Unsupervised Learning

Elsa Ferreira Gomes

2024/2025

- https://www.youtube.com/watch?v=ded_NQqOe7I
- https://www.youtube.com/watch?v=yktzn-Mr2Nw
- https://hastie.su.domains/ISLP/ISLP_website.pdf - Chp12
- This notebook contains an excerpt from the Python Data Science Handbook by Jake VanderPlas

Unsupervised Learning

The Goals of Unsupervised Learning

- 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?

PCA Summary

- Given any high-dimensional dataset it is usefull to start with PCA
  - to visualize the relationship between points,
  - to understand the main variance in the data
  - to understand the intrinsic dimensionality (by plotting the explained variance ratio).
- PCA's main weakness is that it tends to be highly affected by outliers in the data.

Clustering

Clustering algorithms seek to learn, from the properties of the data, an optimal division or discrete labeling of groups of points.

PCA vs Clustering

- PCA looks for a low-dimensional representation of the observations
  - that explains a good fraction of the variance.
- Clustering looks for homogeneous subgroups among the observations.

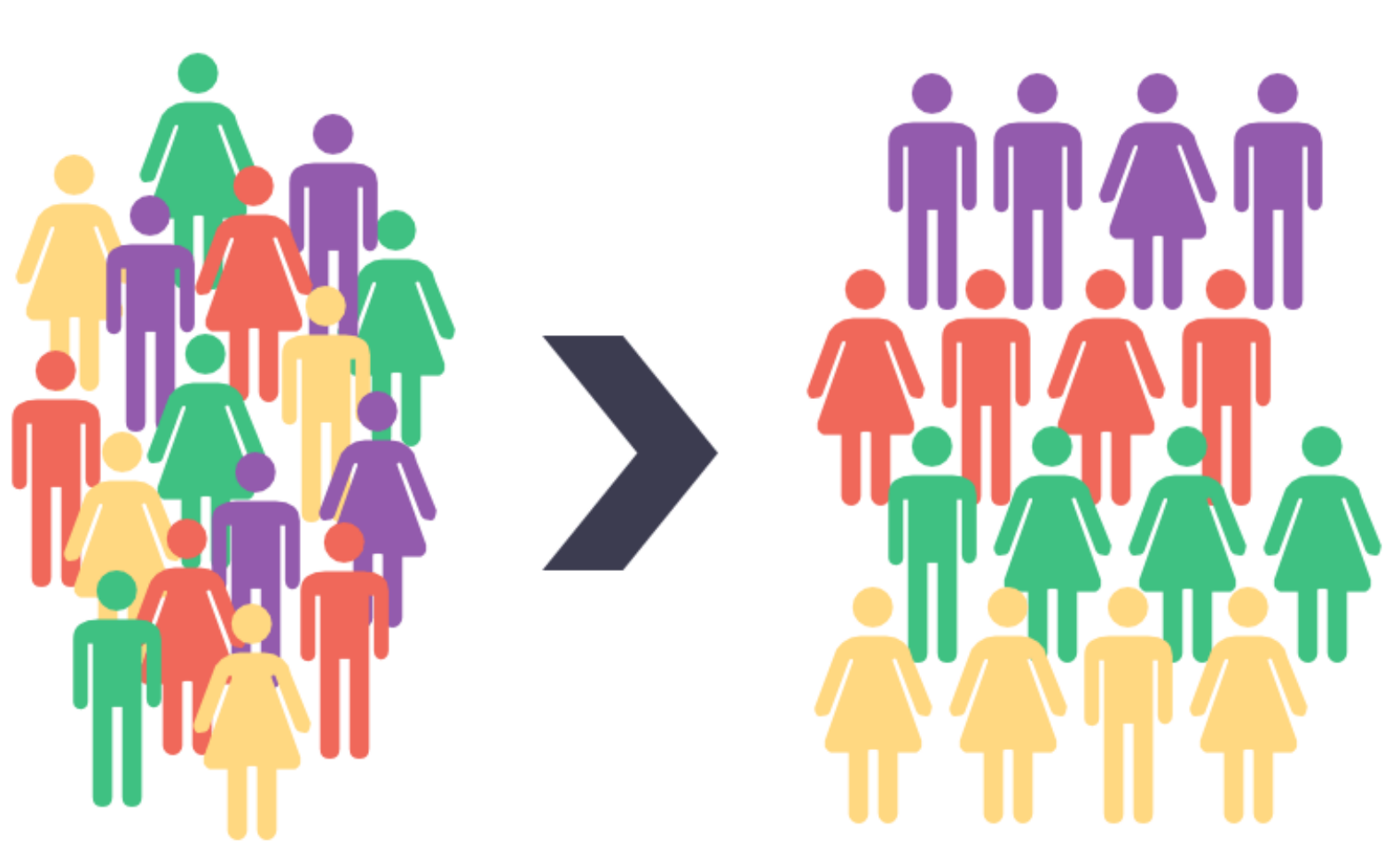
Cluster analysis

No labels available

- **Learn classes** without a supervisor
- Examples have **no class labels**

Some common applications

- Market segmentation
- Social network analysis
- Divide patients in homogeneous groups
- Organize web results by content



Clustering methods

- K-means clustering
  - we seek to partition the observations into a **pre-specified number of clusters**
- Hierarchical clustering
  - we do **not know in advance how many clusters we want**
    - 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.

Cluster analysis

Partition a set of objects into **groups**

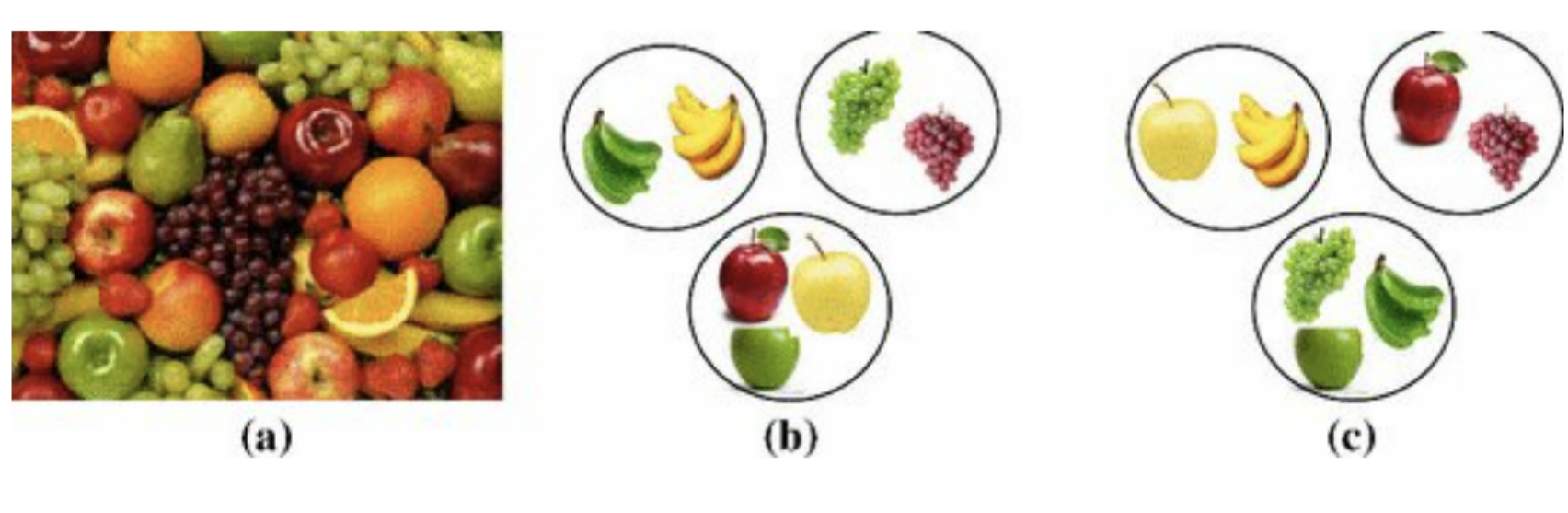
- Each group is a **cluster**
- Objects in the same cluster are **similar**
- Objects in different clusters are **dissimilar**

In the **real world** differences are not always easy to find

- Maximize intra-similarity
- Minimize inter-similarity

Cluster analysis

- **Given**
  - a set of objects
  - a number k of desired groups
- **Obtain**
  - a mapping of each object into each group



K-means Clustering

The k-means algorithm searches for a predetermined number of clusters within an unlabeled multidimensional dataset,

- using a simple conception of what the optimal clustering looks like:
  - The cluster center is the arithmetic mean of all the points belonging to the cluster.
  - Each point is closer to its own cluster center than to other cluster centers.

The k-Means algorithm (sklearn.cluster.KMeans)

The K-means algorithm

- clusters data by trying to separate samples:
  - in *n* groups of equal variance,
  - minimizing a criterion known as the *inertia* or within-cluster sum-of-squares.
- divides a set of N samples X into K disjoint clusters C, each described by the mean μ_j of the samples in the cluster.
  - The means are commonly called the cluster "centroids"; note that they are not, in general, points from X, although they live in the same space.

(https://scikit-learn.org/stable/modules/clustering.html#k-means)

k-Means clustering - Algorithm

- **Input**
  - k: the number of clusters,
  - D: a data set containing n objects.
- **Output**
  - A set of k clusters.
- **Method**
  1. **arbitrarily** choose k objects from D as the initial cluster centers (centroids)
  2. **repeat**
     - **(re)assign** each object to the cluster to which the object is the most similar the cluster center
     - **update** the cluster means, that is, calculate the mean value of the objects for each cluster (this result is set as the new centroid)
  3. **until** no change

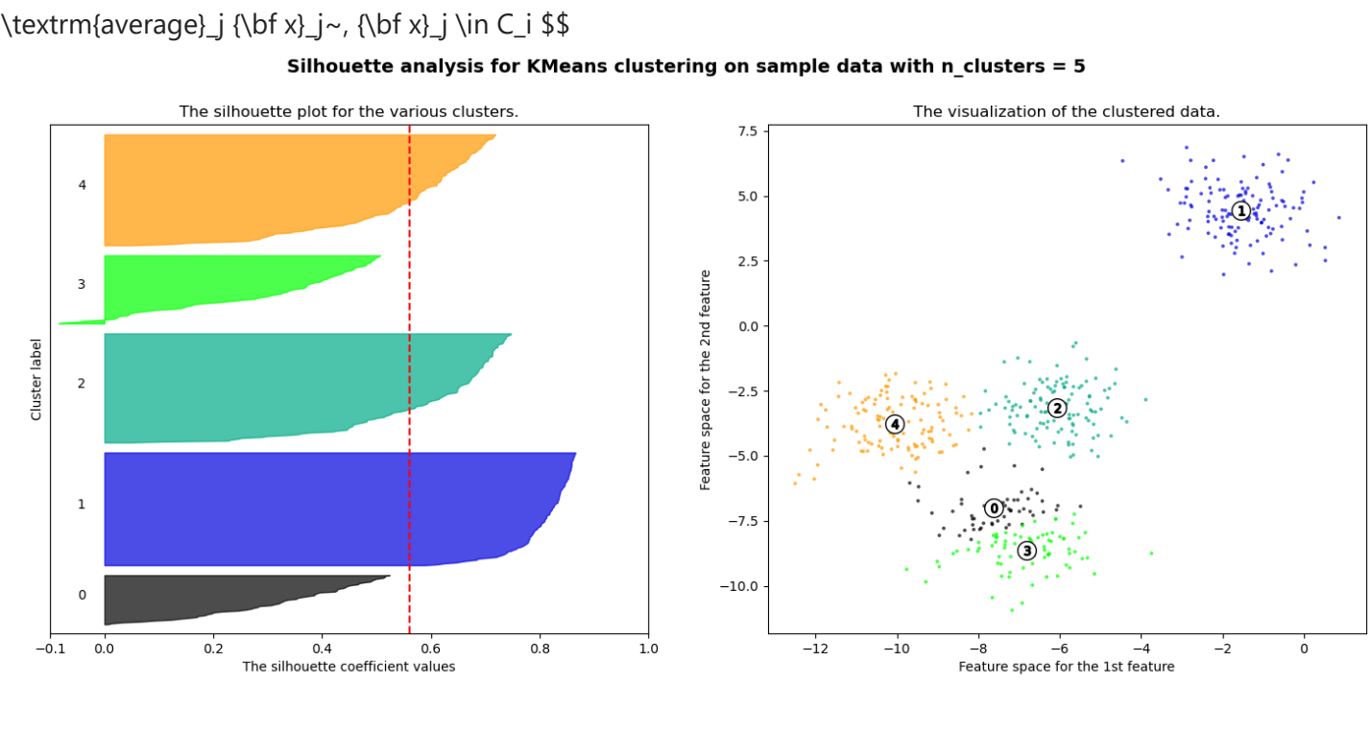
Hint: See sklearn.cluster.KMeans (https://scikit-learn.org/stable/modules/clustering.html#k-means)

k-means algorithm

Result of k-means

- k **disjoint** clusters $C_1, C_2, \ldots, C_k$
- $\bigcup_{i=1}^{k}{C_i}=D$, every point is in one cluster
- Each cluster C_i is characterized by a **centroid** ${\bf c}_i$
- A centroid is a vector but typically **not a true point**

$$ {\bf c}_i = \textrm{average}_j {\bf x}_j~, {\bf x}_j \in C_i $$



k-means clustering

Quality of a cluster

- we want to minimize **within-cluster variation**
- a measure of **error** (an objective function)
- the sum of the squared distances to the **centroid**

$$ E = \sum_i \sum_{{\bf x} \in C_i} dist({\bf c}_i,{\bf x})^2 $$

The clustering problem

The complexity

- In general it is a **NP-hard problem** (https://mathworld.wolfram.com/NP-HardProblem.html)
- k-means is a **greedy approach**
- **Complexity of k-means** is $O(nkt)$
  - $t=iterations$
  - usually dominated by n (**in practice** $O(n)$)
  - very **efficient**

Convergence

- k-means **may not converge** to a global optimum (for a given k)
- results **depend** on the initial seed centers

Options in k-means

Initialization

- Random
- Heuristic
- User's choice

Calculating centroids

- Means
- Modes (for categorical values), a.k.a. k-modes
- Sample for scalability

Outliers

- Means can be affected by outliers
- **k-Medoids** is an alternative that uses **median**
  - and **absolute error** in the objective function

Evaluating the result of clustering

How can the results of clustering be **evaluated**?

- Is there a **cluster structure** in the data?
- Is the **number of clusters** adequate?
- **How good** are the clusters?

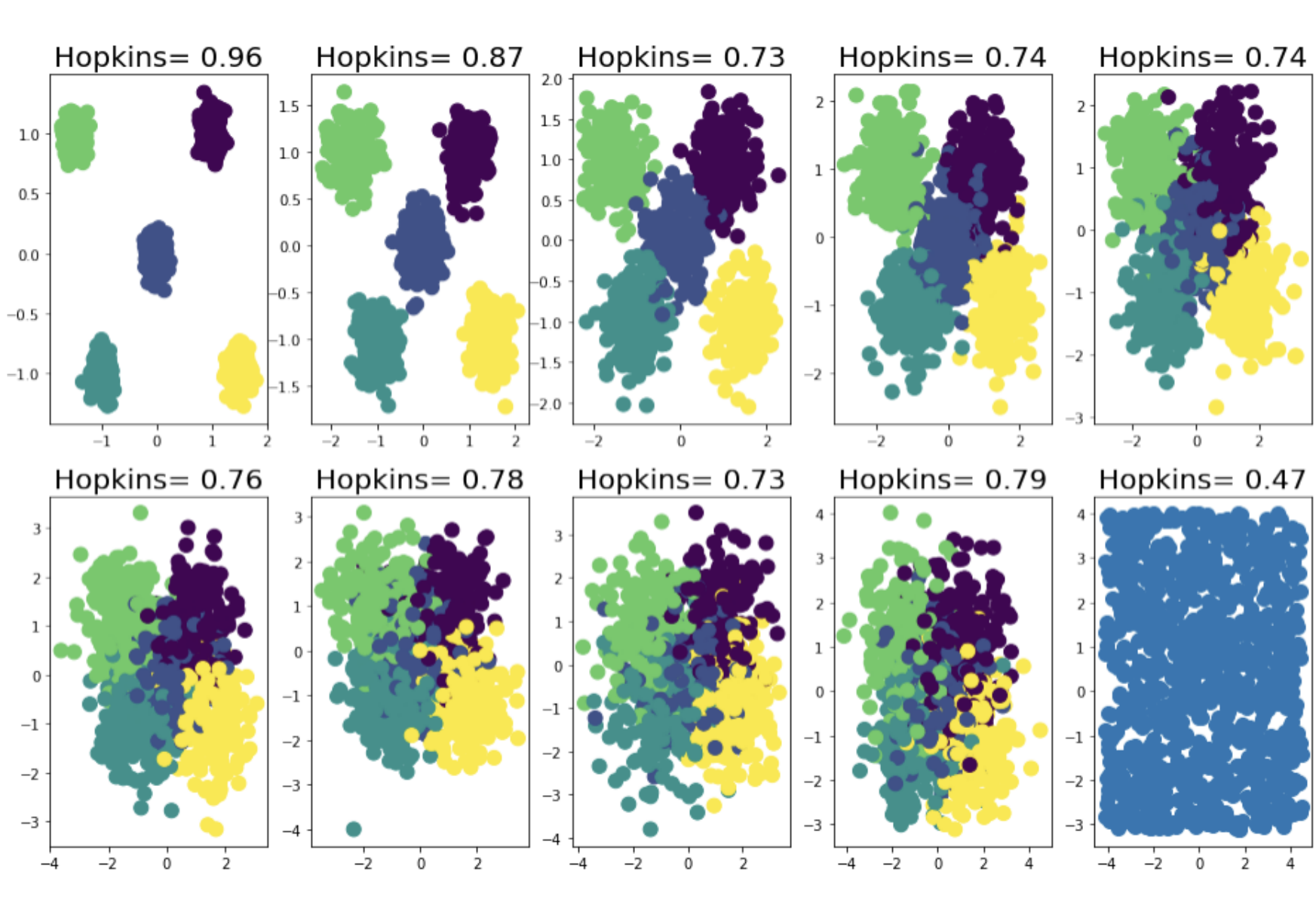
Preparing for clustering

Cluster structure

- Use **Hopkins statistic** to determine spatial randomness
  - $X \leftarrow~$ sample m points from D (dataset) uniformly
    - compute dx_i the distance of each x_i to nearest neighbor in D
  - $Y \leftarrow~$ generate m points uniformly
    - dy_i is the distance of each y_i to nearest neighbor in D
  - if H is close to 0.5 then D is not clusterable
    - $H>0.5$ means good for clustering (some say $H>0.75$)

$$ H=\frac{\sum dy_i}{\sum dy_i + \sum dx_i} $$

Hopkins statistic



The k-means algorithm - performance

The K-means algorithm aims to choose centroids that minimise the **inertia**, or **within-cluster sum-of-squares criterion**:

$\sum_{i=0}^{n}\min_{\mu_j \in C}(||x_i - \mu_j||^2)$

Inertia can be recognized as a measure of **how internally coherent clusters are**.

It suffers from various drawbacks:

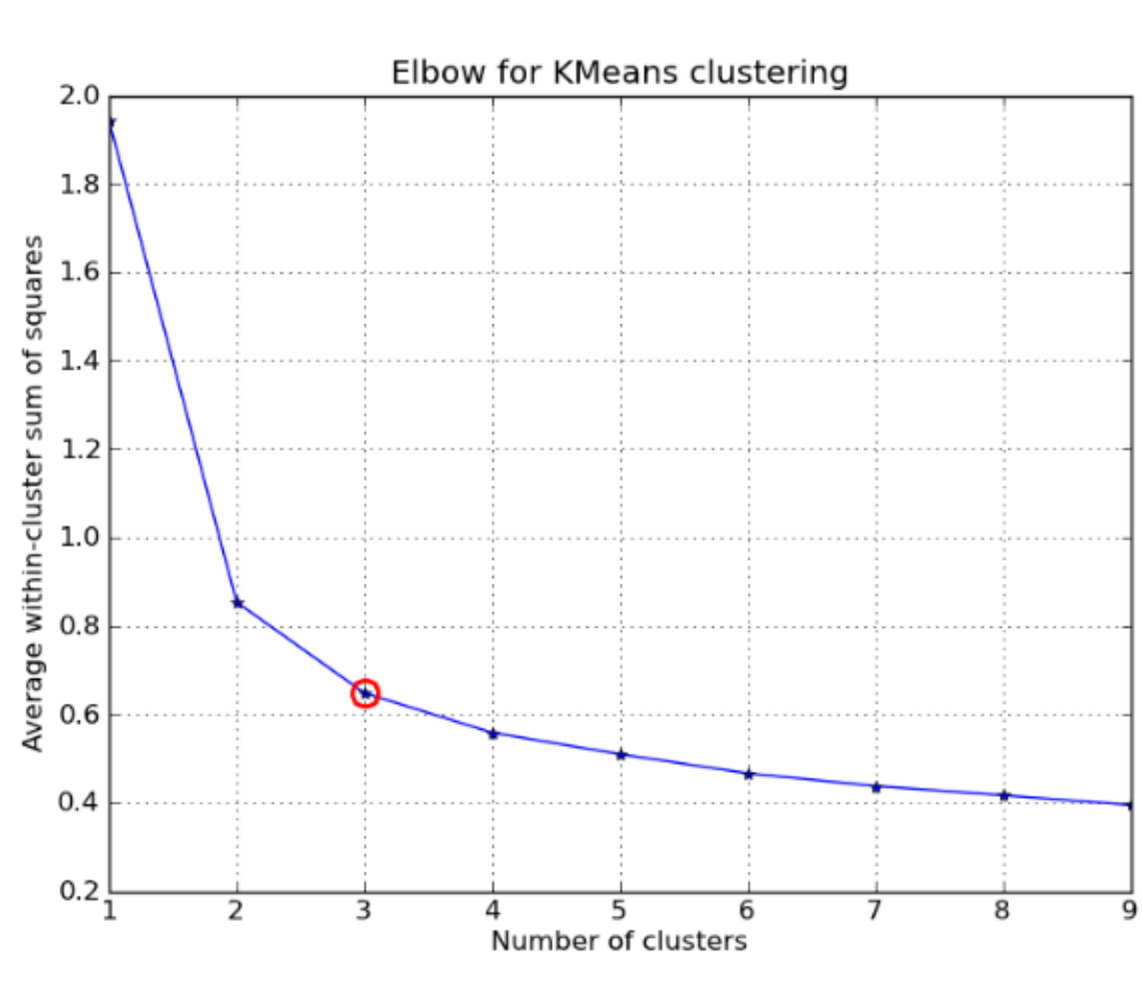
- Inertia makes the assumption that clusters are convex and isotropic, which is not always the case. It responds poorly to elongated clusters, or manifolds with irregular shapes.
- Inertia is not a normalized metric: we just know that lower values are better and zero is optimal. But in very high-dimensional spaces, Euclidean distances tend to become inflated (this is an instance of the so-called "curse of dimensionality").

  Running a dimensionality reduction algorithm such as Principal-Components-Analysis---PCA PCA prior to k-means clustering can alleviate this problem and speed up the computations.

Preparing for clustering

Choosing the number of clusters

- **elbow method**
  - **try** different values for k starting with 1 or around a reasonable number
  - **measure** within-cluster variance (or another quality measure)
    - it may be advisable to **average**
  - **plot** the curve for those values
  - **visually choose** the **turning** point of the curve



Cluster Quality

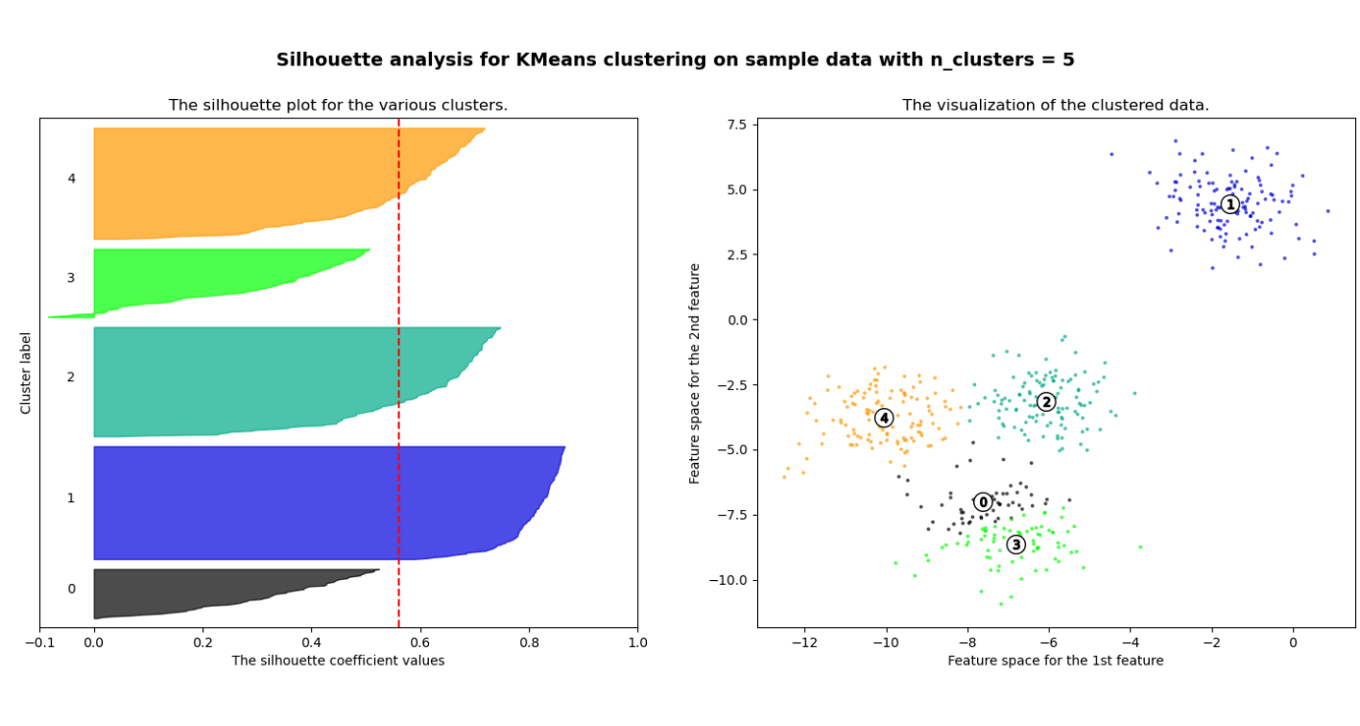
Silhouette coefficient $s({\bf x})$

- Is a measure and a **visualization** of cluster quality
- It helps to identify:
  - **compact** clusters
  - **well separated** clusters
- $s({\bf x})$ tends to 1 if the point is
  - close to other points in same cluster **AND** very far from points in other clusters

Silhouette coefficient

visualization

- Plot **bars** for every point by cluster
- negative values stand out

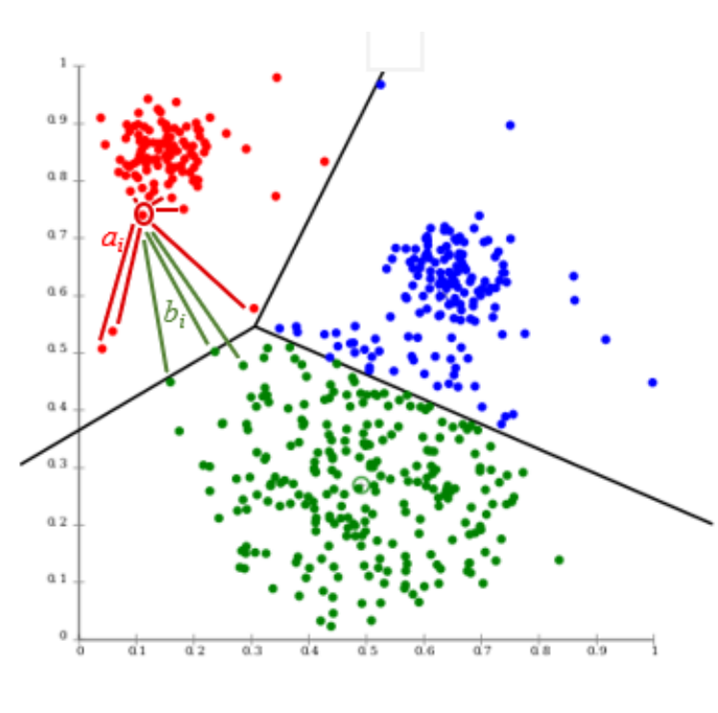


(https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html#sphx-glr-auto-examples-cluster-plot-kmeans-silhouette-analysis-py)

Silhouette coefficient $s({\bf x})$

How to calculate for a point ${\bf x}$

- calculate **average distances** of the point to each cluster
- $a({\bf x})$ the distance within cluster
- $b({\bf x})$ the distance to the nearest cluster
- $s({\bf x}) = (b-a)/\max(b,a)$
- $-1 \le s({\bf x}) \le 1$



https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html

Silhouette Coefficient

If the ground truth labels are not known, evaluation must be performed using the model itself.

- The Silhouette Coefficient is an example of such an evaluation
  - a higher score relates to a model with better defined clusters.
- The Silhouette Coefficient is defined for each sample and is composed of two scores:
  - **a**: The mean distance between a sample and all other points in the same class.
  - **b**: The mean distance between a sample and all other points in the *next nearest cluster*.
- The Silhouette Coefficient *s* for a single sample is then given as:

  $s = \frac{b - a}{max(a, b)}$

The Silhouette Coefficient for a set of samples is given as the mean of the Silhouette Coefficient for each sample. (:func: `sklearn.metrics.silhouette_score`)

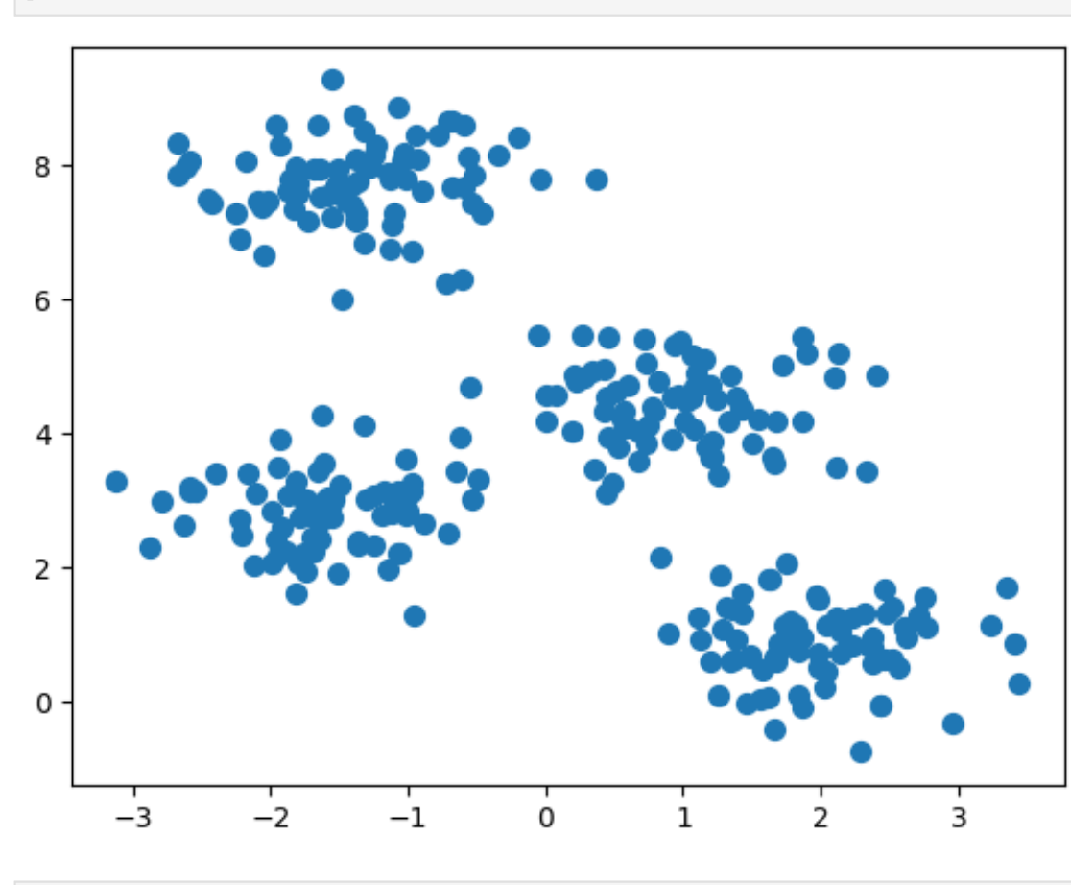
Selecting the number of clusters with silhouette analysis on K-Means clustering

- **Silhouette analysis** can be used to study the separation distance between the resulting clusters.
- The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters and thus provides a way to assess parameters like number of clusters visually.
- This measure has a range of [-1, 1].

(https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html#sphx-glr-auto-examples-cluster-plot-kmeans-silhouette-analysis-py)

In [2]:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs
X, y_true = make_blobs(n_samples=300, centers=4,
                       cluster_std=0.60, random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50);
```



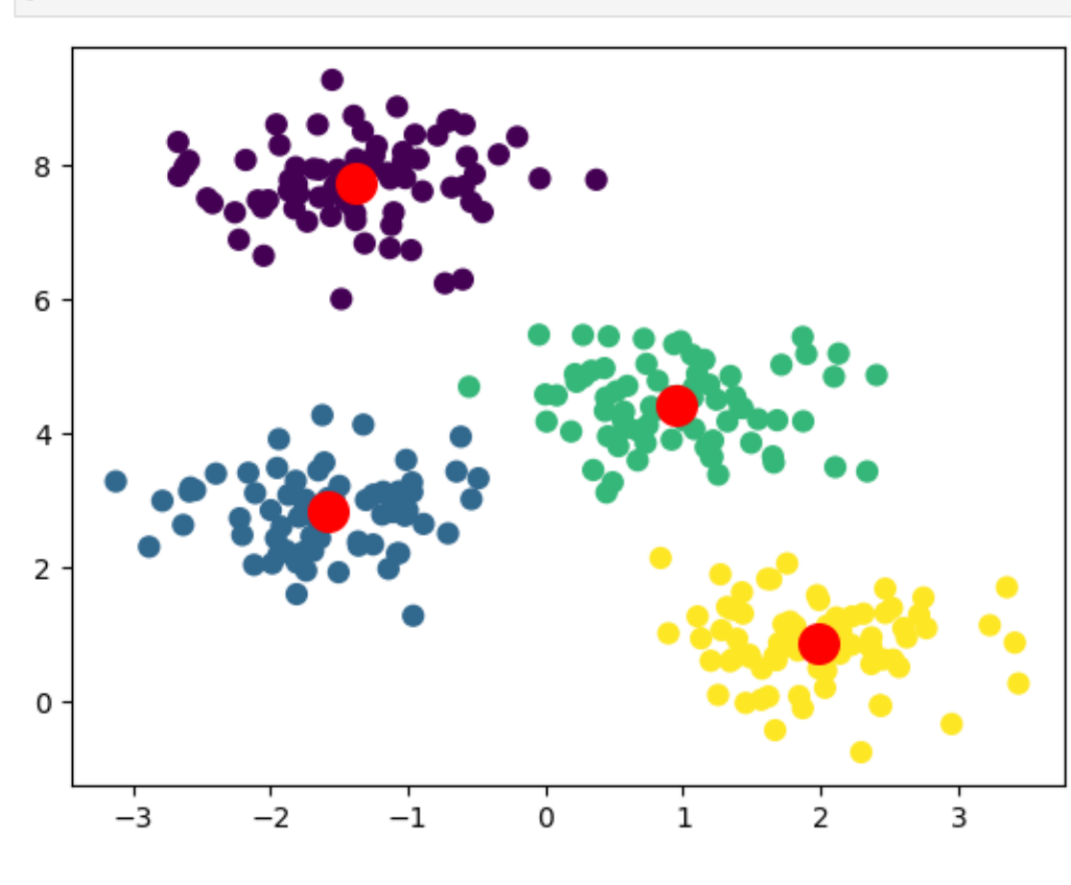
In [3]:

```
import warnings
warnings.filterwarnings("ignore")

from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4);
kmeans.fit(X);
y_kmeans = kmeans.predict(X);
```

The function `KMeans()` performs K-means clustering.
We can visualize the results by plotting the data colored by these labels . We will also plot the cluster centers as determined by the k-means estimator:

In [4]:

```
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=200);
```

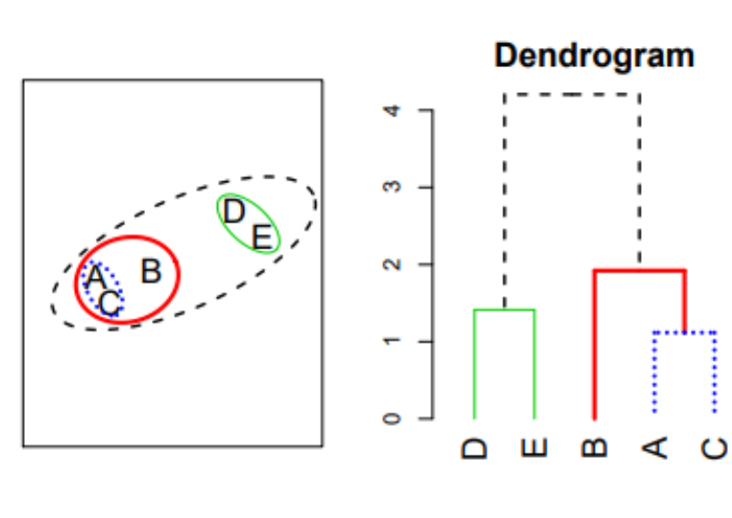


Hierarchical Clustering

- K-means clustering
  - Requires the number of clusters K.
    - This can be a disadvantage
- Hierarchical clustering is an alternative approach
  - Does not require a particular choice of K.
  - We will see bottom-up (or agglomerative) clustering
    - This is the most common type of hierarchical clustering
      - A dendrogram is built starting from the leaves and combining clusters up to the trunk.

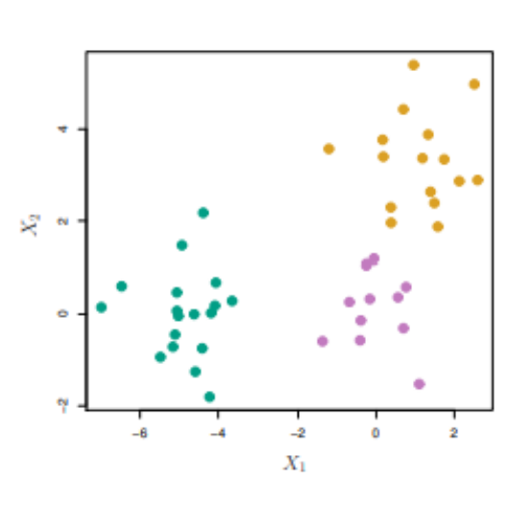
Hierarchical Clustering Algorithm

- 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



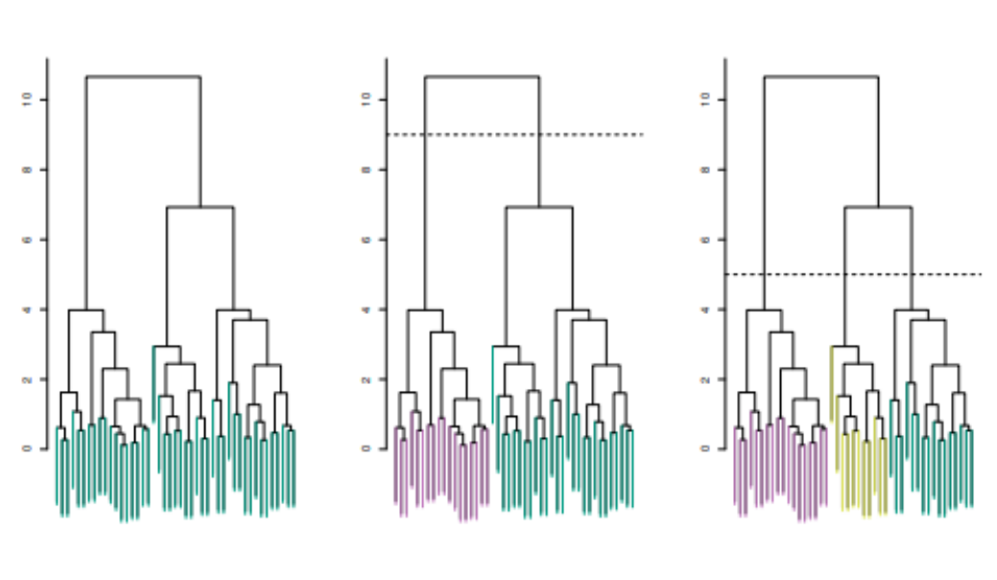
Hierarchical Clustering Algorithm - example

- Consider 45 observations (in 2-dimensional space).
  - There are 3 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.



Example - dendogram

- Left: Dendrogram obtained from hierarchically clustering with complete linkage and Euclidean distance.
- Center: The dendrogram from the left-hand panel, cut at a height of 9 (indicated by the dashed line).
  - This cut results in two distinct clusters, shown in different colors.
- Right: The dendrogram from the left-hand panel, cut at a height of 5.
  - This cut results in three distinct clusters, shown in different colors.

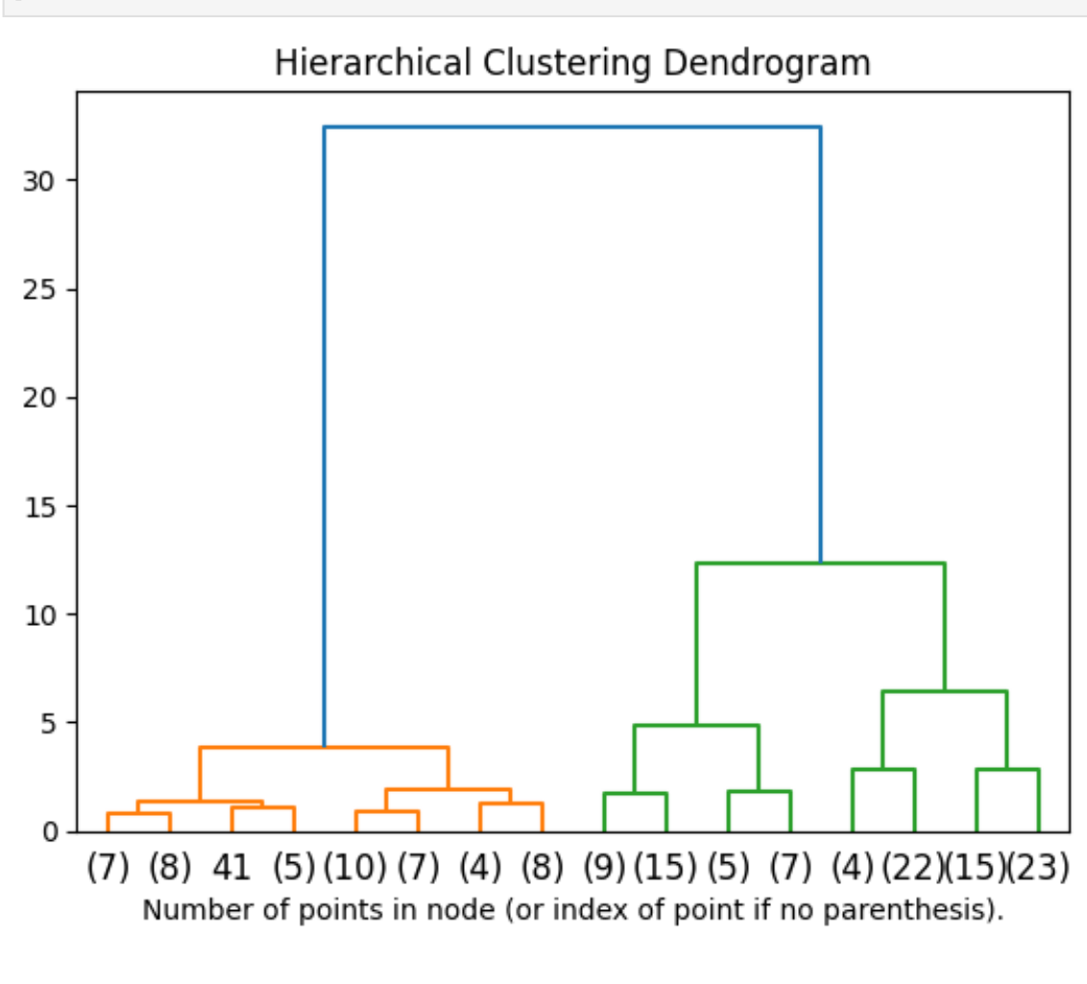


Plot Hierarchical Clustering Dendrogram - sklearn

This example plots the corresponding dendrogram of a hierarchical clustering using AgglomerativeClustering (sklearn) and the dendrogram method available in scipy. (https://scikit-learn.org/stable/auto_examples/cluster/plot_agglomerative_dendrogram.html#sphx-glr-auto-examples-cluster-plot-agglomerative-dendrogram-py)

In [5]:

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram

from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import load_iris


def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack(
        [model.children_, model.distances_, counts]
    ).astype(float)

    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)


iris = load_iris()
X = iris.data

# setting distance_threshold=0 ensures we compute the full tree.
model = AgglomerativeClustering(distance_threshold=0, n_clusters=None)
model = model.fit(X)

plt.title("Hierarchical Clustering Dendrogram")
# plot the top three levels of the dendrogram
plot_dendrogram(model, truncate_mode="level", p=3)
plt.xlabel("Number of points in node (or index of point if no parenthesis).")
plt.show()
```



Example: K-Means on Digits

- Idea:
  - use k-means to identify similar digits without using the original label information;
  - this might be similar to a first step in extracting meaning from a new dataset about which you don't have any a priori label information.

We will start by loading the dataset and find the clusters. This dataset consists of 1,797 samples with 64 features, where each of the 64 features is the brightness of one pixel in an 8 × 8 image.

In [6]:

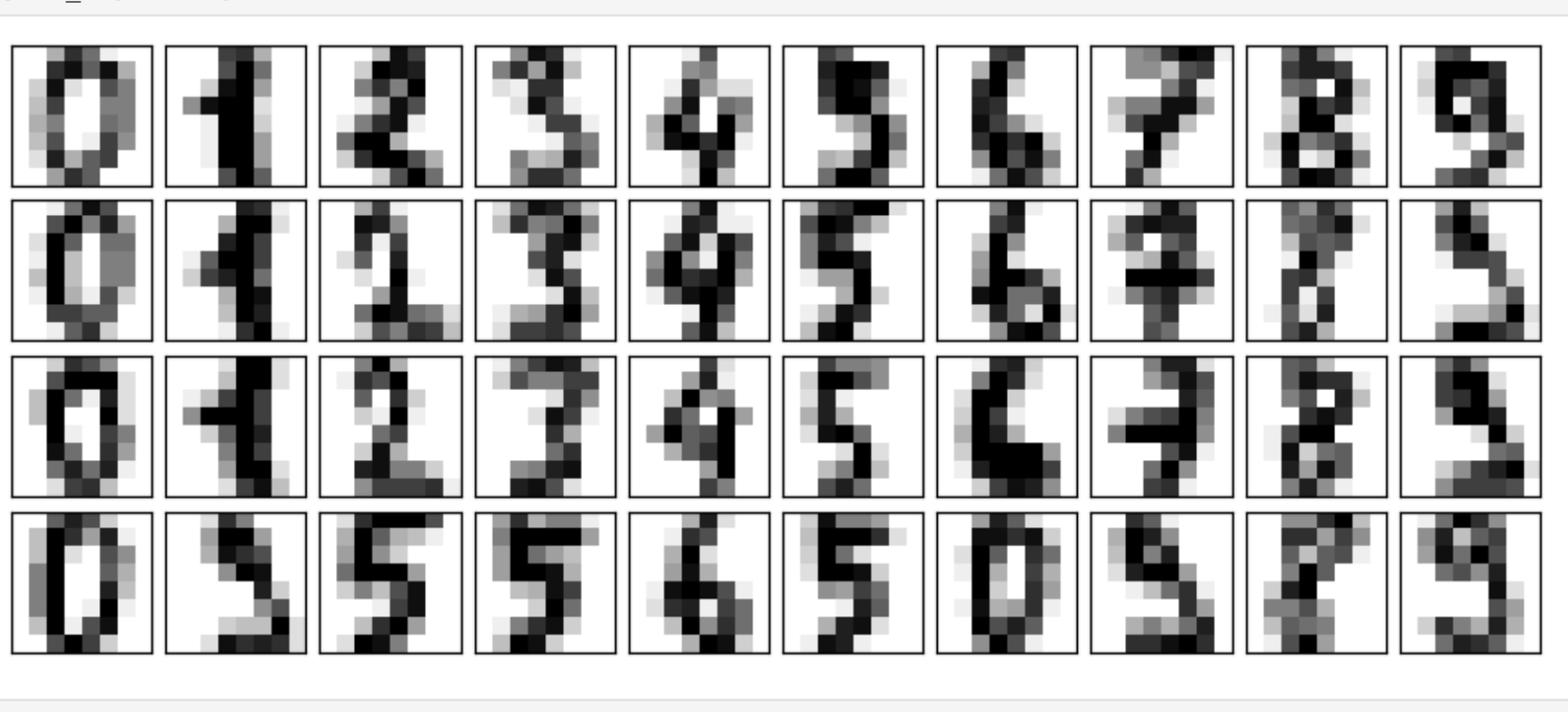
```
from sklearn.datasets import load_digits
digits = load_digits()
digits.data.shape
```

Out[6]:

```
(1797, 64)
```

In [7]:

```
def plot_digits(data):
    fig, axes = plt.subplots(4, 10, figsize=(10, 4),
                             subplot_kw={'xticks':[], 'yticks':[]},
                             gridspec_kw=dict(hspace=0.1, wspace=0.1))
    for i, ax in enumerate(axes.flat):
        ax.imshow(data[i].reshape(8, 8),
                  cmap='binary', interpolation='nearest',
                  clim=(0, 16))
plot_digits(digits.data)
```



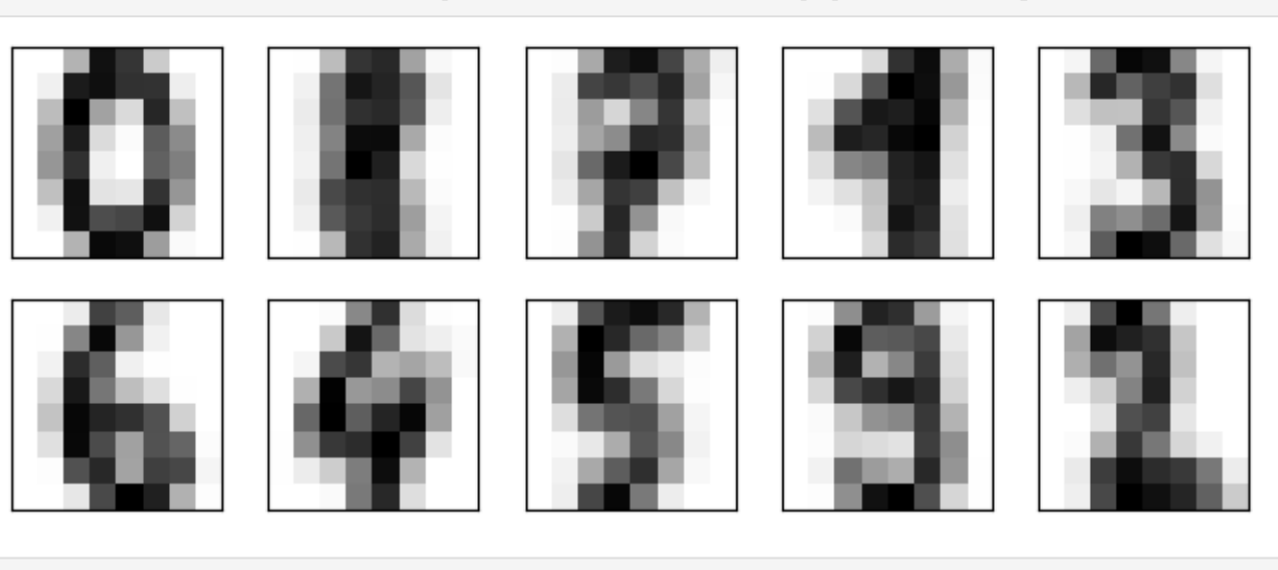
In [8]:

```
kmeans = KMeans(n_clusters=10, random_state=0)
clusters = kmeans.fit_predict(digits.data)
kmeans.cluster_centers_.shape;
```

The result of the clustering, is 10 clusters in 64 dimensions.

In [9]:

```
fig, ax = plt.subplots(2, 5, figsize=(8, 3))
centers = kmeans.cluster_centers_.reshape(10, 8, 8)
for axi, center in zip(ax.flat, centers):
    axi.set(xticks=[], yticks=[])
    axi.imshow(center, interpolation='nearest', cmap=plt.cm.binary)
```



In [10]:

```
from scipy.stats import mode

labels = np.zeros_like(clusters)
for i in range(10):
    mask = (clusters == i)
    labels[mask] = mode(digits.target[mask])[0]
```

In [11]:

```
len(labels)
```

Out[11]:

```
1797
```

In [12]:

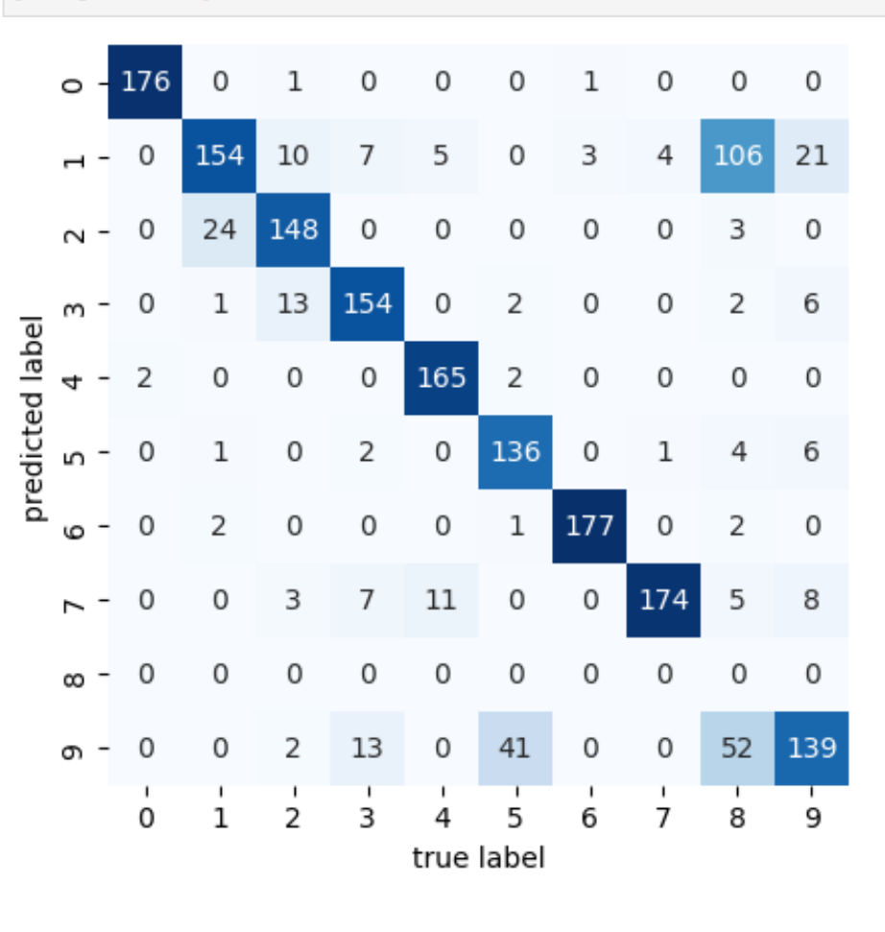
```
from sklearn.metrics import accuracy_score
accuracy_score(digits.target, labels)
```

Out[12]:

```
0.7918753478018921
```

In [13]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
mat = confusion_matrix(digits.target, labels)
sns.heatmap(mat.T, square=True, annot=True, fmt='d',
            cbar=False, cmap='Blues',
            xticklabels=digits.target_names,
            yticklabels=digits.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



Conclusions

- Unsupervised learning is important for
    - understanding the variation and grouping structure of a set of unlabeled data,
    - can be a useful pre-processor for supervised learning
- It is intrinsically more difficult than supervised learning because
    - there is no gold standard (like an outcome variable)
    - and no single objective (like test set accuracy).