Hands on UnSupervised Learning Algorithm with Python, Scikit Learn [case: clustering]

Instructor:
Heru Praptono
[heru.pra@cs.ui.ac.id]

Additional from Hadaiq Rolis' slide, with modification.
some from scikit learn: http://scikit-learn.org/stable/index.html

Agenda

- UnSupervised Learning Algorithm Refresher
- Clustering with Scikit Learn
  - Partitional vs hierarchical
  - DBSCAN
- Visualisation
  - Use Pandas, Matplotlib
- Get your hands dirty

UnSupervised Algorithm Concept Refresher

- Given a number of instances in dataset $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ where $\mathbf{x} \in \mathcal{R}^D$

- **Learning**: Estimate the distribution of $p(\theta|\mathcal{D})$
- **Inference**: So that we find the interesting "pattern" in data $p(\mathbf{x}|\theta)$

UnSupervised Algorithm Concept Refresher

- Some popular tasks:
  - Clustering
  - Dimensionality reduction
  - Latent variable representation
  - … and other, depending on the case..but in principle, “let data tells about its interesting characteristic pattern”

UnSupervised Algorithm Concept Refresher

- Clustering :

  Clustering is the segmentation of objects into different groups, or more precisely, the partitioning of a data set into subsets (clusters), so that the data in each subset (ideally) share some common trait - often according to some defined distance measure.

Clustering

- **Partitional clustering:** Partitional algorithms determine all clusters at once. They include: K-means and its derivatves
- **Hierarchical algorithms**: these fnd successive clusters using previously established clusters
  - <u>Agglomeratve ("botom-up")</u>: Agglomeratve algorithms begin with each element as a separate cluster and merge them into successively larger clusters.
  - <u>Divisive ("top-down")</u>: Divisive algorithms begin with the whole set and proceed to divide it into successively smaller clusters
- **Density-based**: based on connectivity and density functions

Clustering (distance measure)

Distance measure will determine how the *similarity* of two elements is calculated and it will influence the shape of the clusters.

They include for example:

1. The Euclidean distance (also called 2-norm distance) is given by:

$$d(x,y) = \sqrt[2]{\sum_{i=1}^{p} |x_i - y_i|^2}$$

2. The Manhattan distance (also called taxicab norm or 1-norm) is given by:

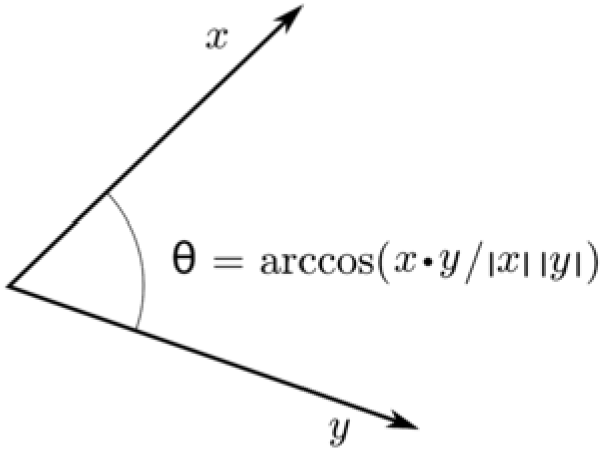
$$d(x,y) = \sum_{i=1}^{p} |x_i - y_i|$$

Clustering (distance measure)

3. The maximum norm is given by:

$$d(x, y) = \max_{1 \le i \le p} | x_i - y_i |$$

4. Inner product space: The angle between two vectors can be used as a distance measure when clustering high dimensional data



Good Clustering…

- A good clustering method will produce high quality clusters with
  - high intra-class similarity
  - low inter-class similarity
- The quality of a clustering result depends on both the similarity measure used by the method and its implementation
- The quality of a clustering method is also measured by its ability to discover some or all of the hidden patterns

Requirements of Clustering

- Scalability
- Ability to deal with different types of attributes
- Discovery of clusters with arbitrary shape
- Minimal requirements for domain knowledge to determine input parameters
- Able to deal with noise and outliers
- Insensitive to order of input records
- High dimensionality
- Incorporation of user-specified constraints
- Interpretability and usability

Clustering (k-Means)

- The **k-means algorithm** is an algorithm to cluster *n* objects based on attributes into *k* partitions, where *k* < *n*.
- An algorithm for partitioning (or clustering) N data points into K disjoint subsets S_j containing data points so as to minimize the sum-of-squares criterion

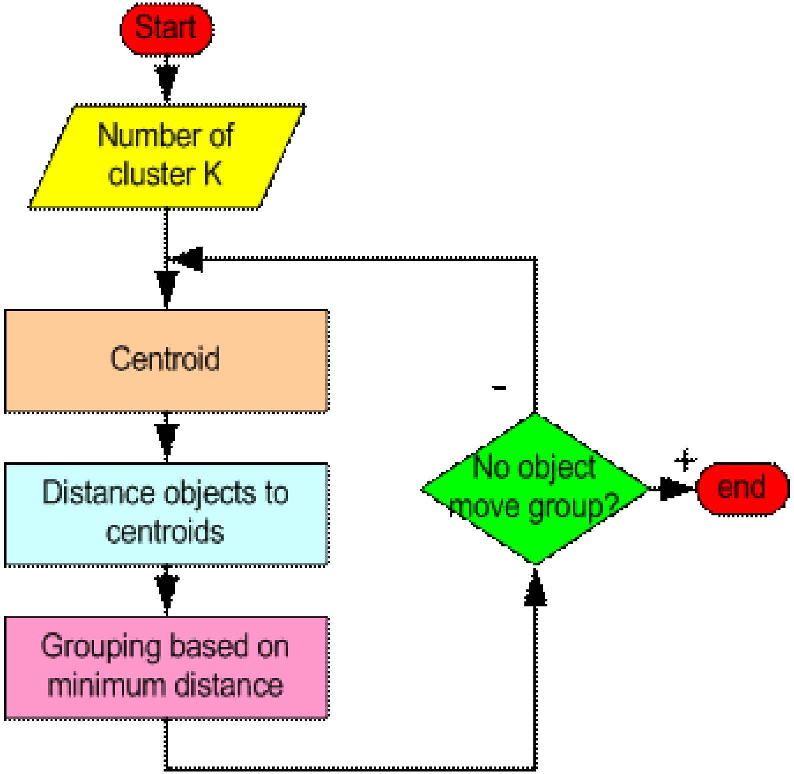
$$J = \sum_{j=1}^{K} \sum_{n \in S_j} |x_n - \mu_j|^2,$$

- where x_n is a vector representing the the n^{th} data point and u_j is the geometric centroid of the data points in S_j.

We will demonstrate on how k-means works

- Simply speaking k-means clustering is an algorithm to classify or to group the objects based on attributes/features into K number of group.
- K is positive integer number.
- The grouping is done by minimizing the sum of squares of distances between data and the corresponding cluster centroid.

How k-Means works



Let us see..

Individual	Variable 1	Variable 2
1	1.0	1.0
2	1.5	2.0
3	3.0	4.0
4	5.0	7.0
5	3.5	5.0
6	4.5	5.0
7	3.5	4.5

K-means: Example

Step 1:
Initialization: Randomly we choose following two centroids (k=2) for two clusters.
In this case the 2 centroid are: m1=(1.0,1.0) and m2=(5.0,7.0).

Individual	Variable 1	Variable 2
1	1.0	1.0
2	1.5	2.0
3	3.0	4.0
4	5.0	7.0
5	3.5	5.0
6	4.5	5.0
7	3.5	4.5

Chosen as initial centroids

	Individual	Mean Vector
Group 1	1	(1.0, 1.0)
Group 2	4	(5.0, 7.0)

K-means: Example

Step 2:

- Thus, we obtain two clusters containing: {1,2,3} and {4,5,6,7}.
- Their new centroids are:

$$m_1 = (\frac{1}{3}(1.0+1.5+3.0), \frac{1}{3}(1.0+2.0+4.0)) = (1.83, 2.33)$$

$$m_2 = (\frac{1}{4}(5.0+3.5+4.5+3.5), \frac{1}{4}(7.0+5.0+5.0+4.5))$$

$$= (4.12, 5.38)$$

Individual	Centroid 1	Centroid 2
1	0	7.21
2 (1.5, 2.0)	1.12	6.10
3	3.61	3.61
4	7.21	0
5	4.72	2.5
6	5.31	2.06
7	4.30	2.92

$$d(m_1, 2) = \sqrt{|1.0-1.5|^2 + |1.0-2.0|^2} = 1.12$$

$$d(m_2, 2) = \sqrt{|5.0-1.5|^2 + |7.0-2.0|^2} = 6.10$$

K-means: Example

Step 3:

- Now using these centroids we compute the Euclidean distance of each object, as shown in table.

- Therefore, the new clusters are:

  {1,2} and {**3**,4,5,6,7}

- Next centroids are: m1=(1.25,1.5) and m2 = (3.9,5.1)

Individual	Centroid 1	Centroid 2
1	1.57	5.38
2	0.47	4.28
3	2.04	1.78
4	5.64	1.84
5	3.15	0.73
6	3.78	0.54
7	2.74	1.08

K-means: Example

- <u>Step 4 </u>:

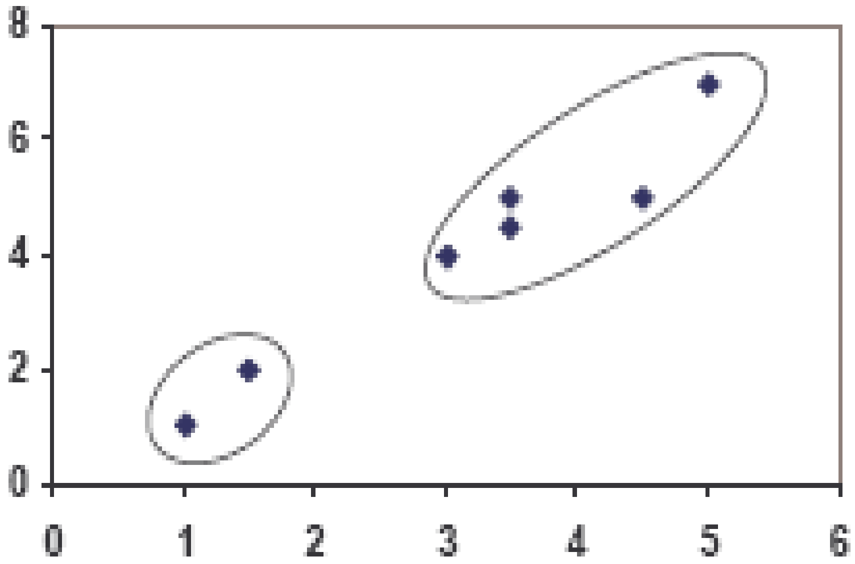
  The clusters obtained are:

  {1,2} and {3,4,5,6,7}

- Therefore, there is no change in the cluster.
- Thus, the algorithm comes to a halt here and final result consist of 2 clusters {1,2} and {3,4,5,6,7}.

Individual	Centroid 1	Centroid 2
1	0.56	5.02
2	0. 56	3.92
3	3.05	1.42
4	6.66	2.20
5	4.16	0.41
6	4.78	0.61
7	3.75	0.72

K-means: Example



Let’s try [the example] with scikit-learn

Clustering (K-Means)

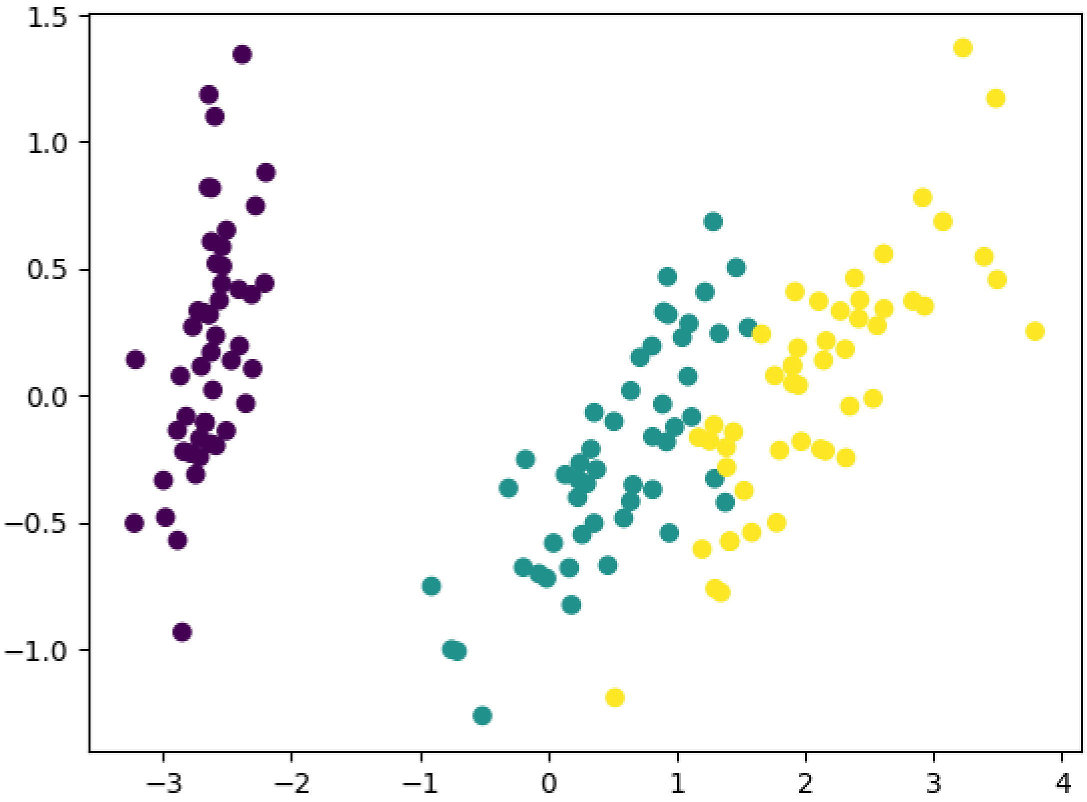
```
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
import pylab as pl
iris = load_iris()
pca = PCA(n_components=2).fit(iris.data)
pca_2d = pca.transform(iris.data)
pl.figure('Reference Plot')
pl.scatter(pca_2d[:, 0], pca_2d[:, 1], c=iris.target)
```

Clustering (K-Means)

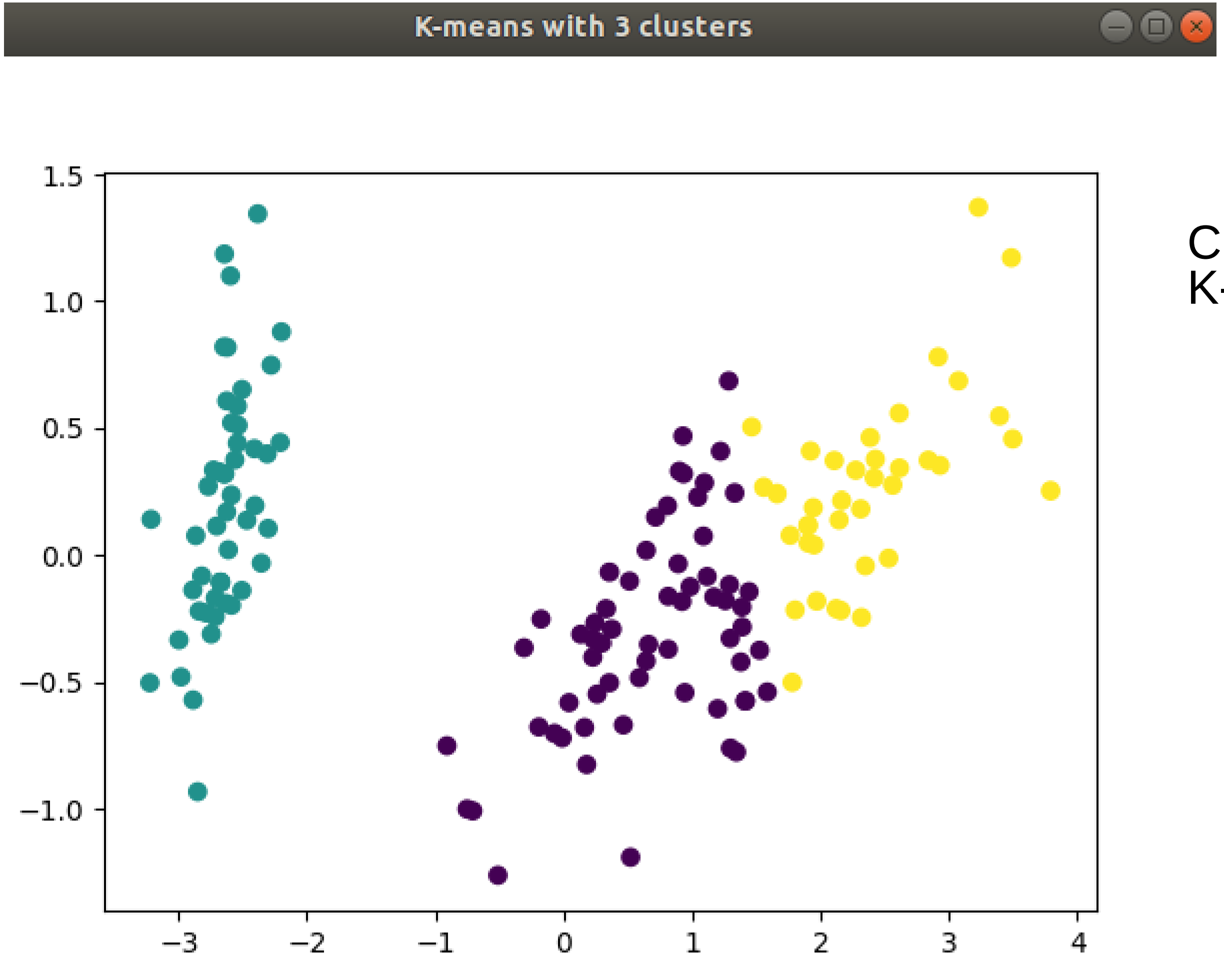
```
kmeans = KMeans(n_clusters=3, random_state=111)
kmeans.fit(iris.data)
pl.figure('K-means with 3 clusters')
pl.scatter(pca_2d[:, 0], pca_2d[:, 1],
c=kmeans.labels_)
pl.show()
```

Clustering (K-Means)

Reference data



Clustering (K-Means)



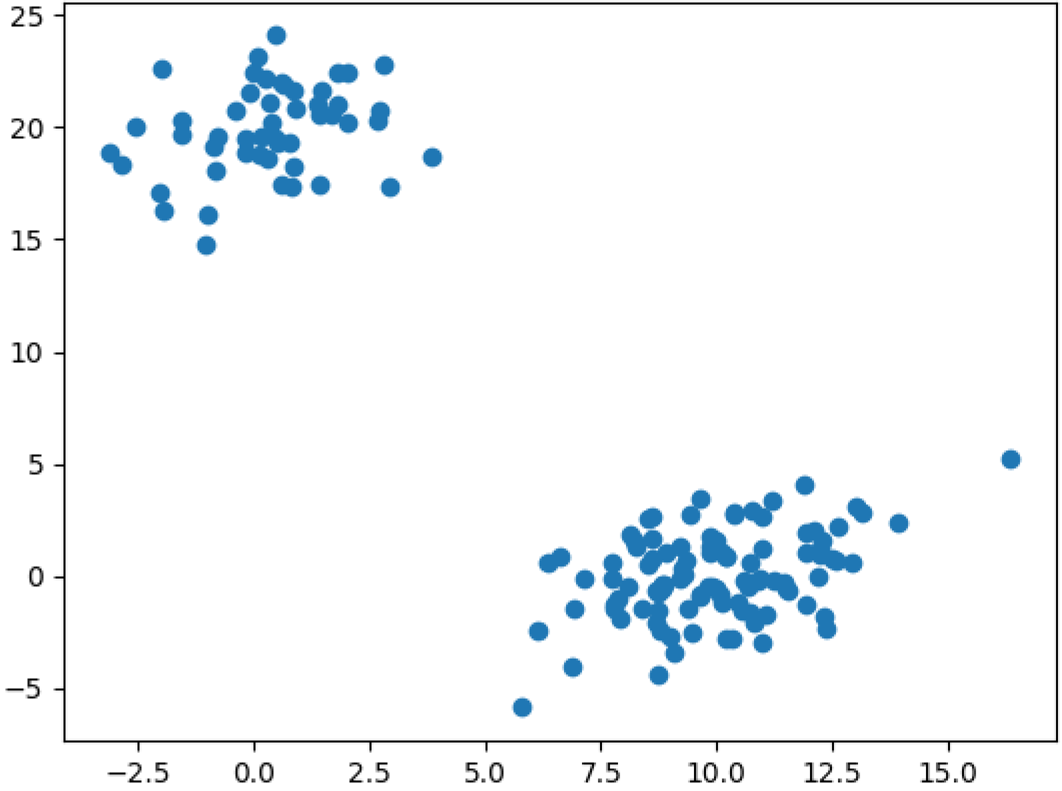
Clustered with K-Means

The others: (1) Hierarchical Clustering

Hierarchical Clustering

```
from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
import numpy as np
np.random.seed(4711) # for repeatability of this tutorial
a = np.random.multivariate_normal([10, 0], [[3, 1], [1, 4]], size=[100,])
b = np.random.multivariate_normal([0, 20], [[3, 1], [1, 4]], size=[50,])
X = np.concatenate((a, b),axis = 0)
print(X.shape) # 150 samples with 2 dimensions
plt.scatter(X[:,0], X[:,1])
plt.show()
```

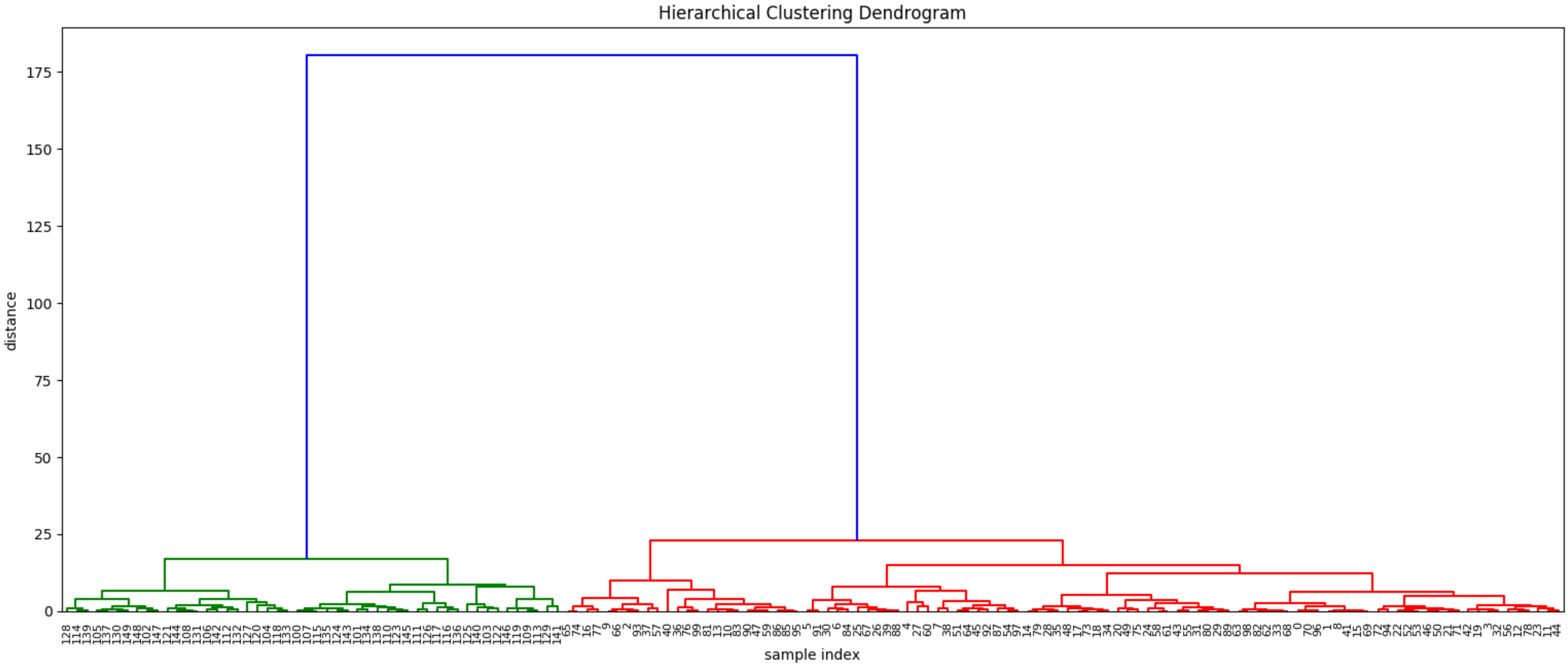
Hierarchical Clustering



Hierarchical Clustering

```
Z = linkage(X, 'ward')
plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(
    Z,
    leaf_rotation=90., # rotates the x axis labels
    leaf_font_size=8., # font size for the x axis labels
)
plt.show()
```

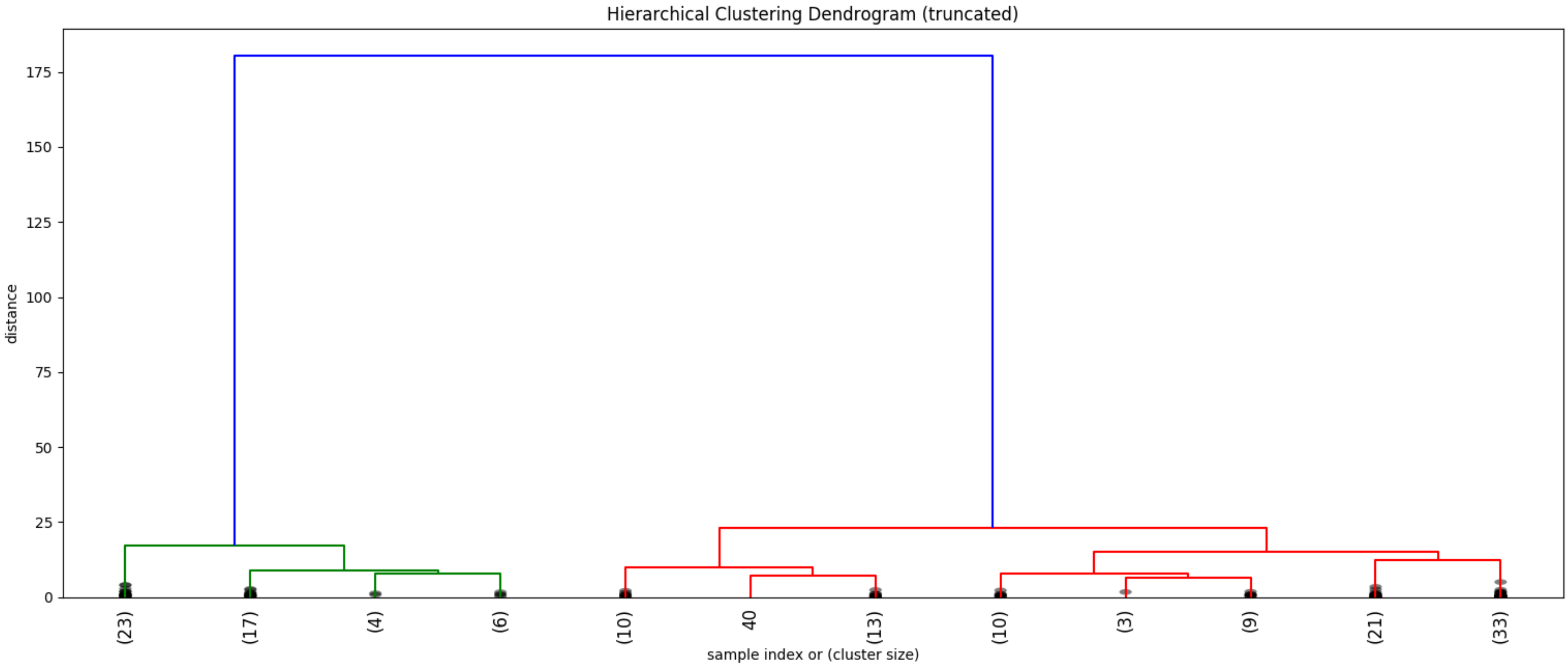
Hierarchical Clustering



Hierarchical Clustering (Truncating)

```
plt.title('Hierarchical Clustering Dendrogram (truncated)')
plt.xlabel('sample index or (cluster size)')
plt.ylabel('distance')
dendrogram(
    Z,
    truncate_mode='lastp', # show only the last p merged clusters
    p=12, # show only the last p merged clusters
    leaf_rotation=90.,
    leaf_font_size=12.,
    show_contracted=True, # to get a distribution impression in truncated
    branches
)
plt.show()
```

Hierarchical Clustering (Truncating)



The others: (2) DBSCAN

DBSCAN

```
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets.samples_generator import make_blobs
from sklearn.preprocessing import StandardScaler
# ######Generate sample Data
centers = [[5, 5], [-5, -1], [0, 0]]
X, labels_true = make_blobs(n_samples=750, centers=centers,
cluster_std=0.4, random_state=0)
X = StandardScaler().fit_transform(X)
```

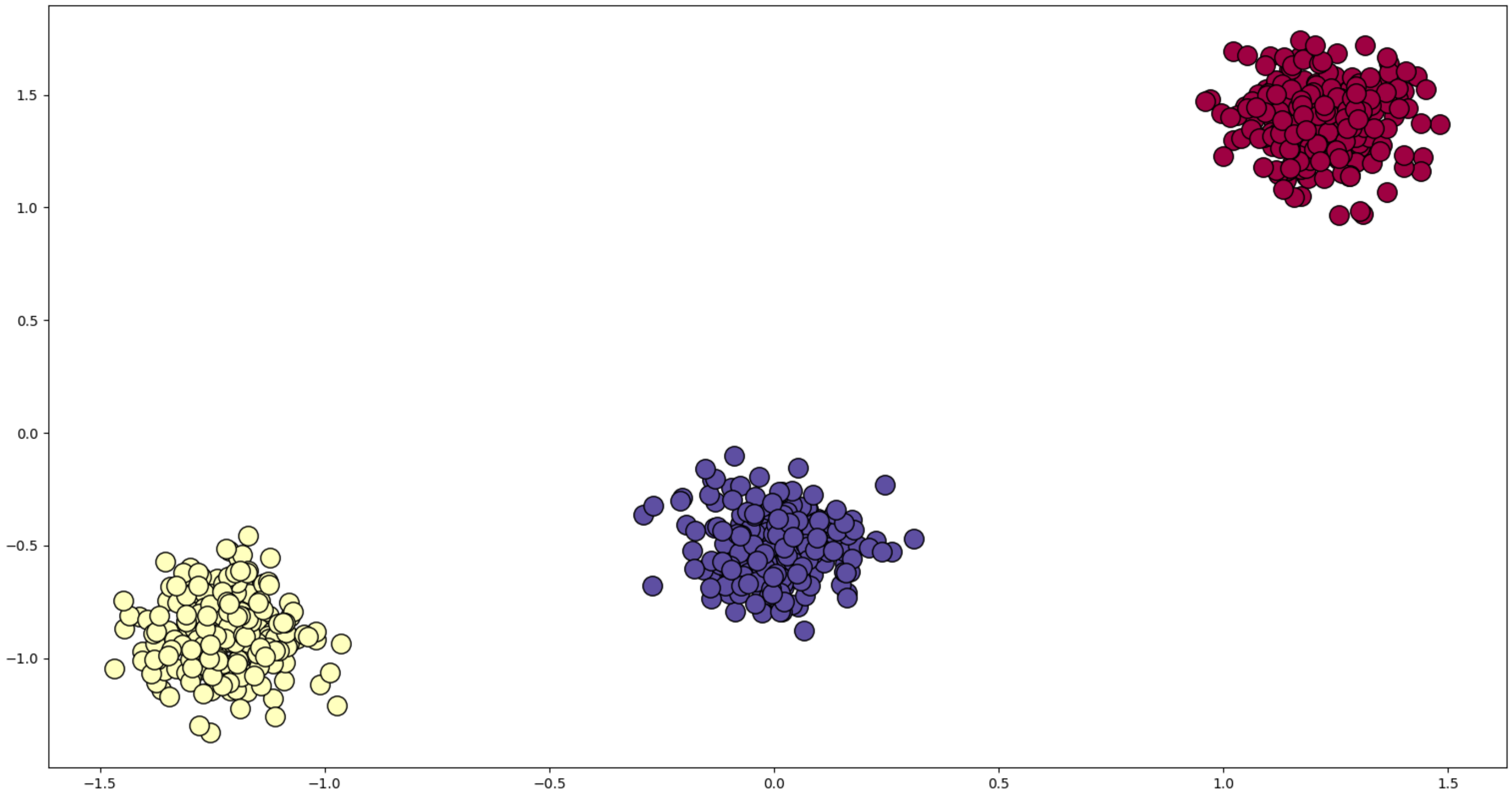
DBScan

```
# Compute DBSCAN
db = DBSCAN(eps=0.3, min_samples=10).fit(X)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_
# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
print('Estimated number of clusters: %d' % n_clusters_)
print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true,
labels))
print("Silhouette Coefficient: %0.3f"% metrics.silhouette_score(X,labels))
```

DBSCAN

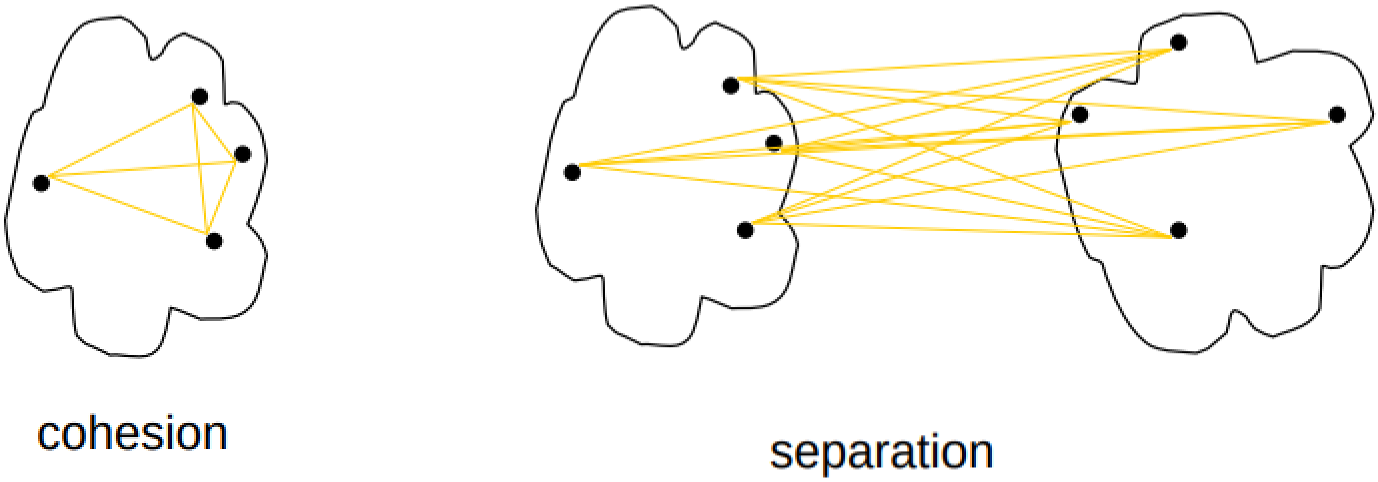
```
import matplotlib.pyplot as plt
unique_labels = set(labels)
colors = [plt.cm.Spectral(each)for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
      # Black used for noise.
      col = [0, 0, 0, 1]
    class_member_mask = (labels == k)
    xy = X[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
    markeredgecolor='k', markersize=14)
    xy = X[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
    markeredgecolor='k', markersize=6)
#plt.title('Estimated number of clusters: %d' % n_clusters_)
plt.show()
```

DBSCAN



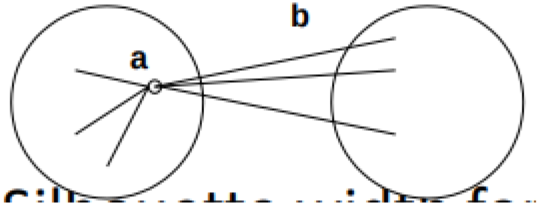
Cohesion and Separation in Clustering

- A proximity graph based approach can also be used for cohesion and separation.
  - Cluster cohesion is the sum of the weight of all links within a cluster.
  - Cluster separation is the sum of the weights between nodes in the cluster and nodes outside the cluster



Sihoutte Coefficient/Index in Clustering

- Silhouette Coefficient combine ideas of both cohesion and separation, but for individual points, as well as clusters and clusterings
- For an individual point, I
  - Calculate a = average distance of i to the points in its cluster
  - Calculate b = min (average distance of i to points in another cluster)
  - The silhouette coefficient for a point is then given by
  - $s = 1 – a/b$ if $a < b$, (or $s = b/a - 1$ if $a >= b$, not the usual case)
  - Typically between 0 and 1.
  - The closer to 1 the better.



- Can calculate the Average Silhouette width for a cluster or a clustering

It is now your turn..and explore on your own