Principles and Practices of Data Science

Lecture 11

Melvin Ayala

Lecture 11: Clustering Techniques

Sections

1. What is Clustering?

Learning Algorithms:

- supervised learning: we have labeled data
- semi-supervised learning: we have a large data set where some of the data is labeled but most of it isn't.
- unsupervised learning: we have a data set that is completely unlabeled.

Clustering Algorithms:

- unsupervised machine learning task.
- cluster analysis
- needs a lot of input data with no labels and finds groupings in the data if it can.
- groupings are called clusters.
- cluster: a group of data points that are similar to each other based on their relation to surrounding data points.
- clustering is used for things like feature engineering or pattern discovery.
- recommended as starting technique when you're starting with data you know nothing about

2. Types of Clustering Methods

- clustering helps in performing surface-level analyses of the unstructured data.
- cluster formation: depends upon different parameters like shortest distance, graphs, and density of the data points.
- grouping into clusters: conducted by finding the measure of similarity between the objects based on some metric called the similarity measure.
- it is easier to find similarity measures in a lesser number of features.
- creating similarity measures becomes a complex process as the number of features increases.
- different types of clustering approaches in data mining use different methods to group the data.

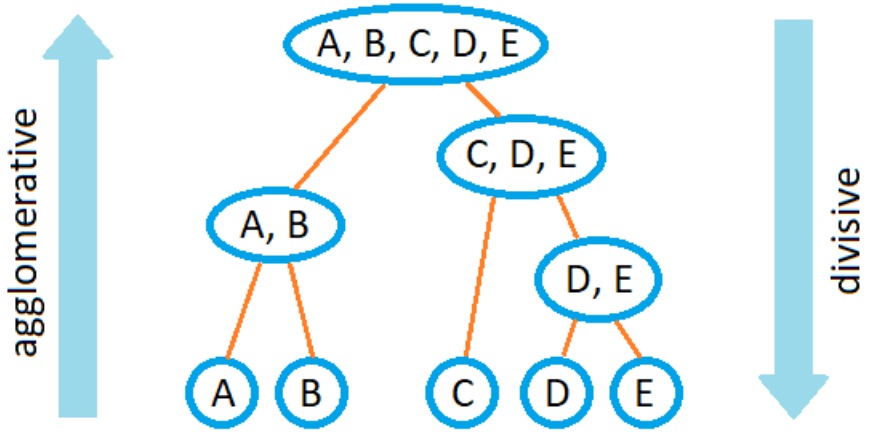
Some of the various types of clustering techniques:

- Connectivity-based Clustering (Hierarchical clustering)
- Centroids-based Clustering (Partitioning methods)
- Distribution-based Clustering
- Density-based Clustering (Model-based methods)
- Fuzzy Clustering
- Constraint-based (Supervised Clustering)

2.1. Hierarchical Clustering (Connectivity-based)

Hierarchical Clustering

- unsupervised learning method for clustering data points.
- builds clusters by measuring the dissimilarities between data.
- unsupervised learning:
  - a model does not have to be trained
  - we do not need a "target" variable
- used to visualize and interpret the relationship between individual data points.
- based on the principle that every object is connected to its neighbors depending on their proximity distance (degree of relationship).
- clusters are represented in extensive hierarchical structures separated by a maximum distance required to connect the cluster parts.
- **clusters are represented as dendrograms:**
  - X-axis represents the objects that do not merge
  - Y-axis is the distance at which clusters merge
- similar data objects have minimal distance falling in the same cluster.
- dissimilar data objects are placed farther in the hierarchy.
- **according to the data flow chosen:**
  - divisive approach
  - agglomerative approach
- **visualize the clusters using:**
  - a dendrogram
  - scatter plot



2.1.1. Divisive Approach

Divisive approach of hierarchical clustering follows a top-down method:

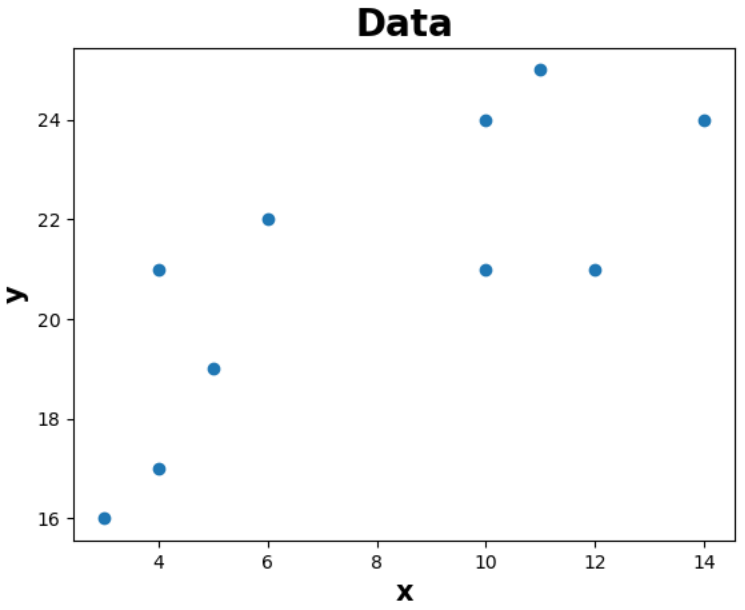
- considers that all the data points belong to one large cluster
- tries to divide the data into smaller groups based on a termination logic
- termination logic:
  - based on a criterion
  - a point beyond which there will be no further division of data points.
  - can be based on the minimum sum of squares of error inside a cluster, or for categorical data, the metric can be the GINI coefficient inside a cluster.
- highly “rigid” when splitting the clusters: once a clustering is done inside a loop, task cannot be undone.

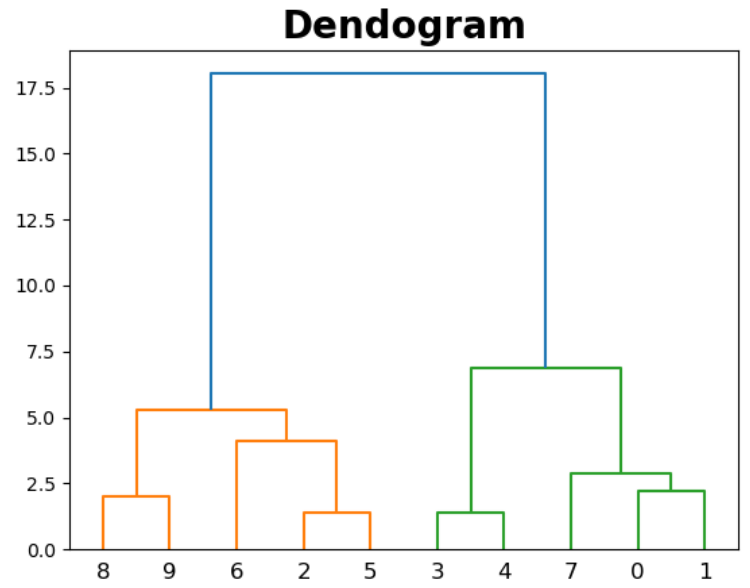
Divisive Approach (Coding Example)

Code:

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

x = [4, 5, 10, 4, 3, 11, 14, 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

plt.figure()
plt.title("Data", fontsize=20, fontweight='bold')
plt.scatter(x, y)
plt.xlabel('x', fontsize=10, fontweight='bold')
plt.ylabel('y', fontsize=10, fontweight='bold')
plt.show(block=False)

data = list(zip(x, y))
linkage_data = linkage(data, method='ward', metric='euclidean')
plt.figure()
dendrogram(linkage_data)
plt.title("Dendogram", fontsize=20, fontweight='bold')
plt.show(block=False)

print("End of Program")
```

Output:





2.1.2. Agglomerative Approach

Agglomerative is the contrary to Divisive:

- all the “N” data points are considered to be a single member of “N” clusters that the data is comprised into.
- iteratively combine these numerous “N” clusters to a fewer number of clusters (k)
- assign the data points to each of these clusters accordingly.
- approach is bottom-up
- uses a termination logic in combining the clusters.
- termination logic can be:
  - a number-based criterion (no more clusters beyond this point)
  - a distance criterion (clusters should not be too far apart to be merged)
  - a variance criterion (increase in the variance of the cluster being merged should not exceed a threshold, Ward Method)

Algorithm:

- start by treating each data point as its own cluster.
- join clusters together that have the shortest distance between them to create larger clusters.
- repeat step above until one large cluster is formed containing all of the data points.

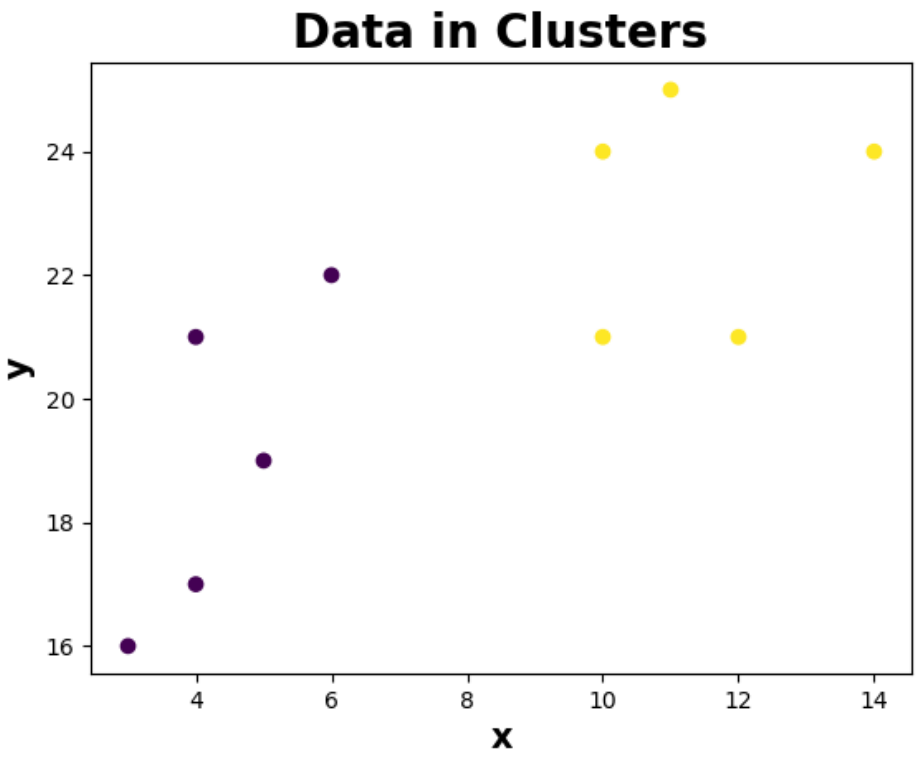
Note: need to decide on both a distance and linkage method. We will use Euclidean distance and the Ward linkage method, which attempts to minimize the variance between clusters.

Agglomerative Approach (Coding Example)

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

data = list(zip(x, y))

hierarchical_cluster = AgglomerativeClustering(n_clusters=2,
affinity='euclidean', linkage='ward')
labels = hierarchical_cluster.fit_predict(data)
print(labels)

plt.figure()
plt.title("Data in Clusters", fontsize=20, fontweight='bold')
plt.scatter(x, y, c=labels)
plt.xlabel('x', fontsize=15, fontweight='bold')
plt.ylabel('y', fontsize=15, fontweight='bold')
plt.show(block=False)

print("End of Program")
```

Output:



Labels = [0 0 1 0 0 1 1 0 1 1]

Agglomerative Wards Linkage Method

- criterion applied in hierarchical cluster analysis.
- steps:
  - starts with declaring each data point as an individual cluster (of size = 1)
  - then, fusions clusters following a criterion
- cluster fusion criterion:
  - based on the optimal value of an objective function.
- **objective function:**
  - any function that reflects the investigator's purpose.
  - Ward method's objective function: error sum of squares (to minimize)
  - minimize the total within-cluster variance.
- linkage function specifying the distance between two clusters is computed as the increase in the "error sum of squares" (ESS) after fusing two clusters into a single cluster.
- successive clustering steps are chosen so as to **minimize the increase in ESS** at each step.

Agglomerative Wards Linkage Method (Cont.)

ESS of a set = sum of squares of the deviations from the mean value or the mean vector (centroid).
ESS of a cluster j:

$$ESS_j = \sum_{i=1}^{n_j} |x_{ij} - c_j|^2$$

where:

j	cluster
n_j	number of elements in cluster j
x_{ij}	data point I from cluster j
c_j	centroid of cluster j

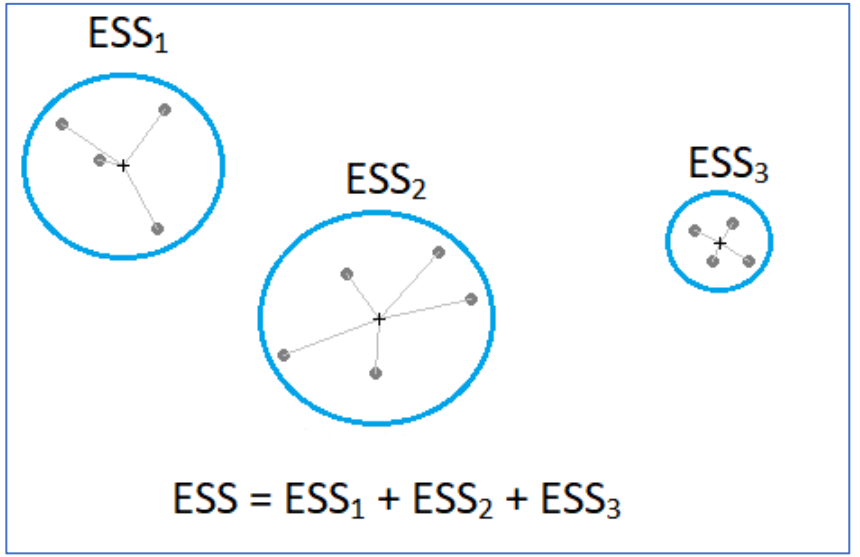
with $c_j = \frac{1}{n_j} \sum_{i=1}^{n_j} x_{ij}$

The error sum for all clusters: $ESS = ESS_1 + ESS_2 + \ldots + ESS_k$
where k is the number of clusters.
The linkage function represents the distance between 2 clusters and is described by the following expression:

$$D = ESS_{12} - (ESS_1 + ESS_2)$$

where

ESS_{12}	ESS resulting from fusing clusters 1 and 2
ESS_j	ESS for cluster j



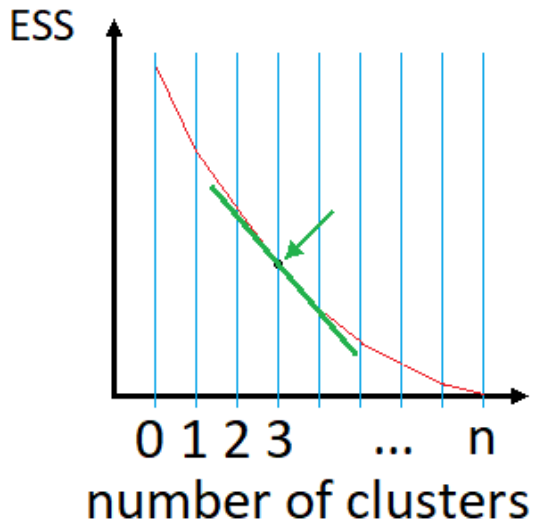
Agglomerative Wards Linkage Method (Cont.)

Steps:

1. Set all points as singletons (clusters containing a single point)
2. Find the pair of clusters that lead to a minimum increase in the total within-cluster variance after merging
3. Evaluate **stopping condition**. If false, repeat steps 2 and 3.

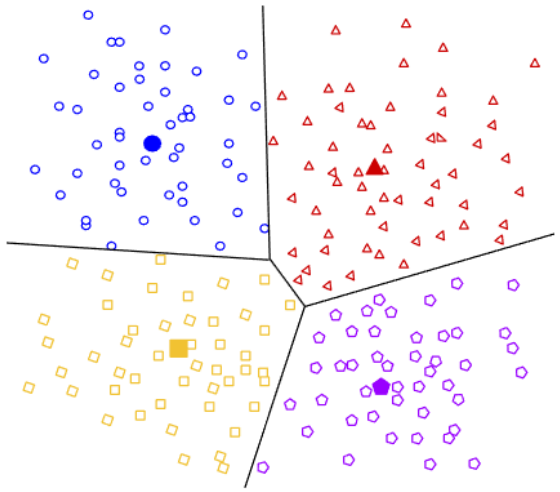
Stopping Condition:

- Could be any of the following:
  - Desired number of clusters reached (if any)
  - Highest increase in SSE obtained if next clusters are merged (found retrospectively)
  - Find k where a measure such as the Calinski and Harabasz index is the largest (found retrospectively)



2.2. Centroid-Based or Partition Clustering

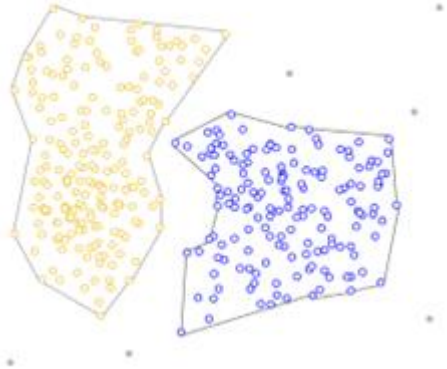
- organizes the data into non-hierarchical clusters (in contrast to hierarchical clustering)
- efficient but sensitive to initial conditions and outliers
- the easiest of all the clustering types in data mining
- works on the closeness of the data points to the chosen central value
- datasets are divided into a given number of clusters, and a vector of values references every cluster
- input data variable is compared to the vector value and enters the cluster with minimal difference.
- pre-defining the number of clusters:
  - the most crucial yet most complicated stage for the clustering approach.
- despite drawback, it is a vastly used clustering approach
- the K-Means algorithm lies in this category.



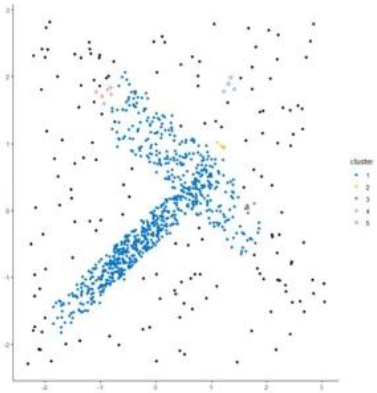
- k-means:
  - is the most widely-used centroid-based clustering algorithm.
  - efficient, effective, and simple clustering algorithm.

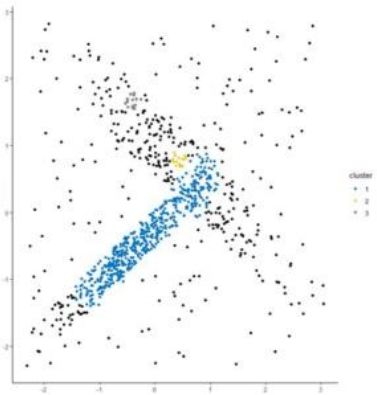
- Measure between clusters and data points:
  - Distance metrics:
    - Euclidian distance
    - Manhattan Distance
    - Minkowski Distance

2.3. Density-Based Clustering (Model-Based Methods)



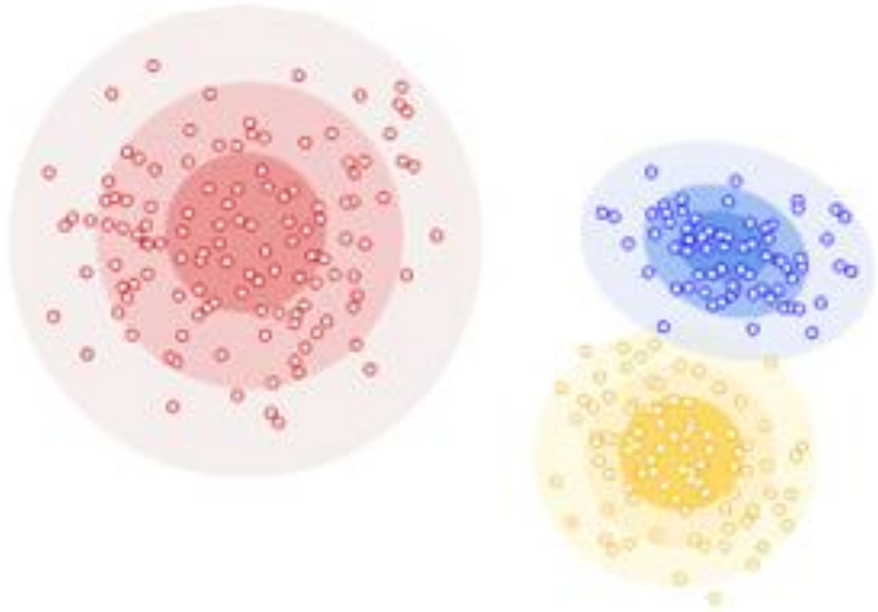
- considers density ahead of distance.
- data is clustered by regions of high concentrations of data objects bounded by areas of low concentrations of data objects.
- density-based clustering connects areas of high density into clusters.
- this allows for arbitrary-shaped distributions as long as dense areas can be connected.
- the clusters formed are grouped as a maximal set of connected data points.
- **algorithm Issues:**
  - difficulty with data of varying densities and high dimensions.
  - do not assign outliers to clusters.





2.4. Distribution-Based Clustering

- assumes data is composed of distributions (such as Gaussian distributions)
- distribution-based algorithm clusters data into distributions
- as distance from the distribution's center increases, the probability that a point belongs to the distribution decreases.
- bands show that decrease in probability.
- use only when you know the type of distribution in your data



2.5. Fuzzy Clustering

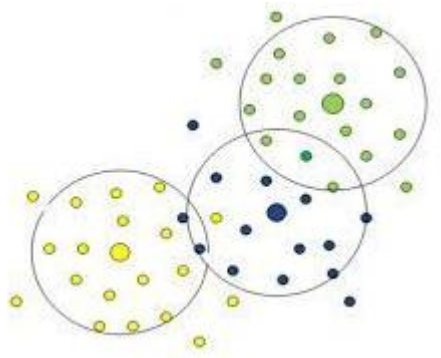
- also called Fuzzy C-Means clustering
- each data point belongs to a cluster to a degree that is specified by a membership value
- algorithm starts with an initial guess for the cluster centers (mean location of each cluster)
- the initial guess for these cluster centers is most likely incorrect.
- assigns every data point a membership grade for each cluster.
- iterations:
  - updates cluster centers and the membership grades for each data point
  - moves the cluster centers to the optimal location within a data set.
  - based on minimizing an objective function that represents the distance from any given data point to a cluster center weighted by the data point membership degree

- FCM algorithm minimizes the following objective function:

$$J_m = \sum_{i=1}^{C} \sum_{j=1}^{N} \mu_{ij}^{m} D_{ij}^{2}$$

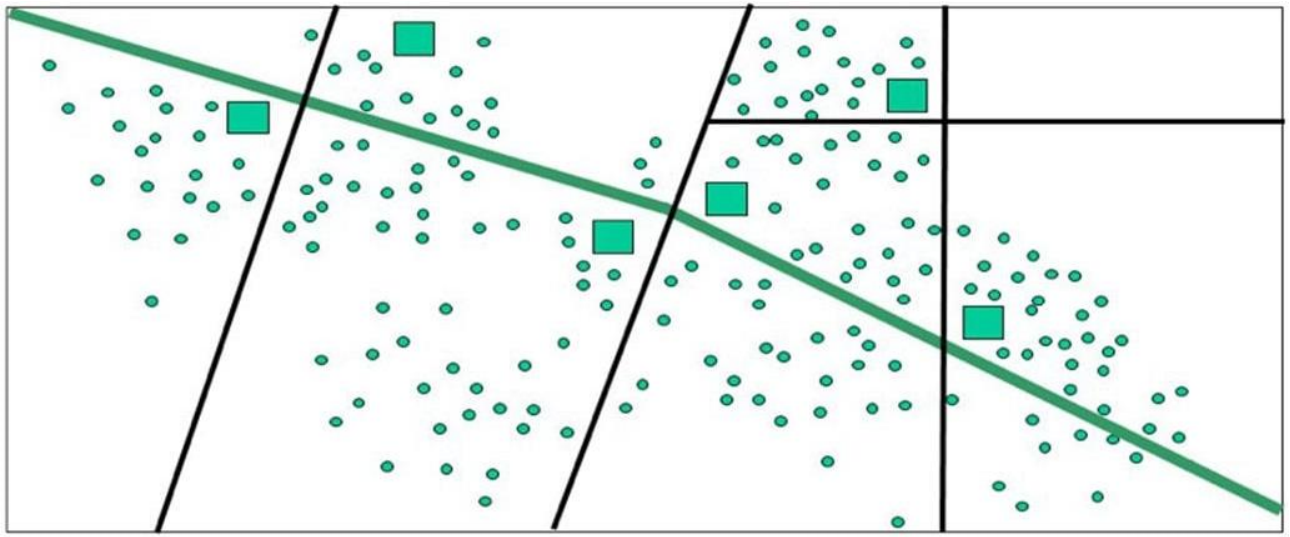
where:

- m is the fuzzy partition matrix exponent for controlling the degree of fuzzy overlap, with $m > 1$. Fuzzy overlap refers to how fuzzy the boundaries between clusters are, that is, the number of data points that have significant membership in more than one cluster.
- D_{ij} is the distance from the jth data point to the ith cluster.
- μ_{ij} is the degree of membership of the jth data point in the ith cluster. For a given data point, the sum of the membership values for all clusters is one.



2.6. Constraint-Based (Supervised) Clustering

- based on the approach that the data can be divided into an optimal number of "unknown" groups
- underlying stages of all the clustering algorithms are to find those hidden patterns and similarities without intervention or predefined conditions
- in certain business scenarios, we might be required to partition the data based on certain constraints (supervision)
- a constraint is defined as:
  - the desired properties of the clustering results or a user's expectation of the clusters so formed
- desired properties can be in terms of:
  - a fixed number of clusters
  - the cluster size
  - important dimensions (variables)



Example:
Clustering by ATM location:
→ Less parameters
→ more user desired constraints

3. The K-Means Clustering Algorithm

- is a partition-based clustering technique
- uses the Euclidean distances between the points as a criterion for cluster formation
- assume there are “n” numbers of data objects:
  - k-means groups them into a predetermined “k” number of clusters
- each cluster has a cluster center allocated and each of them is placed at farther distances.
- every incoming data point gets placed in the cluster with the closest cluster center.
- this process is repeated until all the data points get assigned to any cluster.
- once all the data points are covered the cluster centers or centroids are recalculated.
- after having these “k” new centroids, a new grouping is done between the nearest new centroid and the same data set points. iteratively, there may be a change in the k centroid values and their location
- this loop continues until the cluster centers do not change or in other words, centroids do not move anymore.

K-Means Clustering (cont.)

The algorithm minimizes the following objective function:

$$J = \sum_{i=1}^{n}\sum_{j=1}^{k} w_{ij}\|x_i - \mu_j\|^2$$

where w_{ij}=1 for data point x_i if it belongs to cluster j; otherwise, w_{ij}=0. Also, μ_j is the centroid of x_i's cluster.

It's a minimization problem of two parts:

A) minimize J w.r.t. w_{ij} and treat μ_j fixed.

B) minimize J w.r.t. μ_j and treat w_{ij} fixed.

A-step:

$$\frac{\partial J}{\partial w_{ij}} = \sum_{i=1}^{n}\sum_{j=1}^{k}\|x_i - \mu_j\|^2 = 0$$

$$\rightarrow w_{ij} = \begin{cases} 1 & \text{if } j = \text{argmin}_j \|x_i - \mu_j\|^2 \\ 0 & otherwise \end{cases}$$

→ point x_i assigned to the closest cluster

B-step:

$$\frac{\partial J}{\partial \mu_j} = 2\sum_{i=1}^{n} w_{ij}(x_i - \mu_j) = 0$$

$$\rightarrow \mu_j = \frac{\sum_{i=1}^{n} w_{ij} x_i}{\sum_{i=1}^{n} w_{ij}}$$

→ cluster centroids recomputed

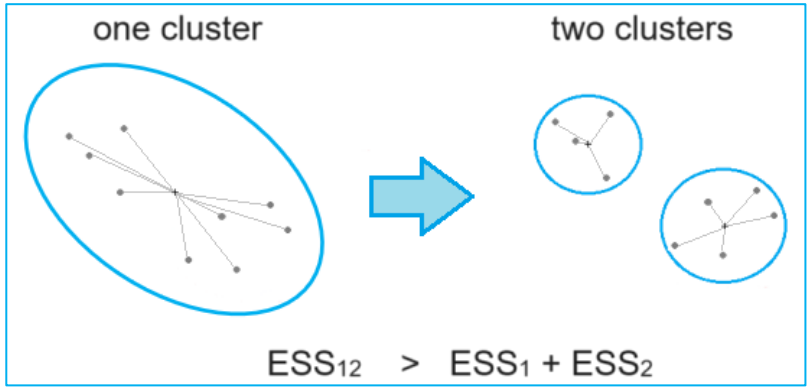
K-Means Clustering (cont.)

Things to note:

- standardize the data to have a mean of zero and a standard deviation of one
- run the algorithm several times:
  - using different initializations of centroids
  - pick the results of the run that yielded the lower sum of squared distance.

K-Means Clustering (cont.)

How to find the optimal number of clusters?

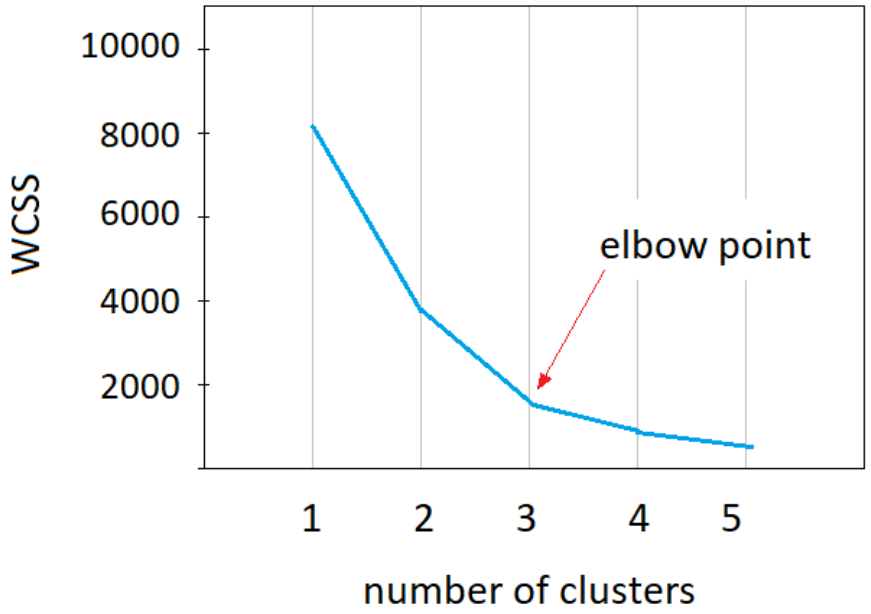


Several methods:

- average silhouette method
- gap statistic method
- elbow method.

Elbow method:

- varies the number of clusters (K) from 1 – 10, for example.
- for each value of K, calculates WCSS (Within-Cluster Sum of Square).
- WCSS is the sum of the squared distance between each point and the centroid in a cluster.
- plot of WCSS vs. K value, looks like an elbow.
- there is a point:
  - at which the graph will rapidly change and thus create an elbow shape.
  - after which the graph moves almost parallel to the X-axis.
- K value corresponding to this point is the optimal value of K



K-Means Clustering (Coding Example)

Code:

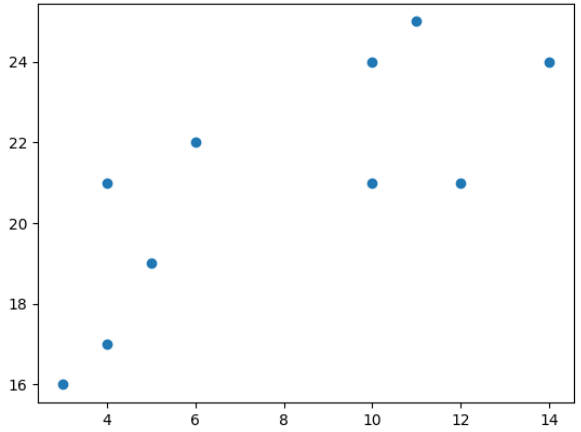
Start by hardcoding and visualizing some data points:

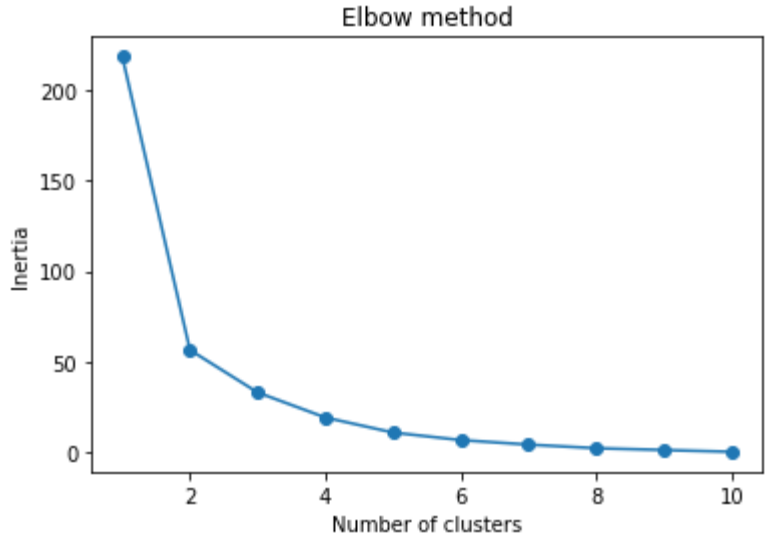
```
import matplotlib.pyplot as plt
x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
plt.figure(figsize = (5,4))
plt.scatter(x, y)
plt.show(block=False)
```

Run the algorithm for different numbers of clusters:

```
from sklearn.cluster import KMeans
data = list(zip(x, y))
inertias = []
for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)
plt.figure(figsize = (5,4))
plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show(block=False)
```

Output:





The elbow method shows that 2 is a good value for K

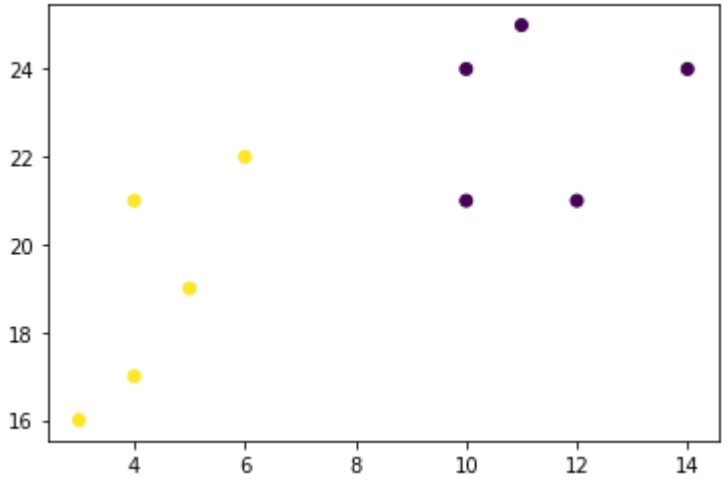
K-Means Clustering (Coding Example)

Code:

Visualizing the results for k = 2:

```
kmeans = KMeans(n_clusters=2)
kmeans.fit(data)
plt.figure(figsize = (5,4))
plt.scatter(x, y, c=kmeans.labels_)
plt.show(block=False)
```

Output:



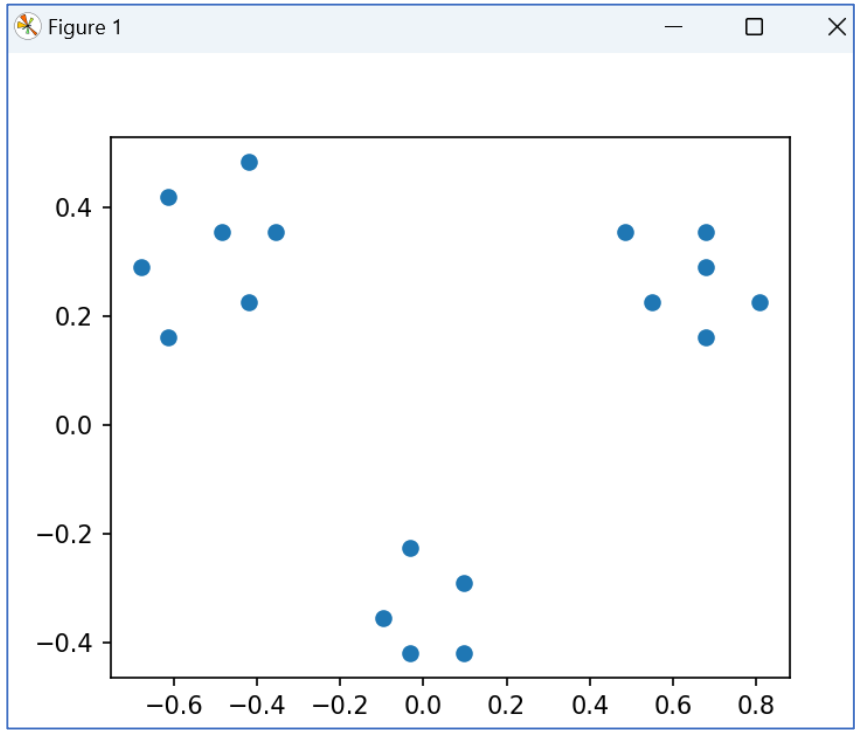
Another example with more data
Here we will automatically find the optimal number of clusters.

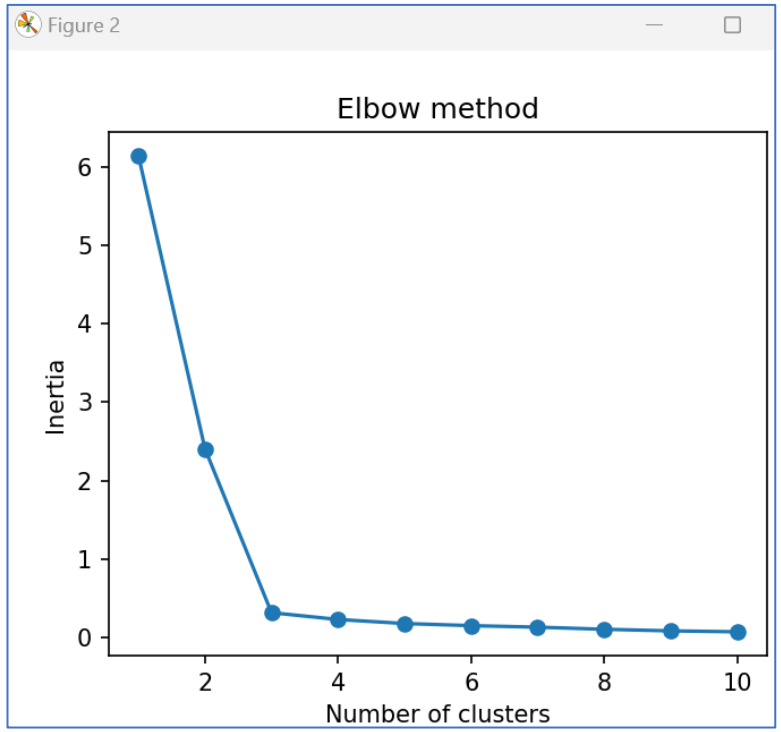
Code:

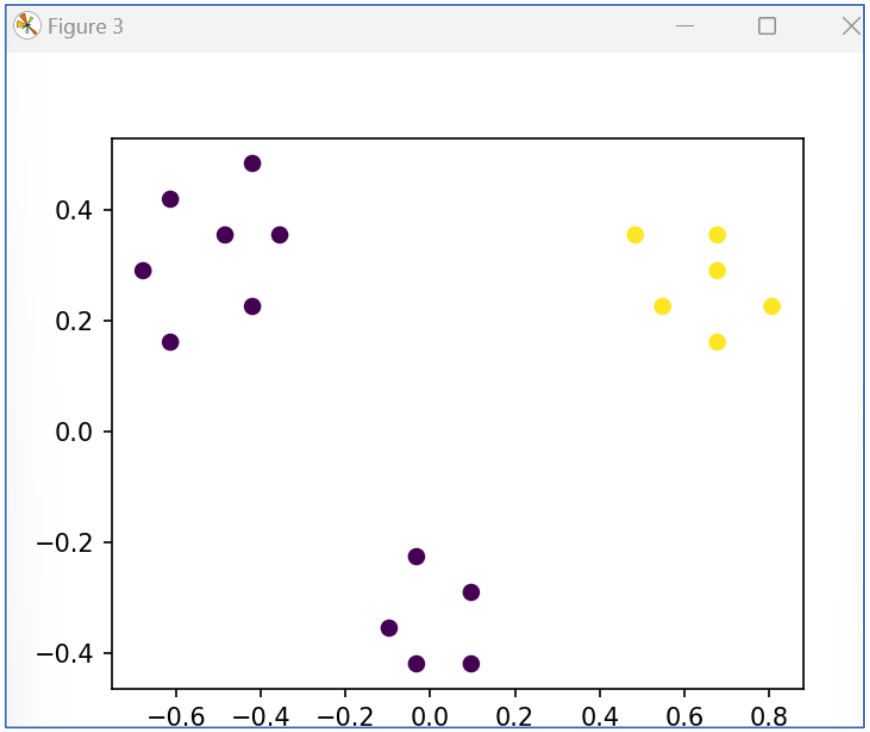
same as before, but now replace x and y with:

```
x = [-0.41935484,-0.61290323,-0.48387097,-0.35483871,0.48387097,0.67741935,-0.67741935,0.67741935,-0.41935484,0.5483871,0.80645161,-0.61290323,0.67741935,-0.03225806,0.09677419,-0.09677419,-0.03225806,0.09677419]
y = [0.48387097,0.41935484,0.35483871,0.35483871,0.35483871,0.35483871,0.29032258,0.29032258,0.22580645,0.22580645,0.22580645,0.16129032,0.16129032,-0.22580645,-0.29032258,-0.35483871,-0.41935484,-0.41935484]
```

Outputs:

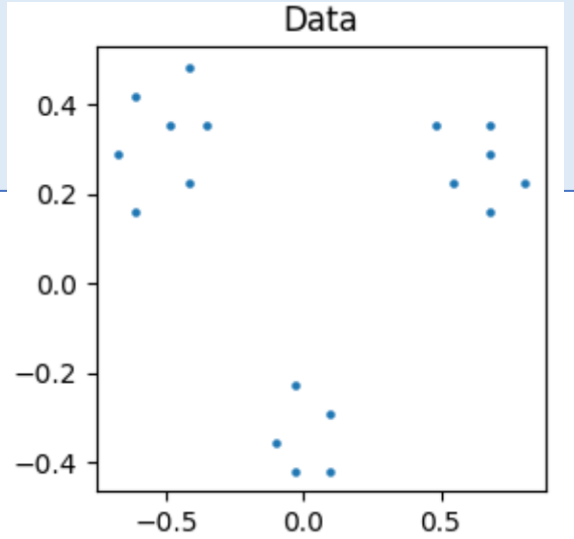






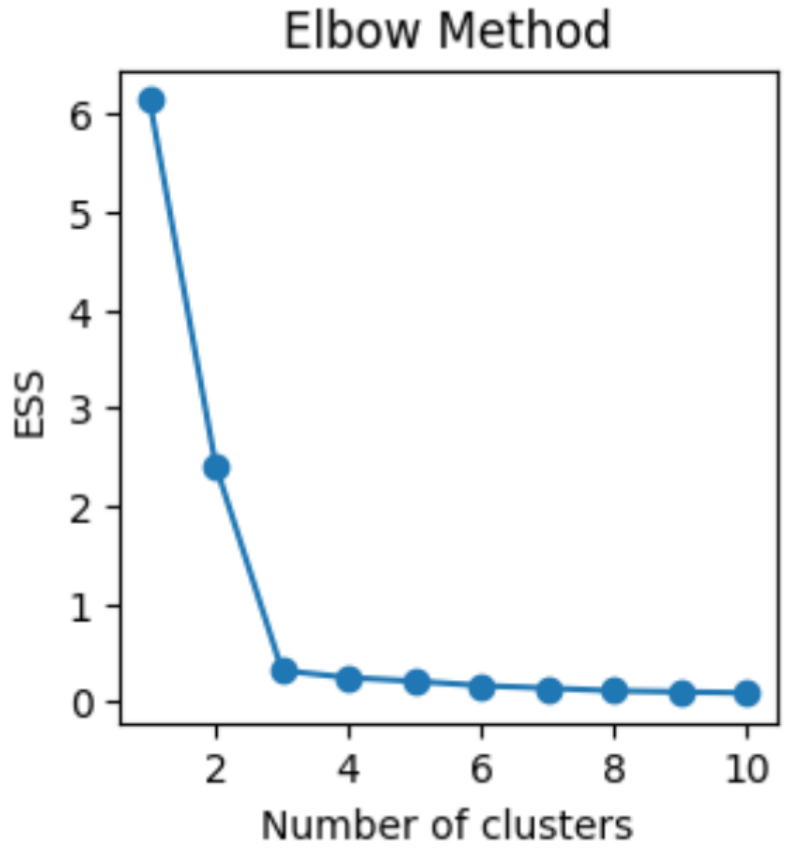
Visualizing the data:
Add this to your code:

```
# Adding more data

import matplotlib.pyplot as plt
# x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
# y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

x = [-0.41935484,-0.61290323,-0.48387097,-0.35483871,0.48387097,0.67741935,-0.67741935,0.67741935,-
0.41935484,0.5483871,0.80645161,-0.61290323,0.67741935,-0.03225806,0.09677419,-0.09677419,-0.03225806,0.09677419]
y =
[0.48387097,0.41935484,0.35483871,0.35483871,0.35483871,0.35483871,0.29032258,0.29032258,0.22580645,0.22580645,0.22580645,0.1
6129032,0.16129032,-0.22580645,-0.29032258,-0.35483871,-0.41935484,-0.41935484]

plt.figure(figsize = (3,3))
plt.title("Data")
plt.scatter(x, y, s=5)
plt.show()
```



Visualizing the ESS by number of clusters:
Add this to your code:

```
# Computing the ESS per number of clusters

from sklearn.cluster import KMeans
data = list(zip(x, y))
inertias = []
for i in range(1,11):
    kmeans = KMeans(n_clusters=i, random_state = 0, n_init='auto')
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)
plt.figure(figsize = (3,3))
plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()
```



An automatic elbow detection method:

- seek for the smallest value of:

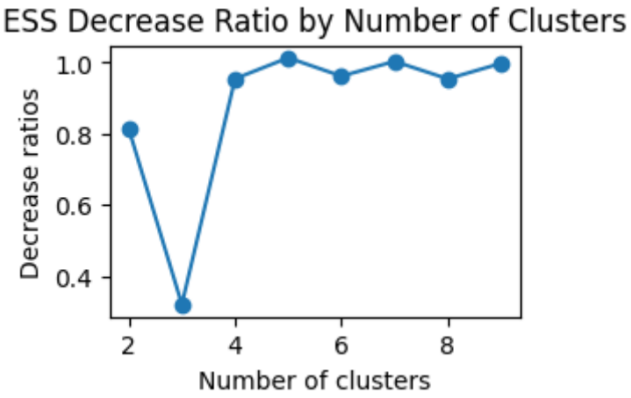
$$DecreaseRatio = \frac{\Delta(i)}{\frac{1}{3}[\Delta(i-1) + \Delta(i) + \Delta(i+1)]}$$

where D(i) = ESS(i) – ESS(i+1)

Applying the ratio technique to find the optimal number of clusters:
Add this to your code:

```
# Appling the ratio technique to find the optimal number of clusters

from sklearn.cluster import KMeans
data =  list(zip(x, y))
inertias = []
ratios = []
for  i  in  range(1,11):
      kmeans = KMeans(n_clusters=i, random_state = 0, n_init='auto')
      kmeans.fit(data)
      inertias.append(kmeans.inertia_)

ess = inertias
ratios = []
for i in range(len(ess) - 1):
   if i == 0:
      continue
   ratio =  3*ess[i] / (ess[i-1] + ess[i] + ess[i+1])
   ratios.append(ratio)
```

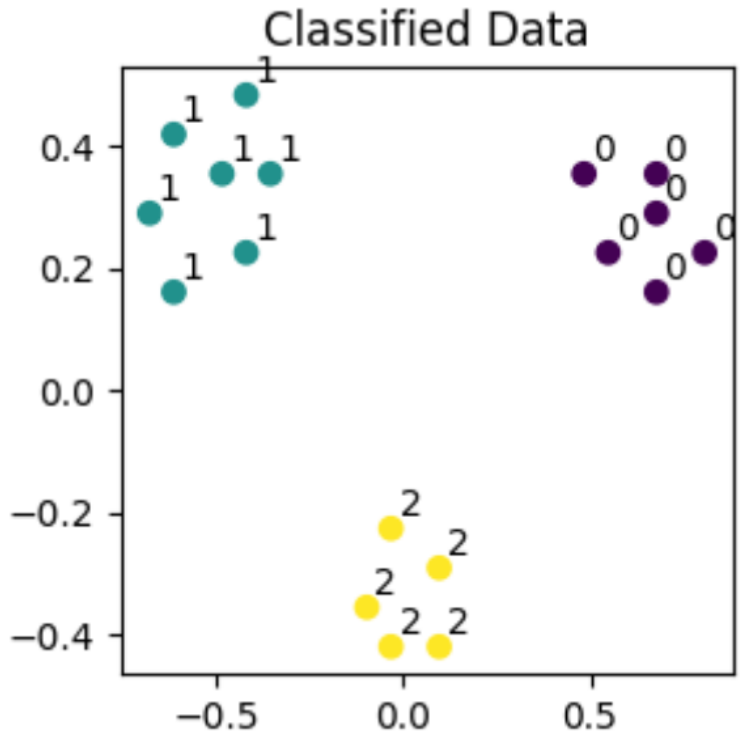
```
np_ratios = np.array(ratios)
# plot the ratios
plt.figure(figsize = (3,2))
plt.plot(range(1+1,11-1), np_ratios, marker='o')
plt.title('ESS Decrease Ratio by Number of Clusters')
plt.xlabel('Number of clusters')
plt.ylabel('Decrease ratios')
plt.show()

print("ESS values for i-1, 2, 3... clusters:", ratios)
print("The optimal number of clusters is {}, which yields the lowest
ESS".format(np.argsort(ratios)[0] + 2))
```




```
ESS values for i-1, 2, 3... clusters: [0.8142179976330783, 0.32075975321321837, 0.9545454524633, 1.0134680130399618, 0.96195652
33405897, 1.0033613372985126, 0.9542743610077112, 0.9976905310786234]
The optimal number of clusters is 3, which yields the lowest ESS
```

Getting the labels and adding colors to the plot.
Add this to your code:

```
# Now we find the labels and add colors to the plot by running the KMeans algorithm again
# for the maximum number of clusters just found, which was 3.

kmeans = KMeans(n_clusters=3, random_state = 0, n_init='auto')
kmeans.fit(data)
# labels = hierarchical_cluster.fit_predict(data)
labels = kmeans.fit_predict(data)

plt.figure(figsize = (3,3))
plt.title("Classified Data")
plt.scatter(x, y, c=kmeans.labels_)
# Annotate each point
for i, label in enumerate(labels):
    plt.annotate(label, (x[i] + 0.02, y[i] + 0.02))
plt.show()

print(labels)
```




```
[1 1 1 1 0 0 1 0 1 0 0 1 0 2 2 2 2 2]
```

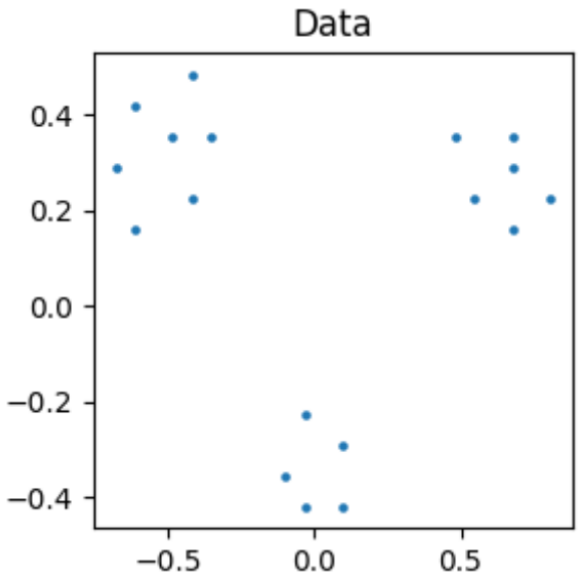
Show the list of points and their membership values:

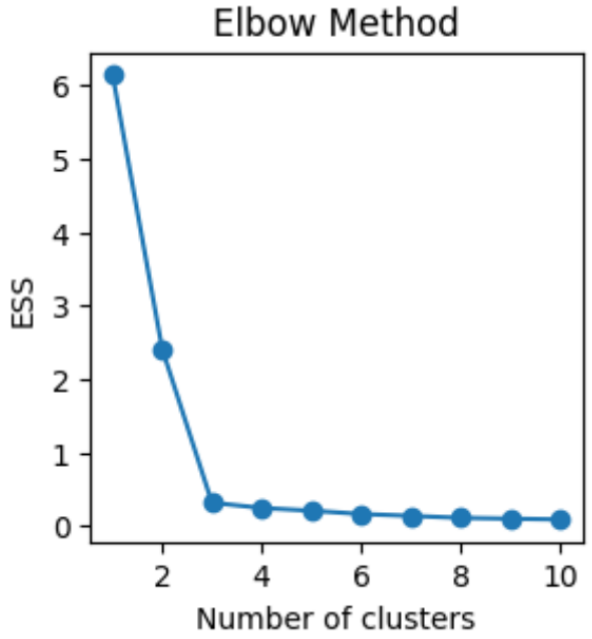
```
# Show the list of points and their membership values
print("    Data Points and Memberships")
print("     X       Y    Membership")
print("  _____________________________")
for i in range(len(x)):
    print("{:>8.2}  {:>8.2}  {:>3}".format(x[i], y[i], labels[i]))
```

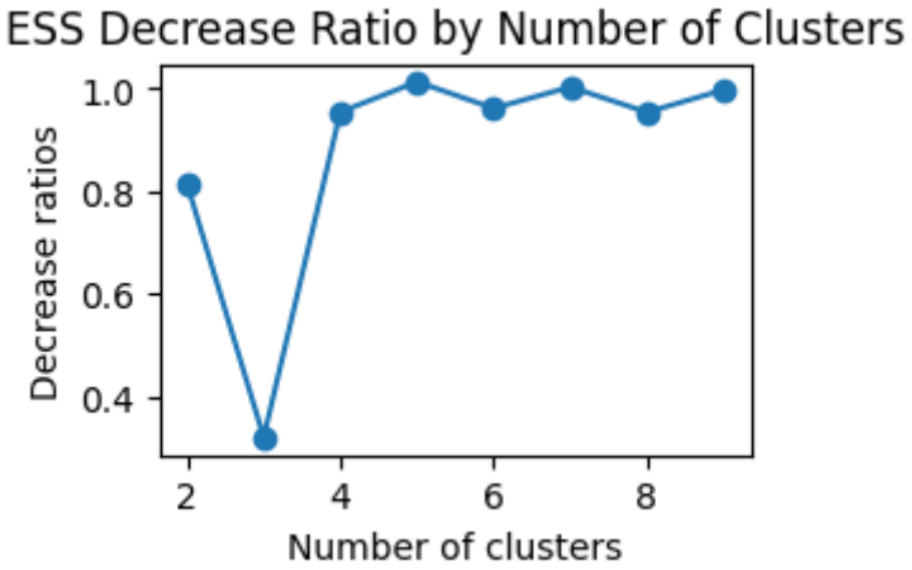
```
  Data Points and Memberships
   X          Y     Membership
 ____________________________
 -0.42       0.48     1
 -0.61       0.42     1
 -0.48       0.35     1
 -0.35       0.35     1
  0.48       0.35     0
  0.68       0.35     0
 -0.68       0.29     1
  0.68       0.29     0
 -0.42       0.23     1
  0.55       0.23     0
  0.81       0.23     0
 -0.61       0.16     1
  0.68       0.16     0
-0.032      -0.23     2
 0.097      -0.29     2
-0.097      -0.35     2
-0.032      -0.42     2
 0.097      -0.42     2
```

$$DecreaseRatio = \frac{\Delta(i)}{\frac{1}{3}[\Delta(i-1) + \Delta(i) + \Delta(i+1)]}$$

with D(i) = ESS(i) – ESS(i+1)







End of Lecture