 0, 2, 3, 0, 3, 3,
##        0, 0, 1, 0, 1, 2, 0, 2, 1, 2, 2, 0, 1, 0, 1, 2, 1, 3, 1, 0, 0, 0,
##        1, 2, 1, 0, 3, 0, 1, 0, 0, 1, 0, 3, 2, 1, 2, 3, 2, 2, 1, 3, 2, 3,
##        1, 1, 3, 2, 1, 0, 0, 3, 2, 2, 3, 0, 1, 2, 1, 2, 3, 2, 2, 3, 1, 3,
##        0, 0, 2, 1, 2, 3, 1, 2, 2, 3, 0, 2, 0, 2, 2, 2, 2, 0, 2, 0, 1, 0,
##        0, 2, 1, 0, 0, 1, 3, 1, 1, 0, 3, 0, 3, 0, 1, 3, 1, 1, 1, 3, 1, 3,
##        2, 0, 1, 0, 2, 3, 1, 3, 3, 2, 3, 0, 0, 3, 2, 3, 3, 1, 2, 3, 0, 1,
##        2, 2, 3, 0, 2, 3, 0, 0, 3, 3, 3, 3, 2, 1, 3, 0, 3, 3, 0, 0, 0, 3,
##        0, 1, 3, 0, 2, 0, 3, 1, 0, 1, 3, 1, 3, 0, 3, 3, 1, 0, 0, 2, 2, 3,
##        1, 2, 2, 0, 2, 0, 3, 1, 1, 3, 3, 1, 3, 2, 0, 3, 2, 0, 1, 0, 2, 3,
##        2, 1, 1, 1, 1, 0, 0, 1, 3, 0, 2, 3, 0, 0, 0, 2, 2, 1, 3, 3, 0, 2,
##        1, 0, 3, 1, 3, 2, 2, 0, 0, 3, 2, 2, 2, 3, 1, 1, 2, 2, 3, 2, 2, 2,
##        1, 0, 1, 3, 2, 2, 1, 1, 1, 2, 2, 3, 1, 0])
```

Example in Python

```
plt.clf()
plt.scatter(X[:,0], X[:,1], c=cluster.labels_, cmap='rainbow')
```



In Class Exercise

Use k-means clustering and hierarchical clustering to cluster the following data. When do you get the best results?

```
from sklearn.datasets import make_moons
X, y = make_moons(200, noise = .05, random_state = 0)
plt.scatter(X[:,0], X[:,1], s = 50)
```