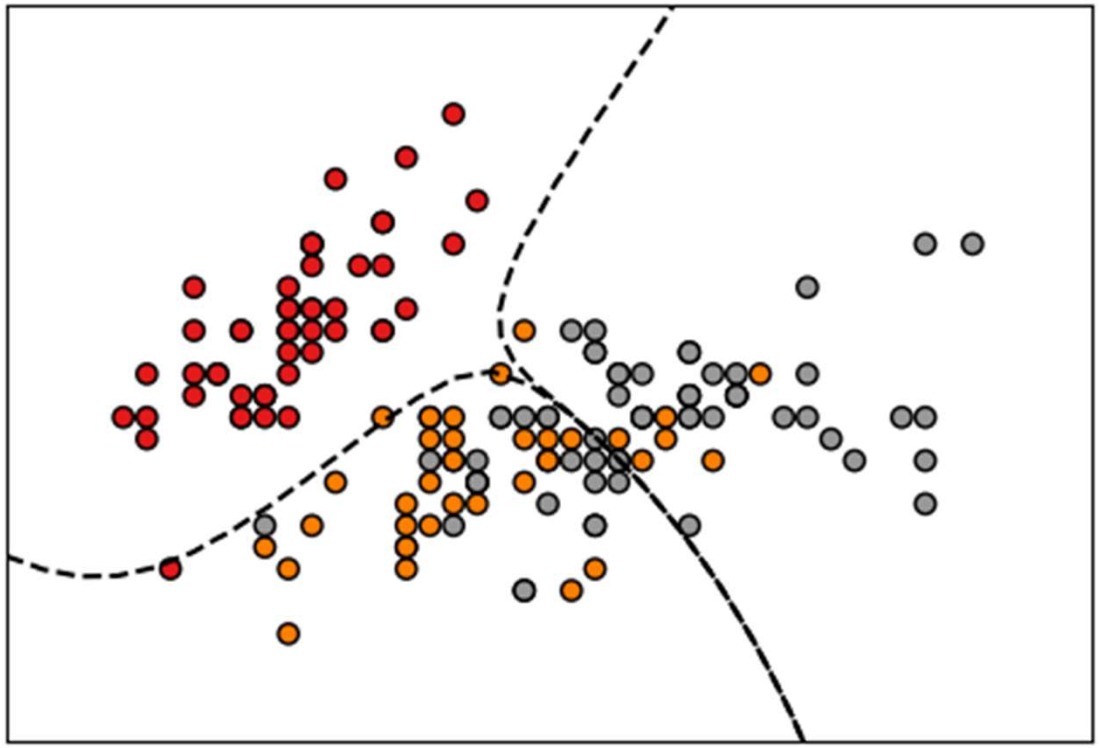


Chapter 8 Cluster analysis

CSS 341 Introduction to Data Science

Chukiat Worasucheep

Important Notice

การเรียนการสอนหัวข้อนี้ ผ่านทางสื่อออนไลน์ (Online meeting)

และมีการบันทึกภาพและเสียงเพื่อประโยชน์ทางการศึกษาต่อไปในอนาคต.

หากท่านไม่ยินยอมให้มีการเผยแพร่การบันทึกดังกล่าว ขอให้แจ้งให้ผู้สอนทราบภายใน 36 ชั่วโมง.

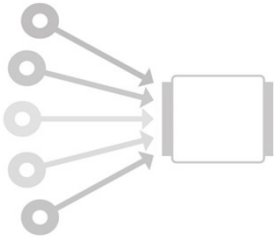
Learning objectives

- เข้าใจแนวคิดของปัญหาการจัดกลุ่ม (Clustering problem)
- เข้าใจหลักการเบื้องต้นของขั้นตอนวิธี k-Means และ Agglomerative Clustering สำหรับปัญหาการจัดกลุ่ม
- หัดใช้ Pythonในการแก้ปัญหาการจัดกลุ่มด้วยวิธี k-Means และ Agglomerative Clustering
- เข้าใจผลลัพธ์ที่ได้จากการแก้ปัญหาการจัดกลุ่มด้วย Python
- ทดลองประยุกต์ใช้ Python เพื่อจัดกลุ่มลูกค้า (customer segmentation)

Outline

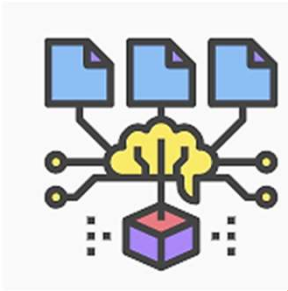
- Concept of cluster analysis
- K-mean clustering
- An application to customer segmentation
- Agglomerative clustering
- DBSCAN *(optional)*

Data science process



Data acquisition 02

Data Preparation 04



06 Data modeling

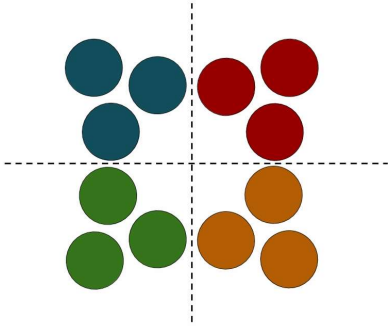
07 Deployment and maintenance

05 Data visualization

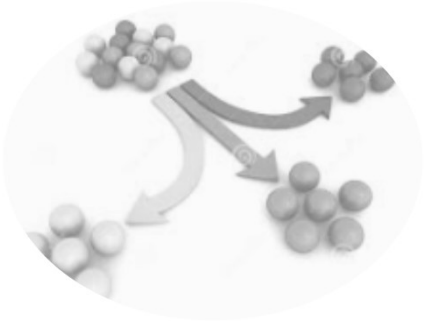
03 Data Exploration

01 Understand business and problems

Major modeling techniques for data science

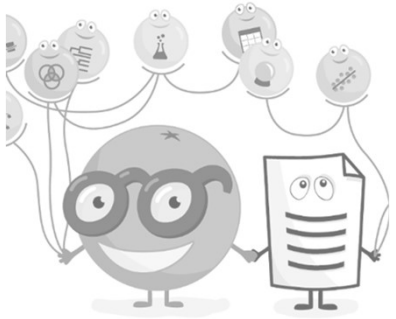


Clustering
การจัดกลุ่ม

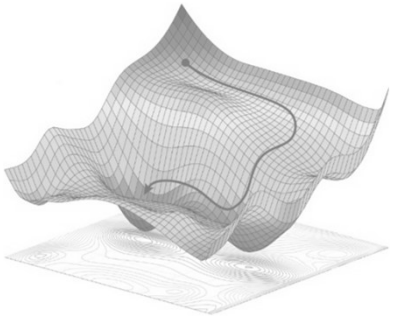


Classification
การจำแนกประเภท

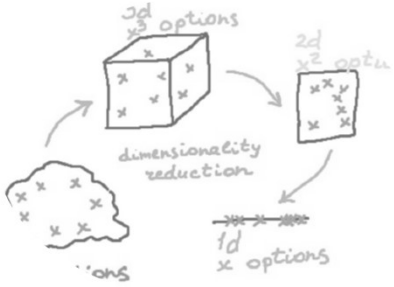
Regression
การวิเคราะห์ถดถอย



Association rule mining
การค้นหากฎความสัมพันธ์



Optimization
การหาค่าเหมาะที่สุด



Dimensionality reduction
การลดมิติข้อมูล

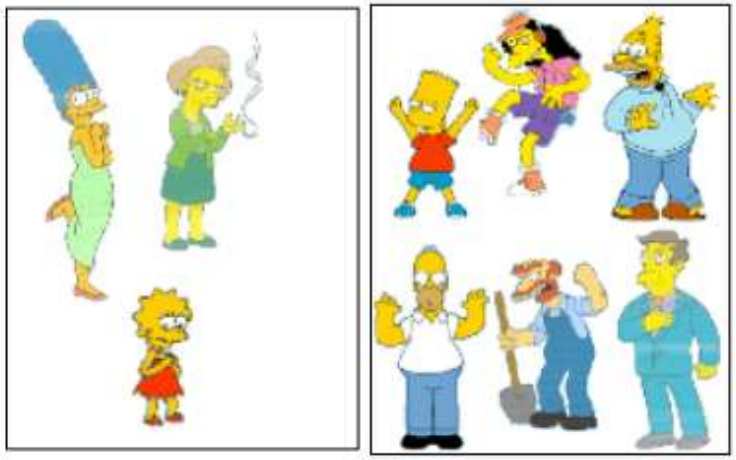
ขั้นตอนวิธีการจัดกลุ่ม (Clustering algorithm)

- Clustering เป็นการจัดกลุ่มของข้อมูลจำนวนหนึ่งออกเป็นกลุ่มย่อย ๆ โดยที่...
  - สมาชิกในแต่ละกลุ่มย่อยควรจะมีลักษณะคล้ายกันมาก ๆ และ ...
  - แตกต่างจากกลุ่มอื่น
- จัดเป็น *Unsupervised learning* คือไม่มีการฝึกสอน (training) แล้วค่อยนำ model ไปใช้งาน
- แต่ใช้หลักการทางคณิตศาสตร์และสถิติมาคำนวณเพื่อจัดกลุ่ม

- ลองแบ่งกลุ่ม...

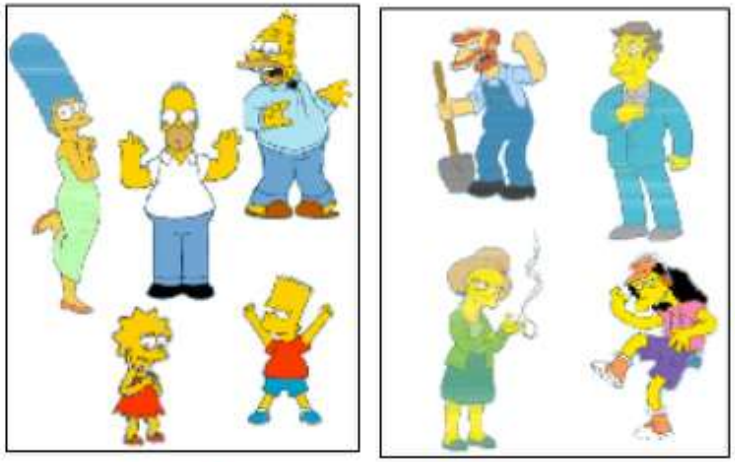
How do you group them?

GENDER



Female Male

ROLE



Simpson's Family School Employees

Cluster analysis

- After applying the clustering technique, each cluster or group is provided with a *cluster-ID,* which can be used for further analysis.
- The clustering technique is commonly used for statistical data analysis.

- *Important Note:* Clustering is somewhere similar to the classification algorithm, but the difference is the type of dataset that we are using.
  - In classification, we work with the *labeled* data set,
  - whereas in clustering, we work with the *unlabeled* dataset.

How to do cluster analysis?

- Organizing data into classes:
  - high intra-class (within class) similarity
  - low inter-class (between classes) similarity

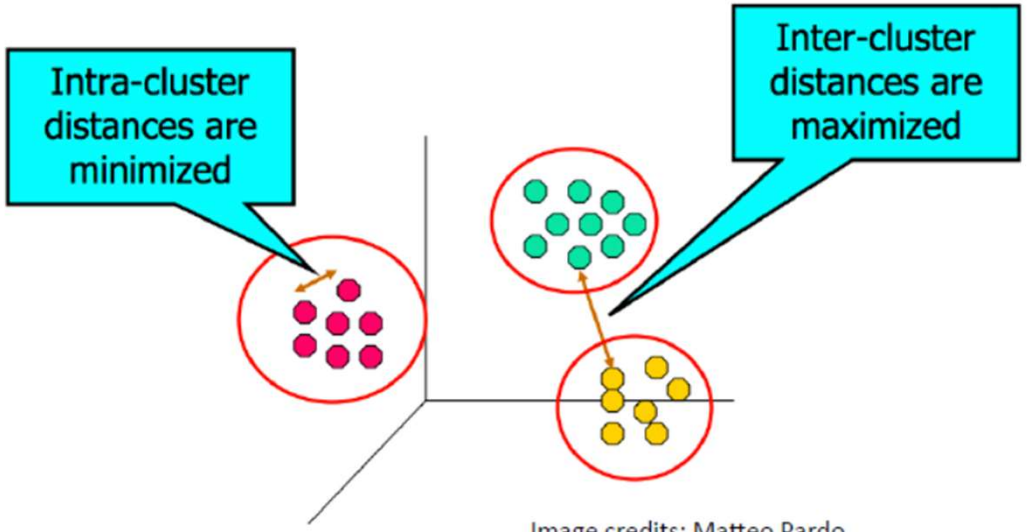
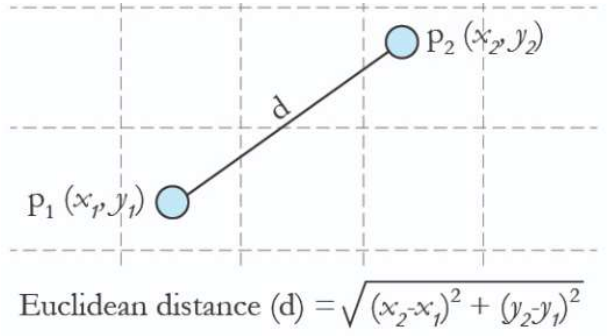


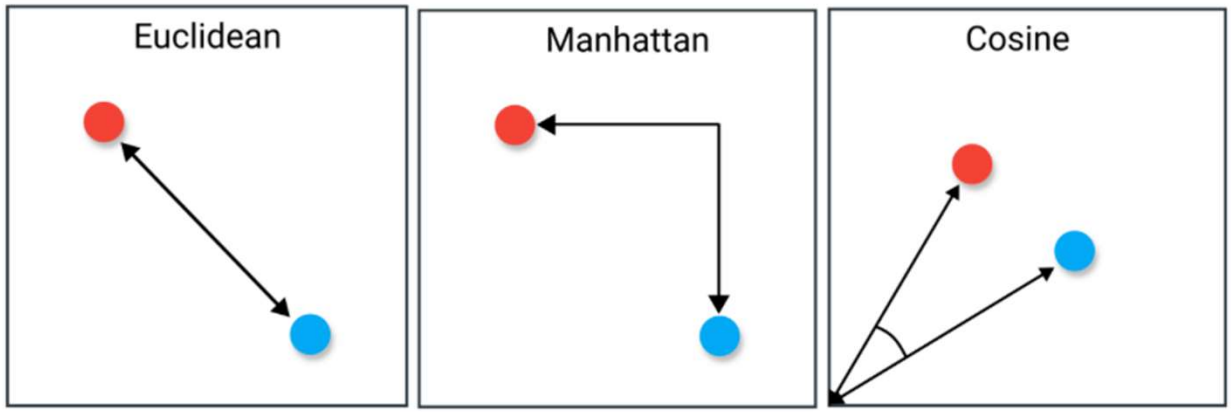
Image credits: Matteo Pardo

- What is similarity?

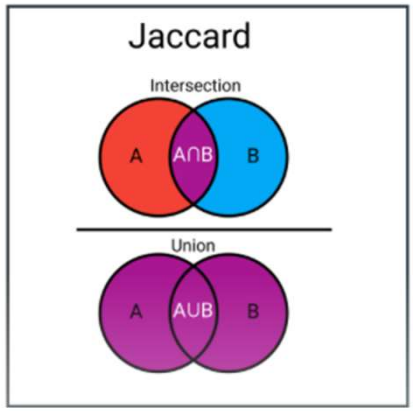
Foundation of cluster analysis

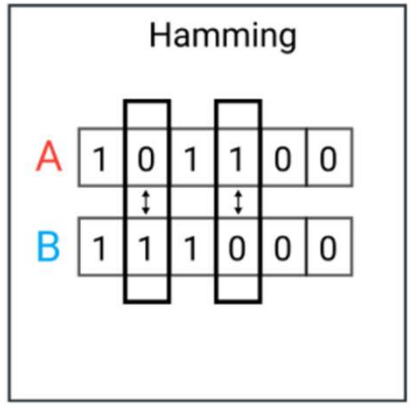
- Distance (for quantitative data)
  - Euclidean distance





- Similarity (for qualitative data)





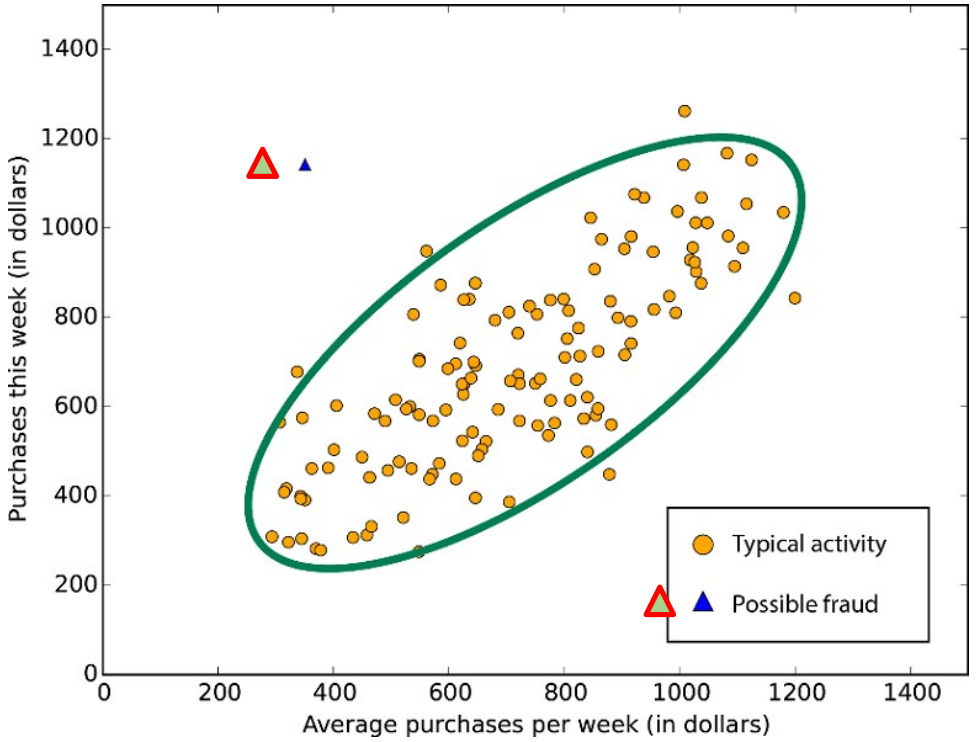
$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Hamming distance is the number of positions at which the corresponding items are different.

Applications of cluster analysis

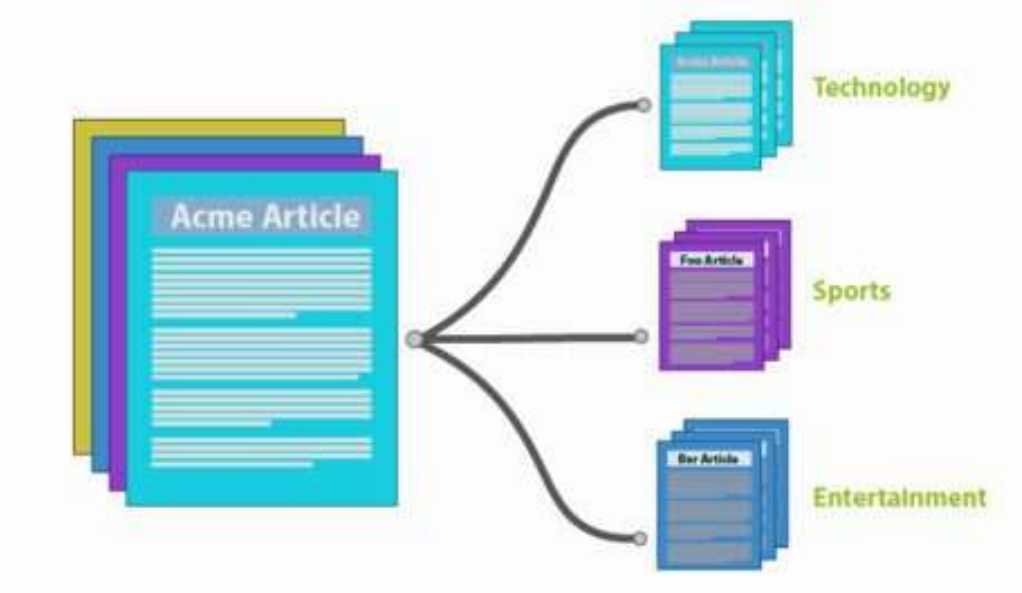
- Customer/market segmentation
  - based on customer profile, purchase history, interests, or activity monitoring.
  - for marketing and sales.

- Identify fraudulence (*anomaly detection*)

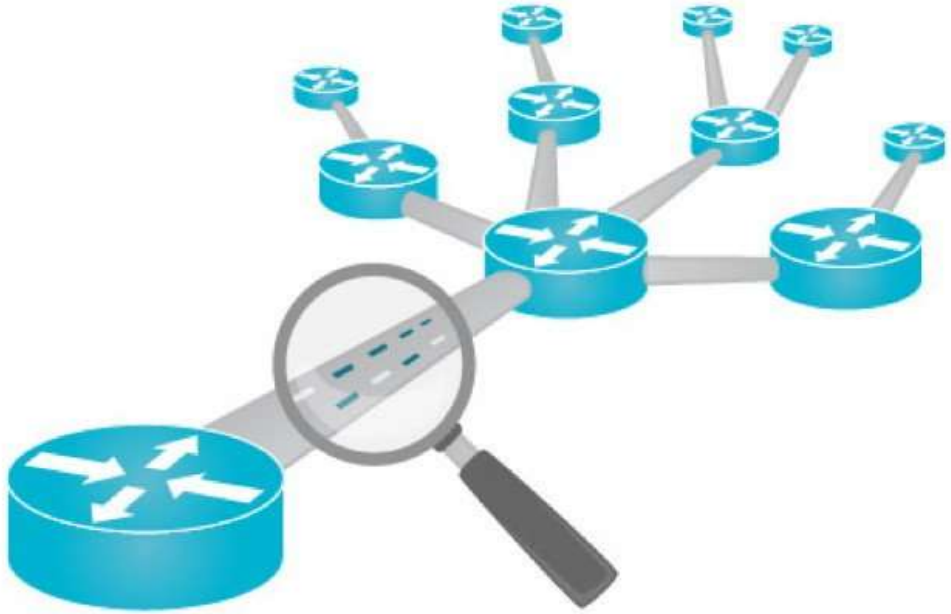


Applications of cluster analysis

- Document filtering
  - Contents

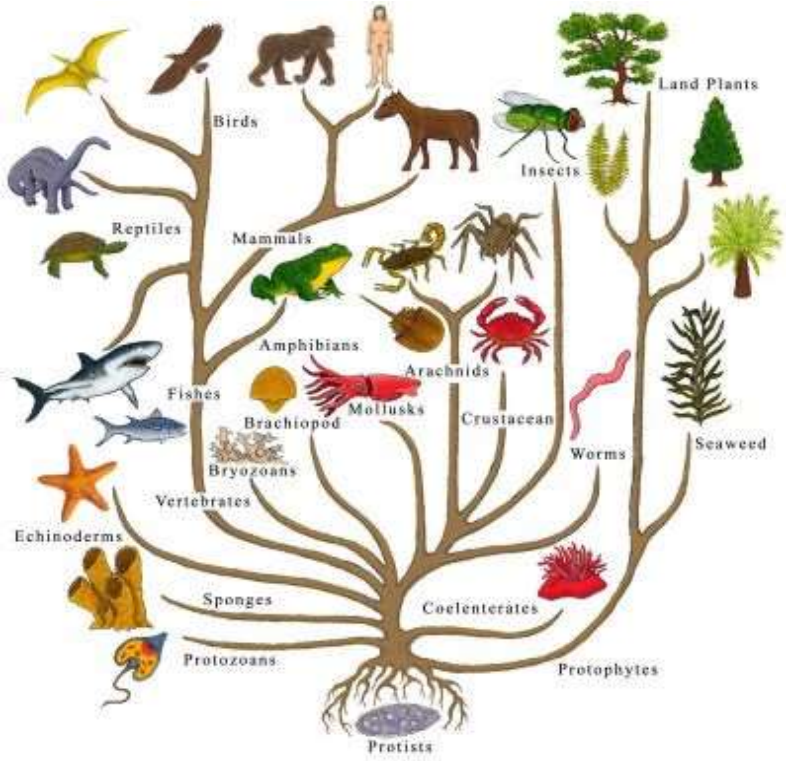


- Classify network traffic into a web site
  - Normal or from bots?



Applications of cluster analysis

- Life taxonomy
  - To derive plant and animal taxonomies, or to categorize genes with similar functionalities.



- Search Engines
  - To group similar data objects in one group that is far from the other dissimilar objects. The accurate result of a query depends on the quality of the clustering algorithm used.

Carrot², an open-source search results clustering engine.

Taxonomy of clustering algorithms

- Partitioning-based (centroid-based)
  - K-means (1967)
- Hierarchical-based
  - Agglomerative hierarchical clustering
- Distribution-based
  - Gaussian mixture models (GMM) (1999)
- Density-based
  - DBSCAN (Density-Based Spatial Clustering of Applications with Noise) (1996)
  - Fast-DBSCAN (2016)
- Hybrid
  - HDBSCANS (2017)

The clustering methods are broadly divided into:

- *Hard* clustering (data point belongs to only *one* group).
- *Soft* clustering (each data point can belong to more than group group).

Well-known clustering algorithms

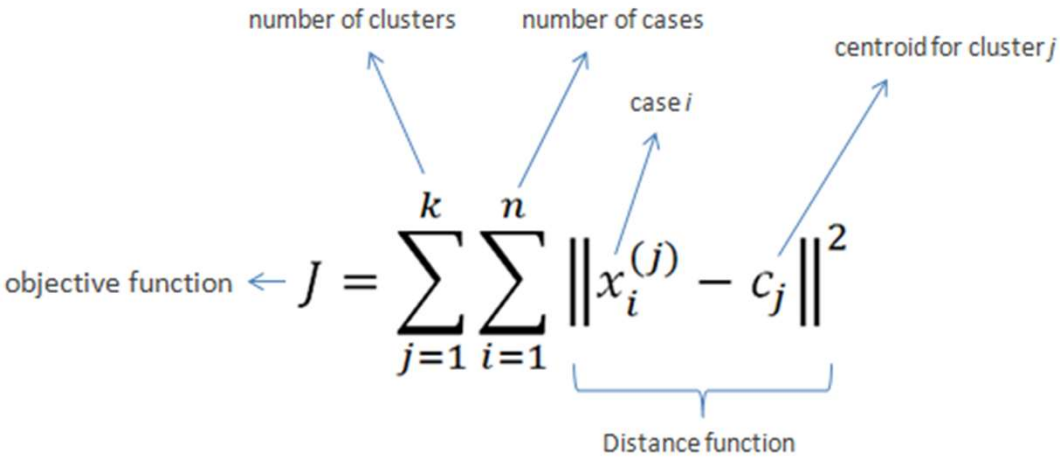
- *K-Means* algorithm classifies the dataset by dividing the samples into different clusters of equal variances. The number of clusters must be specified in this algorithm. It is fast with fewer computations required.
- *Agglomerative hierarchical clustering* algorithm performs the bottom-up hierarchical clustering. Each data point is treated as a single cluster at the outset and then successively merged. The cluster hierarchy can be represented as a tree-structure.
- *DBSCAN* (Density-Based Spatial Clustering of Applications with Noise). is an example of a density-based model with some remarkable strengths. The areas of high density are separated by the areas of low density. Because of this, the clusters can be found in any arbitrary shape.

Outline

- Concept of cluster analysis
- K-mean clustering
- An application to customer segmentation
- Agglomerative clustering
- DBSCAN

K-means Clustering

- An iterative algorithm that tries to partition the dataset into K pre-defined *nonoverlapping clusters* where each data point belongs to only one group.
- The K-means algorithm aims to choose *centroids* that minimize the *W*ithin-*C*luster *S*um of *S*quares (WCSS) criterion.



How does K-means work?

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids.

Step-3: Assign each data point to their closest centroid according to Euclidean distance function, which will form the predefined K clusters.

Step-4: Calculate the centroid or mean of all data points in each cluster and place a new centroid of each cluster.

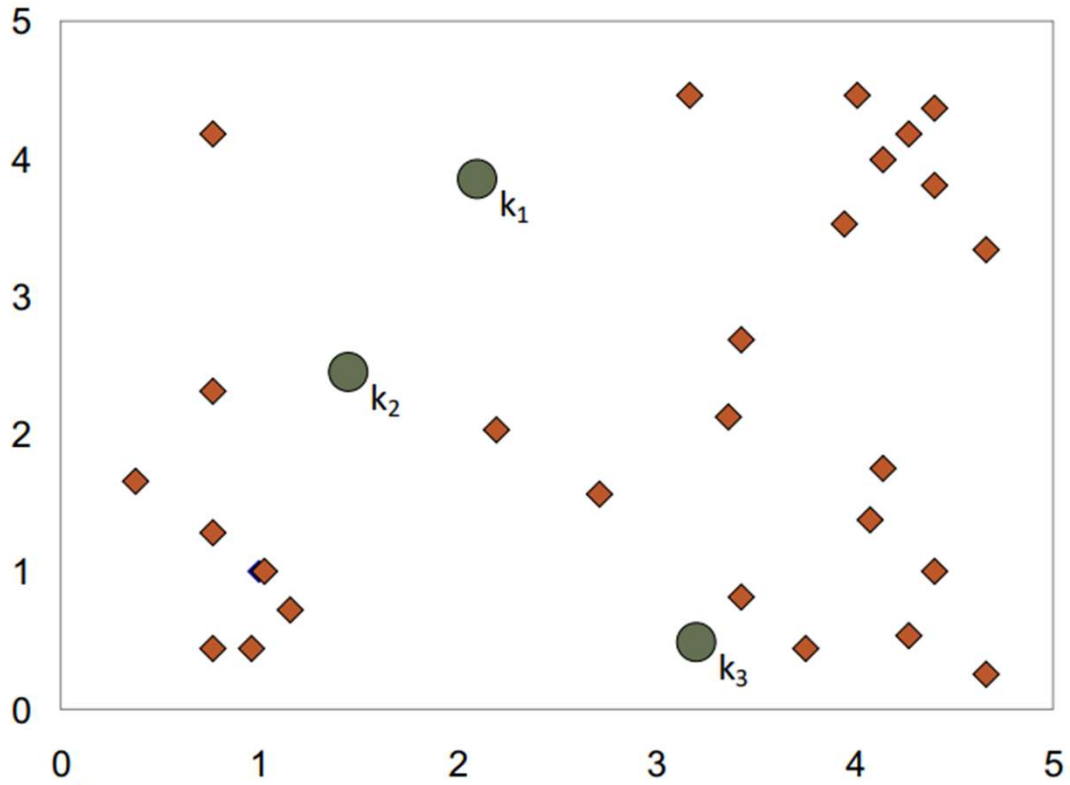
Step-5: Repeat the third (and 4^{th}) steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: The model finishes.

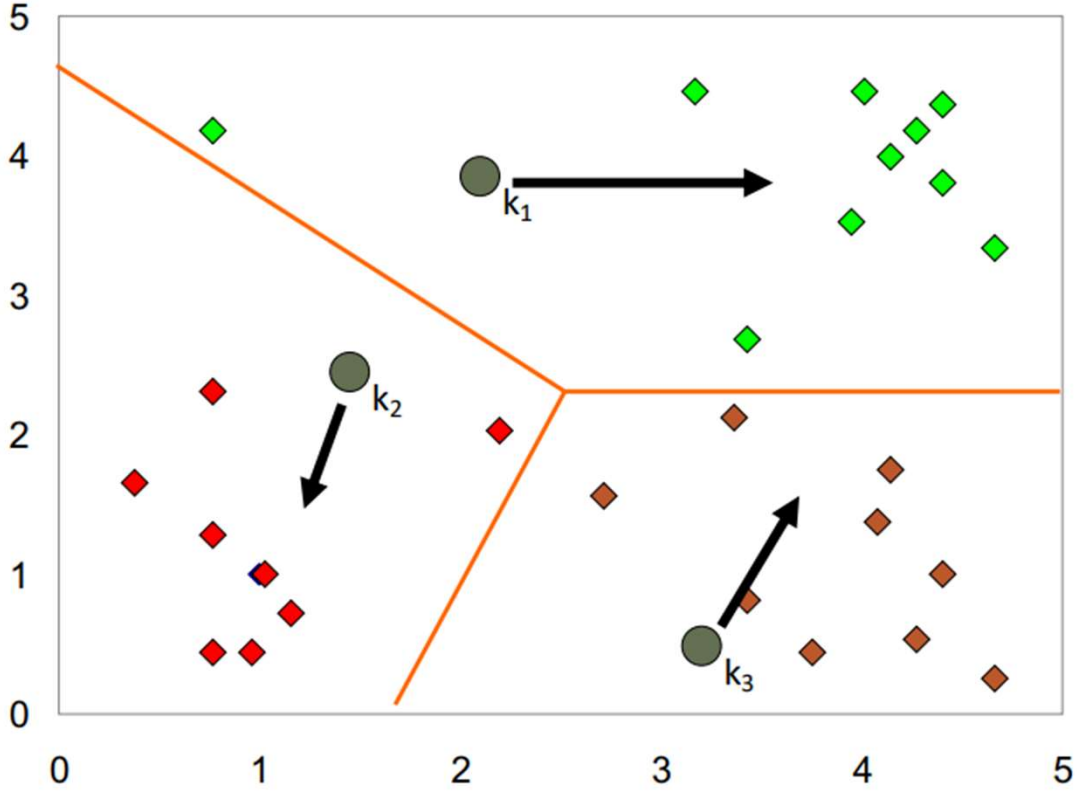
Demonstration of K-means

Step 1

K-means, with K=3
using Euclidean distance

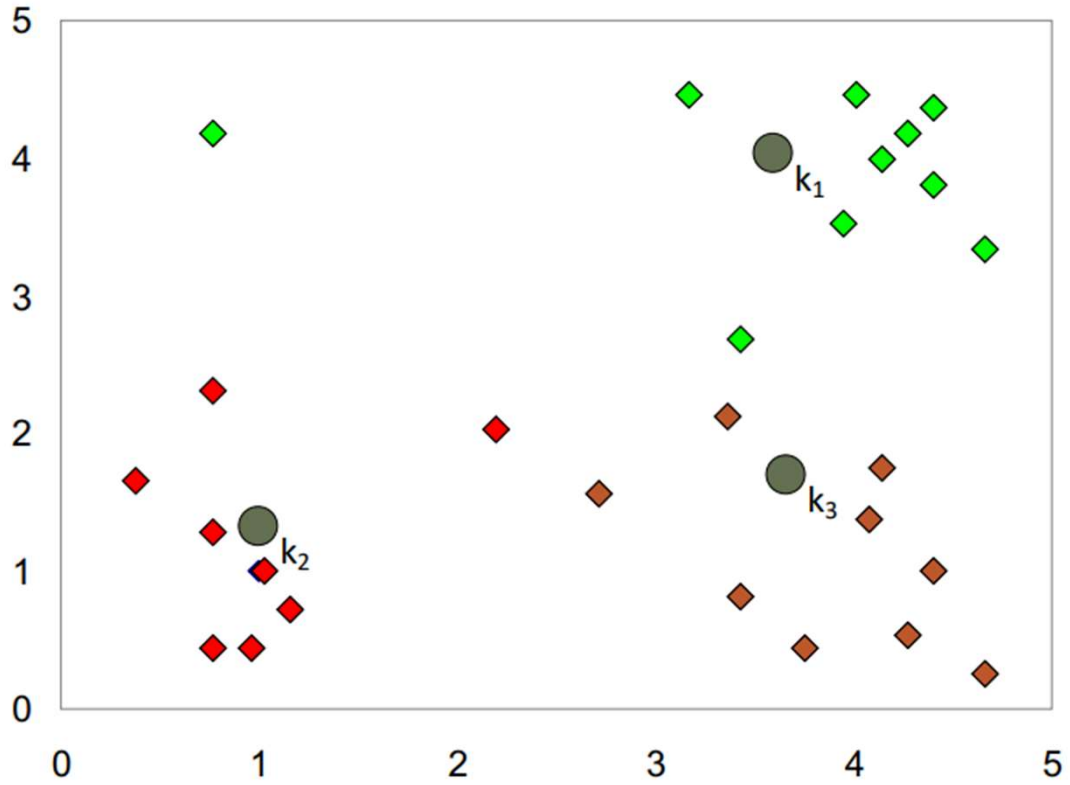


Step 2

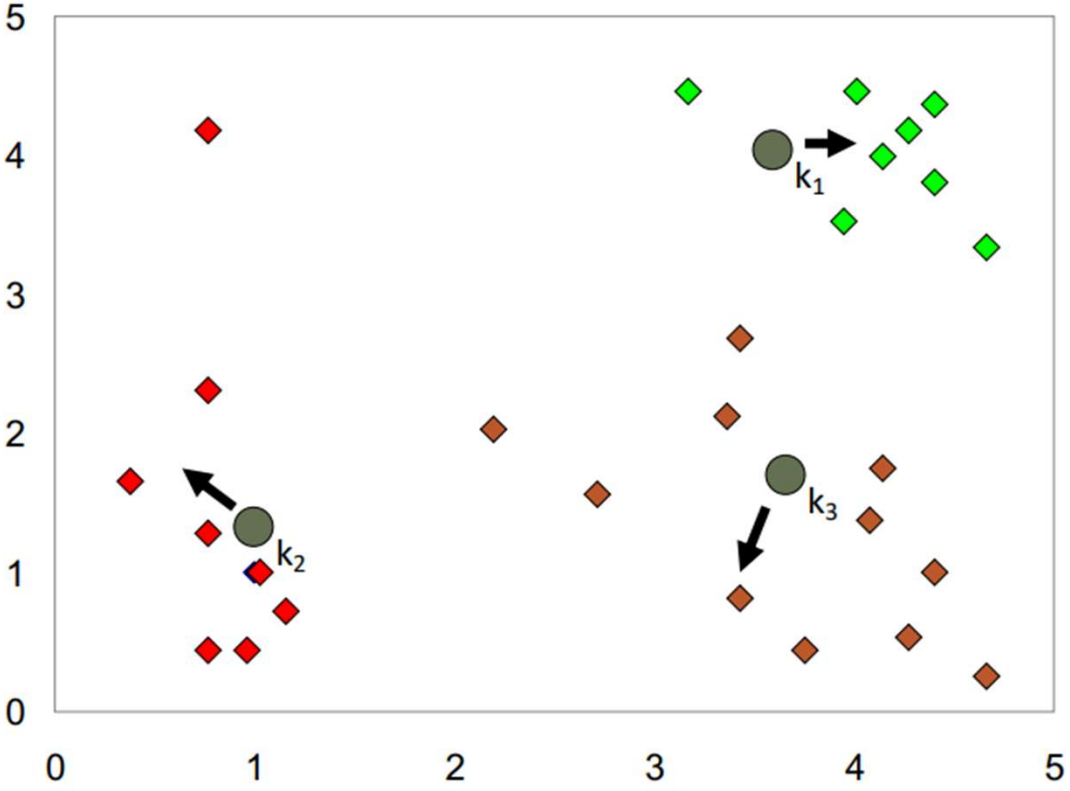


Demonstration of K-means

Step 3

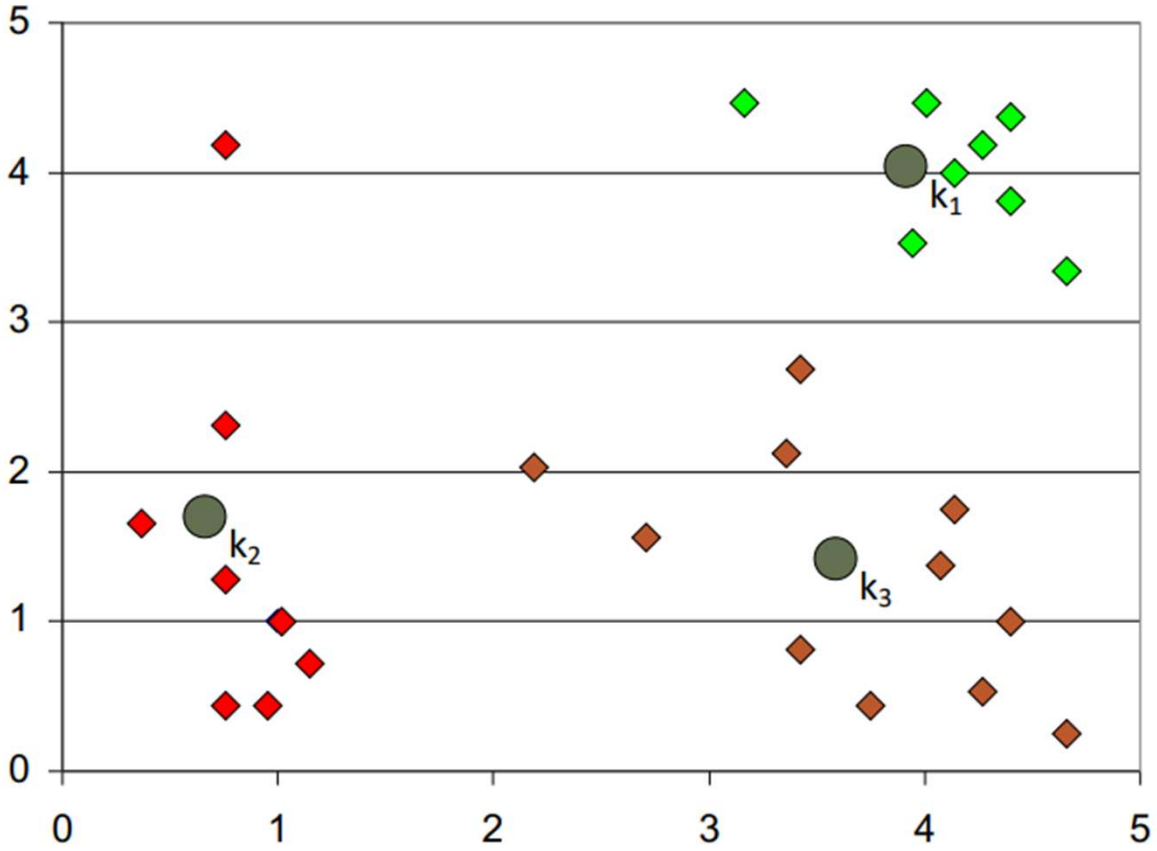


Step 4



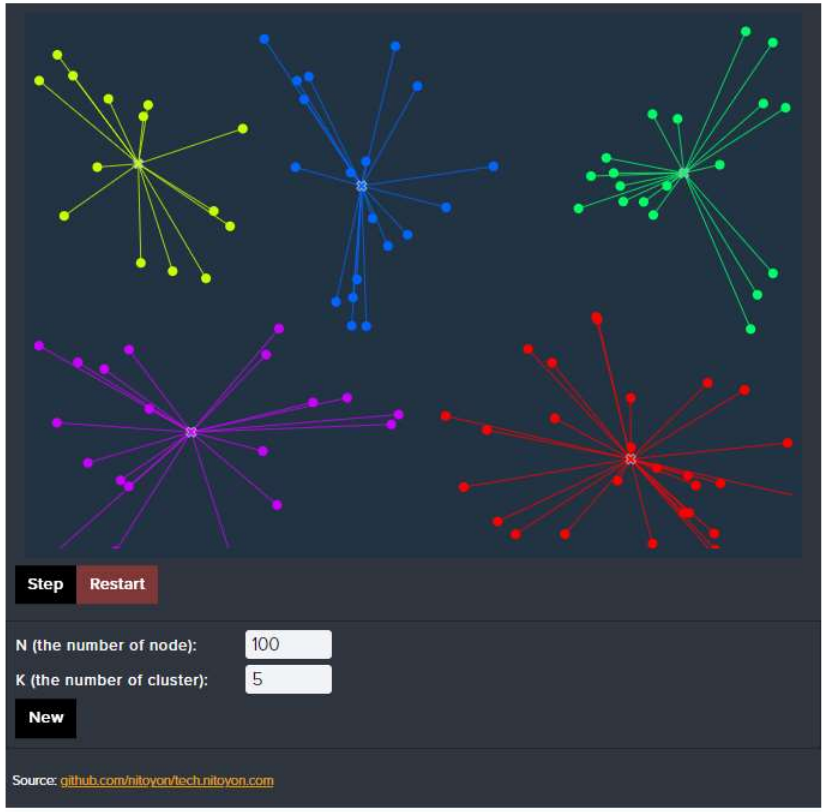
Demonstration of K-means

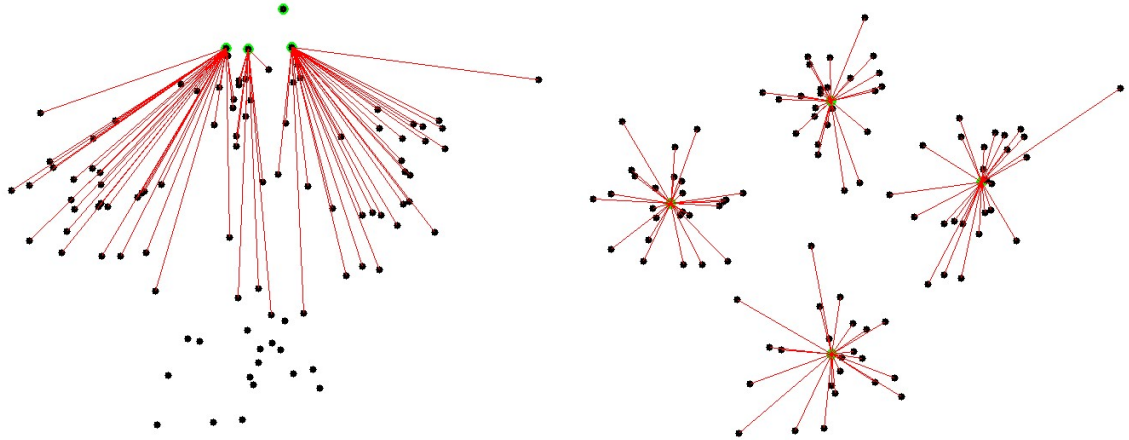
Done



Demonstrations of K-mean clustering

- https://www.learndatasci.com/tutorials/k-means-clustering-algorithms-python-intro/
- http://shabal.in/visuals/kmeans/4.html





K-Means

1. Create / Import dataset

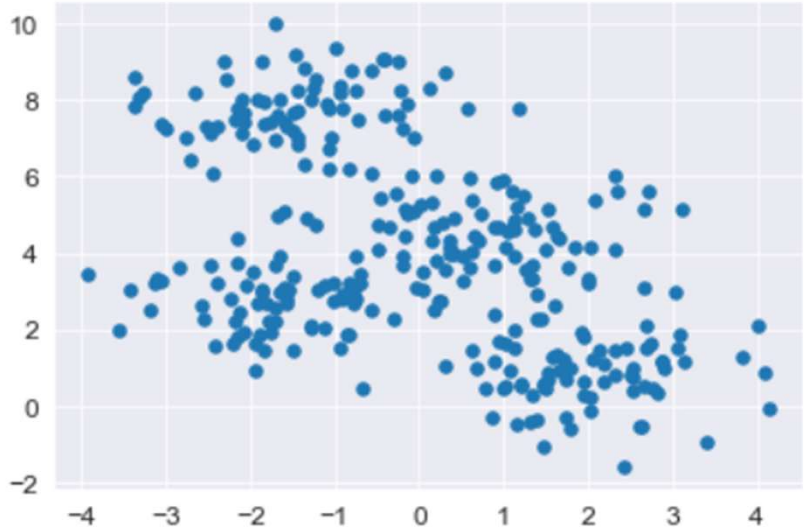
```
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.9, random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50)
plt.show()
```

2. Import Library

```
from sklearn.cluster import KMeans
```

3. Initialize K-means model

```
kmeans = KMeans(n_clusters=4, random_state=99)
```



Important Reference https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

sklearn.cluster.KMeans

sklearn.cluster.KMeans

class `sklearn.cluster.` **`KMeans`**(*n_clusters=8, *, init='k-means++', n_init=10, max_iter=300, tol=0.0001, precompute_distances='deprecated', verbose=0, random_state=None, copy_x=True, n_jobs='deprecated', algorithm='auto'*) [source]

K-Means clustering.

Read more in the User Guide.

Parameters:

n_clusters : *int, default=8*
The number of clusters to form as well as the number of centroids to generate.

init : *{'k-means++', 'random', ndarray, callable}, default='k-means++'*
Method for initialization:

'k-means++' : selects initial cluster centers for k-mean clustering in a smart way to speed up convergence. See section Notes in k_init for more details.

'random': choose `n_clusters` observations (rows) at random from data for the initial centroids.

If an ndarray is passed, it should be of shape (n_clusters, n_features) and gives the initial centers.

If a callable is passed, it should take arguments X, n_clusters and a random state and return an initialization.

n_init : *int, default=10*
Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.

max_iter : *int, default=300*
Maximum number of iterations of the k-means algorithm for a single run.

K-Means

1. Compute K-Mean clusters

```
kmeans.fit(X)
```

```
[[ 1.98345909  0.85592657]
 [-1.45052775  7.78696146]
 [-1.57553756  2.79999831]
 [ 0.91296517  4.54286627]]
[2 1 3 1 0 0 2 3 1 1 2 1 3 1 0 3 3 0 2 2 0 0 3 2 2 2 0 3 2 3 1 1 3 1 1 1 1
 1 2 0 3 2 1 3 2 2 1 2 1 0 2 0 1 0 0 2 1 2 1 0 1 3 1 2 2 2 1 0 1 2 3 3 1 2
 2 1 2 3 0 1 0 3 0 0 1 3 0 3 1 1 3 0 1 2 2 3 0 0 3 2 1 0 1 0 3 0 0 3 1 3 2
 2 0 1 0 3 1 0 0 3 2 0 2 0 0 0 0 2 0 2 1 2 2 0 1 2 2 1 2 1 1 2 3 2 3 2 1 3
 1 1 1 3 3 3 0 2 1 2 0 3 1 3 3 0 3 2 2 3 0 3 3 1 0 3 2 1 0 0 3 2 0 3 2 2 3
 3 3 3 0 1 2 2 3 3 2 2 2 3 2 1 3 2 0 2 3 2 2 1 3 1 3 2 3 3 1 2 2 0 0 3 1 0
 0 2 0 2 2 1 1 3 3 1 3 0 2 3 0 2 1 2 0 3 0 1 1 1 1 2 2 3 3 2 0 3 2 2 3 0 0
 1 3 3 2 0 1 2 3 1 3 0 0 2 2 0 0 0 0 3 1 1 0 0 3 0 0 0 1 3 1 3 0 0 1 1 1 0
 0 3 1 2]
```

2. Predict cluster index for each sample

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

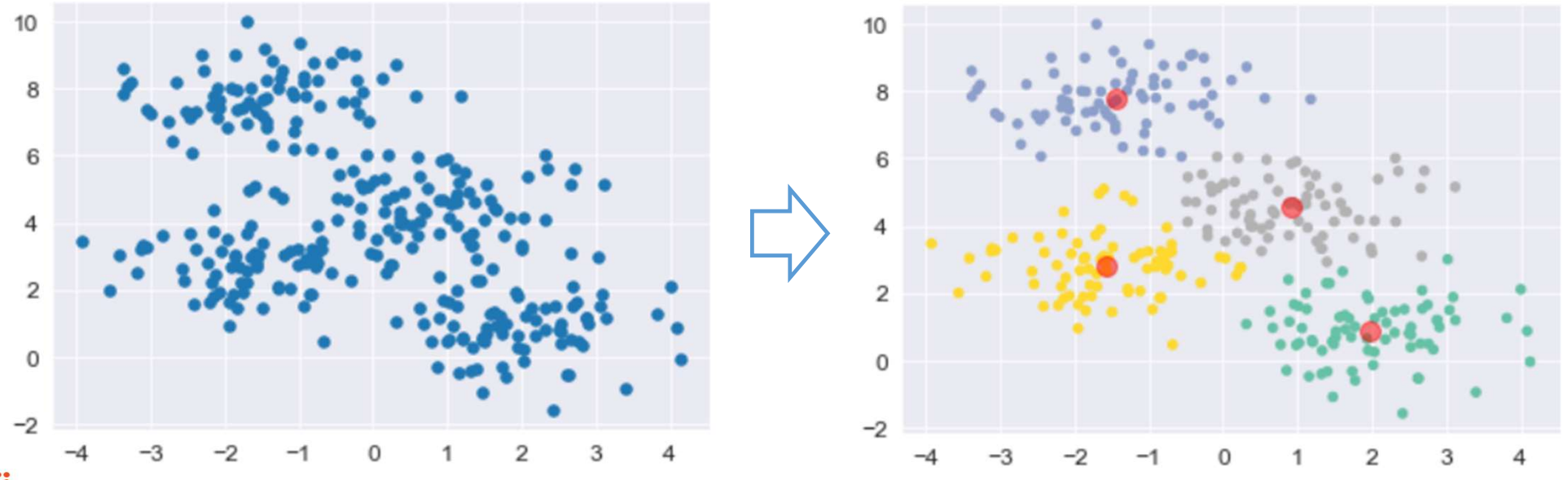
3. See the results

```
print(kmeans.cluster_centers_) # see centroid of each resulted cluster
print(kmeans.labels_)          # see cluster number of each data
```

K-Means

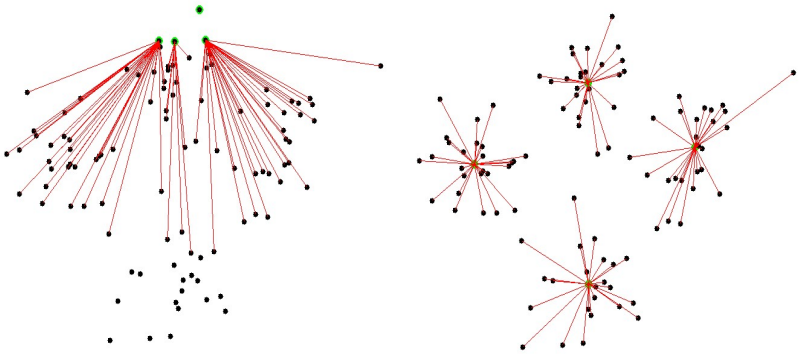
4. Visualize

```
plt.scatter(X[:, 0],  X[:, 1],  c=y_kmeans, s=20, cmap='Set2')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='darkgreen', s=100, alpha=0.5)
plt.show()
```



Properties of K-Means

- Guaranteed to converge in a finite number of iterations.



- Running time per iteration:
  - Assign data points to closest cluster center
    - = ?
  - Change the cluster center to the average of its assigned points
    - = ?

Outline

- Concept of cluster analysis
- K-mean clustering
- An application to customer segmentation
- Agglomerative clustering
- DBSCAN

Common data to do customer segmentation

Demographic Information

- Age อายุ
- Gender เพศ
- Marital Status สถานะสมรส
- Income รายได้

Transactional Information

- Time or Day of purchase เวลาหรือวันที่ซื้อ
- Frequency of purchase ความถี่การซื้อ
- $ amount purchase มูลค่าการซื้อ
- # of items purchase จำนวนสินค้าที่ซื้อ
- Products purchase สินค้าที่ซื้อ

- *RFM analysis*
  - Recency, Frequency, Monetary

K-means on Customer Segmentation

- Download Mall Customer Data (https://www.kaggle.com/vjchoudhary7/customer-segmentation-tutorial-in-python/data#)

```
# 1) Import all necessary libraries and modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# 2) Prepare dataset
df = pd.read_csv("Mall_Customers.csv")
df.head()
```

Any unnecessary attribute?

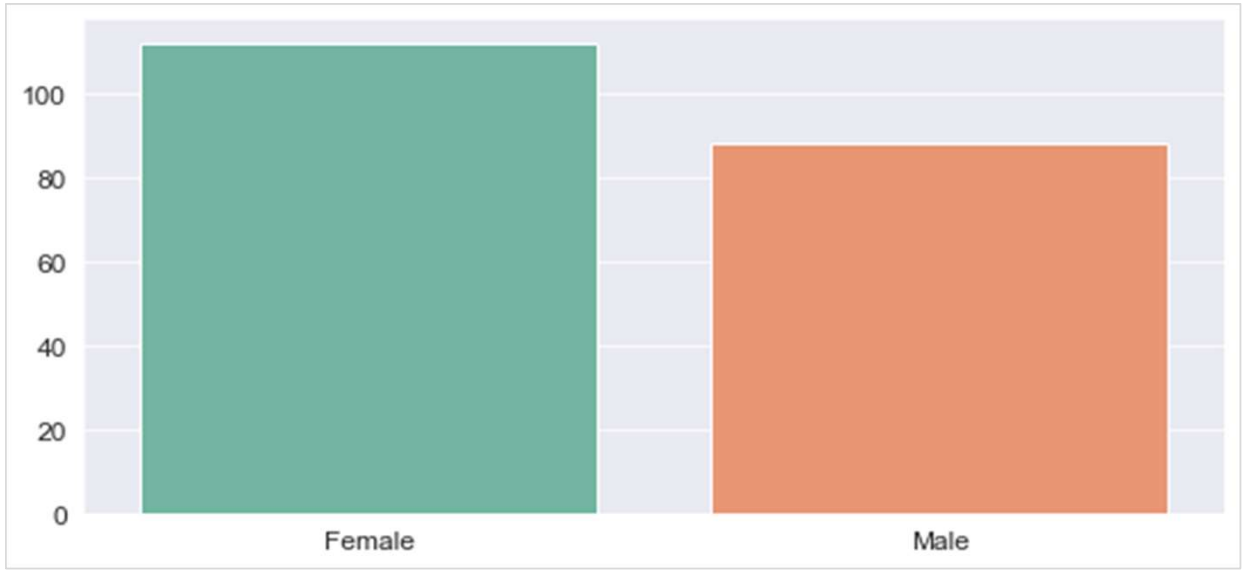
	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

```
# 3) pre-processing
df.drop(["CustomerID"], axis=1, inplace=True)
df.columns = ['Gender', 'Age', 'Income', 'Spending']
print(df.head())
```

	Gender	Age	Income	Spending
0	Male	19	15	39
1	Male	21	15	81
2	Female	20	16	6
3	Female	23	16	77
4	Female	31	17	40

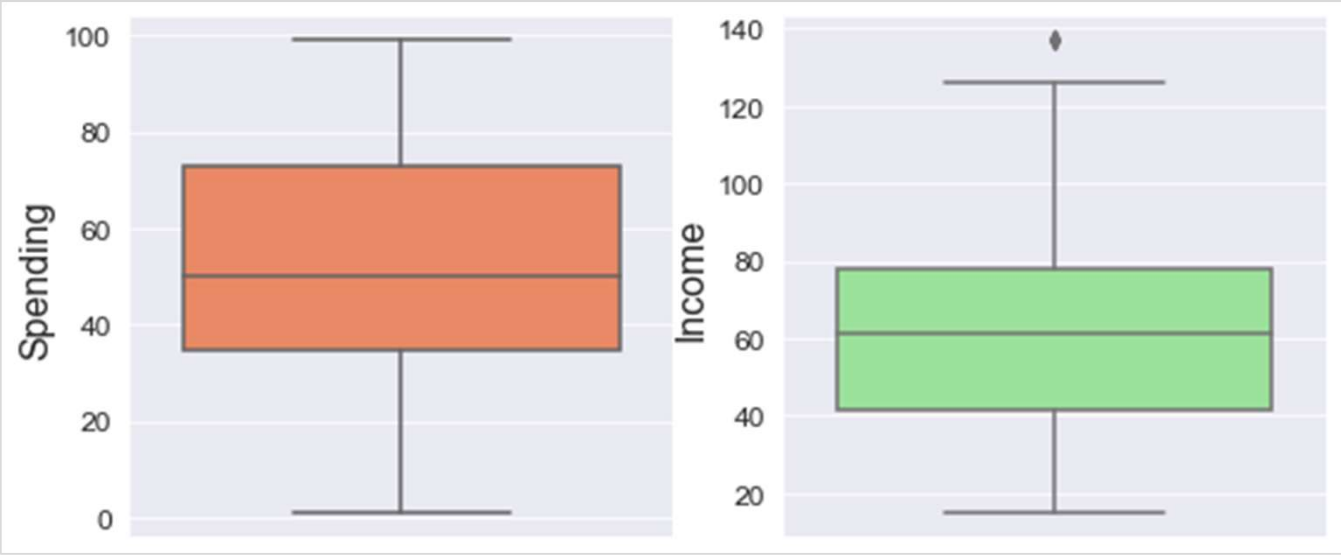
Compare number of males and females

```
genders = df.Gender.value_counts()
sns.set_style("darkgrid")
plt.figure(figsize=(10,4))
sns.barplot(x=genders.index, y=genders.values, palette="Set2")
plt.show()
```



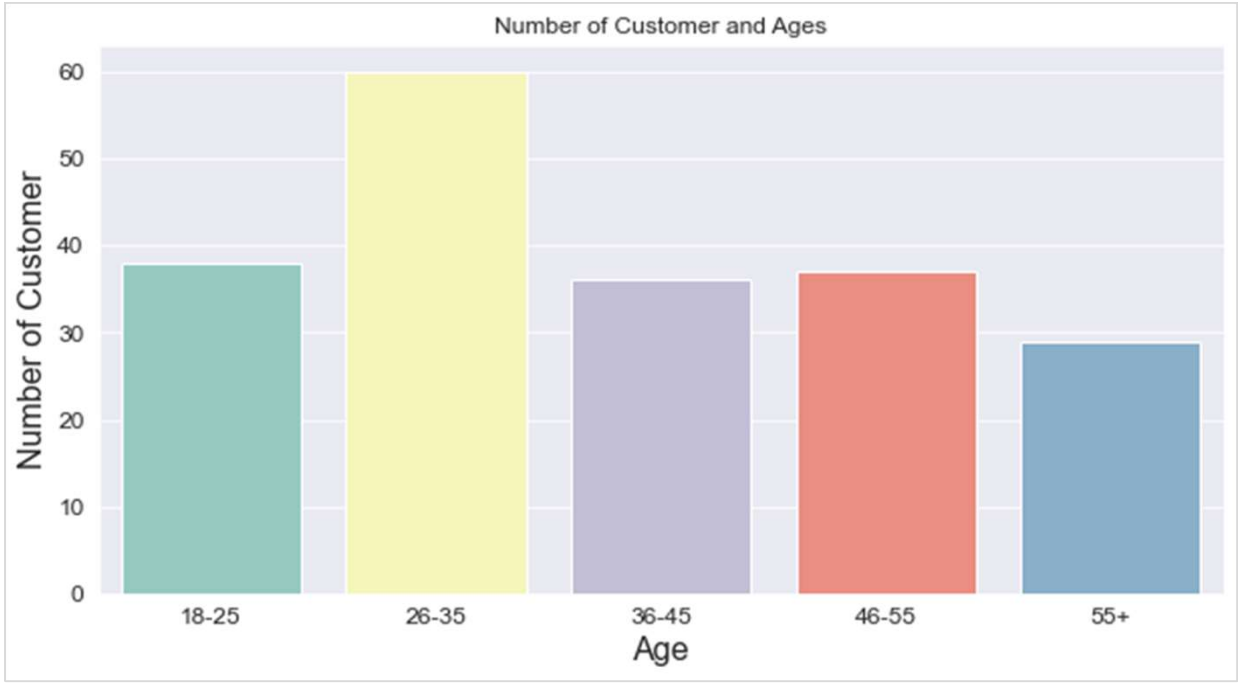
Visualize Spending Score and Annual Income

```
plt.figure(figsize=(9,4))
plt.subplot(1,2,1)
sns.boxplot(y=df["Spending"], color="coral")
plt.subplot(1,2,2)
sns.boxplot(y=df["Income"], color="lightgreen")
plt.show()
```



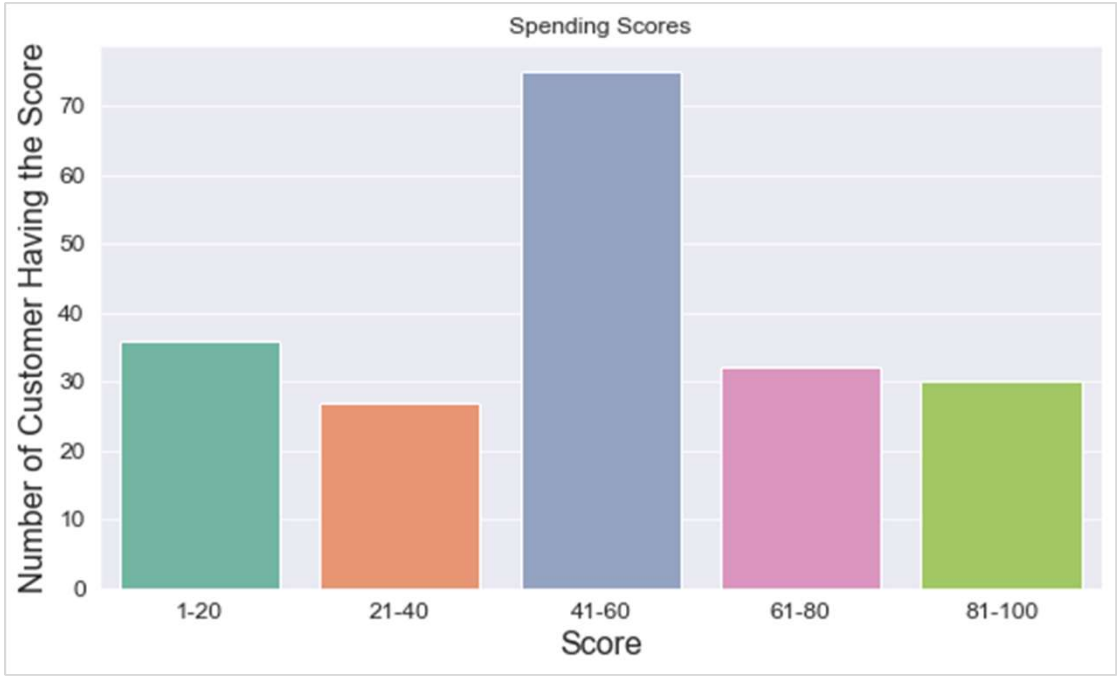
Compare number of customers by group of age

```
age18_25 = df.Age[(df.Age <= 25) & (df.Age >= 18)]
age26_35 = df.Age[(df.Age <= 35) & (df.Age >= 26)]
age36_45 = df.Age[(df.Age <= 45) & (df.Age >= 36)]
age46_55 = df.Age[(df.Age <= 55) & (df.Age >= 46)]
age55above = df.Age[df.Age >= 56]

x = ["18-25","26-35","36-45","46-55","55+"]
y = [len(age18_25.values),len(age26_35.values),
     len(age36_45.values),len(age46_55.values),
     len(age55above.values)]

plt.figure(figsize=(10,5))
sns.barplot(x=x, y=y, palette="Set3")
plt.title("Number of Customer and Ages")
plt.xlabel("Age")
plt.ylabel("Number of Customer")
plt.show()
```



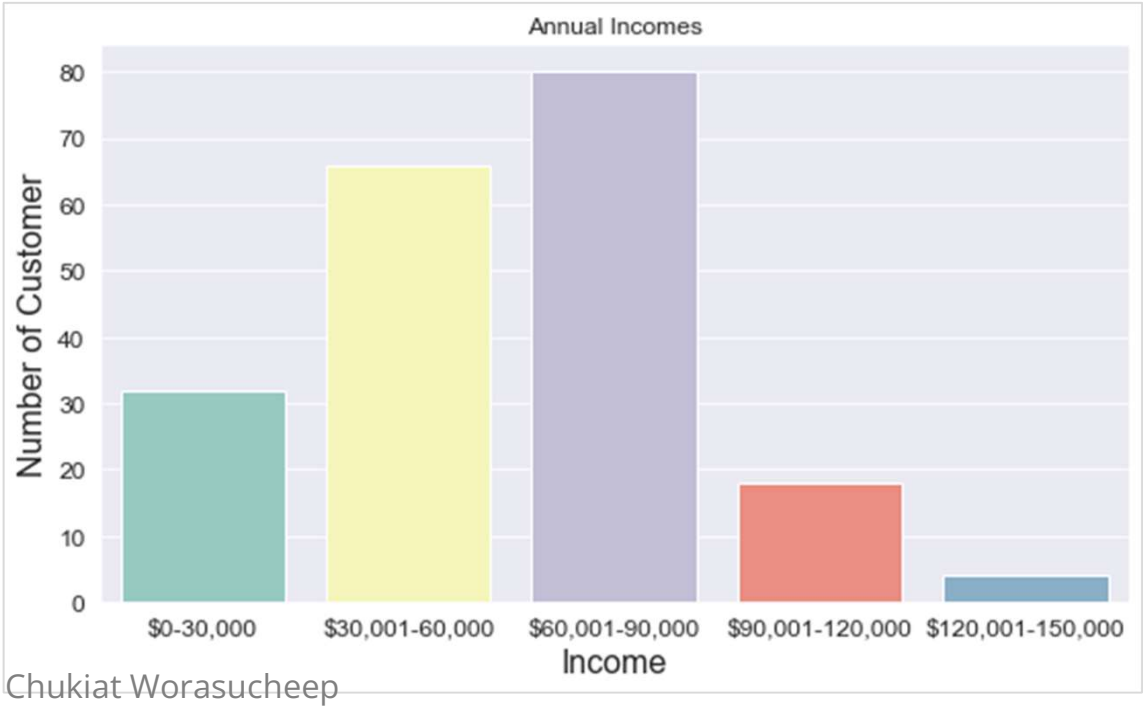
Number of customers grouped by spending score

```
ss1_20  = df["Spending"][(df["Spending"] >= 1) & (df["Spending"] <= 20)]
ss21_40 = df["Spending"][(df["Spending"] >= 21) & (df["Spending"] <= 40)]
ss41_60 = df["Spending"][(df["Spending"] >= 41) & (df["Spending"] <= 60)]
ss61_80 = df["Spending"][(df["Spending"] >= 61) & (df["Spending"] <= 80)]
ss81_100 = df["Spending"][(df["Spending"] >= 81) & (df["Spending"] <= 100)]

ssx = ["1-20", "21-40", "41-60", "61-80", "81-100"]
ssy = [len(ss1_20.values), len(ss21_40.values), len(ss41_60.values),
      len(ss61_80.values), len(ss81_100.values)]

plt.figure(figsize=(9,5))
sns.barplot(x=ssx, y=ssy, palette="Set2")
plt.title("Spending Scores")
plt.xlabel("Score")
plt.ylabel("Number of Customer Having the Score")
plt.show()
```



Number of customers grouped by annual income

```
ai0_30 = df["Annual Income (k$)"][(df["Annual Income (k$)"] >= 0) & (df["Annual Income (k$)"] <= 30)]
ai31_60 = df["Annual Income (k$)"][(df["Annual Income (k$)"] >= 31) & (df["Annual Income (k$)"] <= 60)]
ai61_90 = df["Annual Income (k$)"][(df["Annual Income (k$)"] >= 61) & (df["Annual Income (k$)"] <= 90)]
ai91_120 = df["Annual Income (k$)"][(df["Annual Income (k$)"] >= 91) & (df["Annual Income (k$)"] <= 120)]
ai121_150 = df["Annual Income (k$)"][(df["Annual Income (k$)"] >= 121) & (df["Annual Income (k$)"] <= 150)]

aix = ["$ 0 - 30,000", "$ 30,001 - 60,000", "$ 60,001 - 90,000", "$ 90,001 - 120,000", "$ 120,001 - 150,000"]
aiy = [len(ai0_30.values), len(ai31_60.values), len(ai61_90.values), len(ai91_120.values), len(ai121_150.values)]

plt.figure(figsize=(15,6))
sns.barplot(x=aix, y=aiy, palette="Set3")
plt.title("Annual Incomes")
plt.xlabel("Income")
plt.ylabel("Number of Customer")
plt.show()
```



K-means on *Customer Segmentation*

```
df.drop(['Gender'], axis=1, inplace=True)
df
```

Out[16]:

	Age	Income	Spending
0	19	15	39
1	21	15	81
2	20	16	6
3	23	16	77
4	31	17	40
...	...	...	...
195	35	120	79
196	45	126	28
197	32	126	74
198	32	137	18
199	30	137	83

200 rows × 3 columns

```
# 4. Initialize model
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=5, random_state=0)

# 5) Compute K-mean clustering using Age, Income, Spending
kmeans.fit(df)
print('Centroids:\n', kmeans.cluster_centers_)

# 6) Compute cluster index for each point
y_kmeans = kmeans.predict(df)
print('Cluster IDs:\n', y_kmeans)
```

Age *Income* *Spending*

```
Centroids:
 [[45.2173913  26.30434783 20.91304348]
 [43.08860759 55.29113924 49.56962025]
 [32.69230769 86.53846154 82.12820513]
 [40.66666667 87.75        17.58333333]
 [25.52173913 26.30434783 78.56521739]]
Cluster IDs:
 [0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4
 4 0 4 0 4 0 4 0 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 2 3 2 1 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 1
 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2
 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2]
```

See Within-Cluster Sum of Square

```
# Weighted sum of squared distances of samples to their closest cluster center
print('WCSS =', kmeans.inertia_)
```

```
WCSS = 75350.77917248776
```

sklearn.cluster.KMeans

Attributes:

cluster_centers_ : *ndarray of shape (n_clusters, n_features)*
Coordinates of cluster centers. If the algorithm stops before fully converging (see `tol` and `max_iter`), these will not be consistent with `labels_`.

labels_ : *ndarray of shape (n_samples,)*
Labels of each point

inertia_ : *float*
Sum of squared distances of samples to their closest cluster center.

n_iter_ : *int*
Number of iterations run.

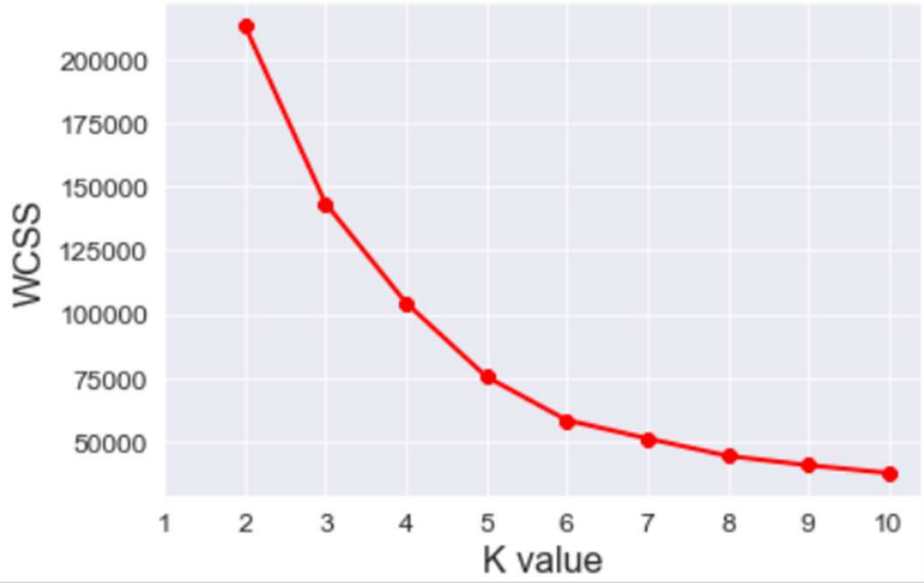
Important Reference https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

What K should we choose?

- Determining good cluster size using elbow method

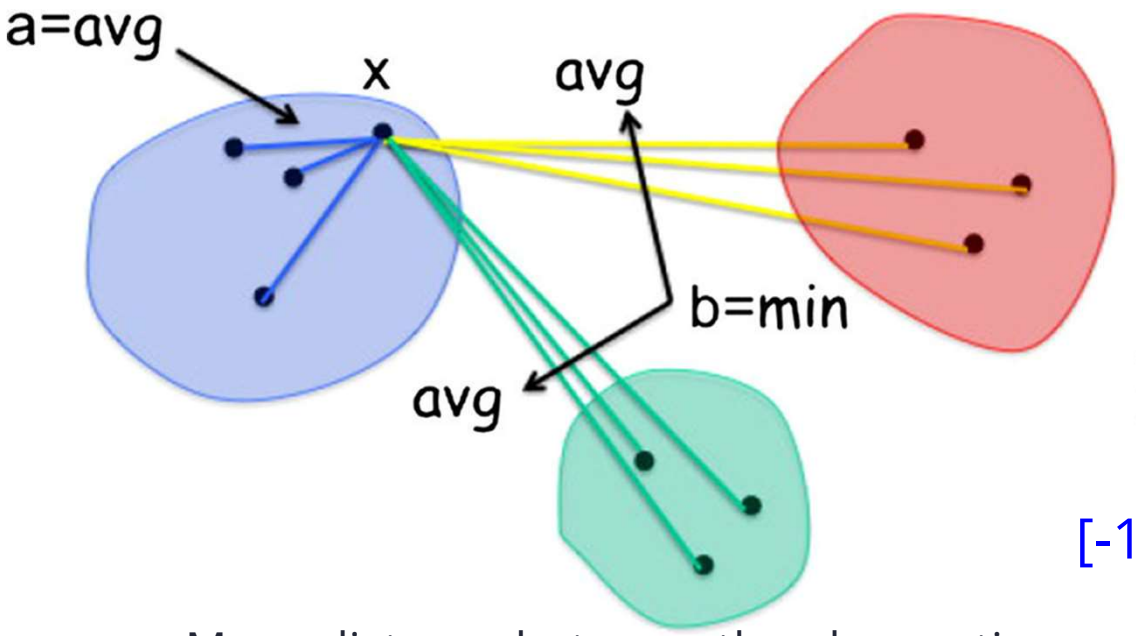
```
from sklearn.cluster import KMeans
from sklearn import metrics
wcss = []
for k in range(2, 11):  # test for 2 to 10 clusters
    kmeans = KMeans(n_clusters=k, random_state=123)
    kmeans.fit(df)
    labels = kmeans.labels_
    wcss.append(kmeans.inertia_)
    metrics.silhouette_score(df, labels, metric='euclidean')
    print("silhouette_score of k = " + str(k) + ":{:.2f}"
        .format(metrics.silhouette_score(df, labels, metric='euclidean')))
plt.plot(range(2,11), wcss, linewidth=2, color="red", marker ="8")
plt.xticks(np.arange(1,11,1))
plt.xlabel("K value")
plt.ylabel("WCSS")
plt.grid(True)
plt.show()
```

```
silhouette_score of k = 2:0.29
silhouette_score of k = 3:0.38
silhouette_score of k = 4:0.41
silhouette_score of k = 5:0.44
silhouette_score of k = 6:0.45
silhouette_score of k = 7:0.44
silhouette_score of k = 8:0.43
silhouette_score of k = 9:0.41
silhouette_score of k = 10:0.40
```



Performance Metrics for Clustering

1. *Silhouette Coefficient* *(1987)*
2. *Dunn's index* *(1974)*
3. *Davies–Bouldin* *(1979)*



$$\frac{b_i - a_i}{\max(bi, a_i)}$$

[-1, 1] and Higher is better

- **a** = Mean distance between the observation and all other data points *in the same* cluster. This distance can also be called a **mean intra-cluster distance.**
- **b** = Mean distance between the observation and *all other data points of the next nearest cluster*. This distance can also be called a **mean nearest-cluster distance.**

Rerun with the same K=5

[19]:
```
# 1) Create K-Mean clustering
kmeans = KMeans(n_clusters=5, random_state=123)

# 2) Compute K-mean clustering
kmeans.fit(df)
print(kmeans.cluster_centers_)

# 3) Compute cluster index for each point
y_kmeans = kmeans.predict(df)

print(y_kmeans)
```

```
[[43.08860759 55.29113924 49.56962025]
 [25.52173913 26.30434783 78.56521739]
 [40.66666667 87.75       17.58333333]
 [32.69230769 86.53846154 82.12820513]
 [45.2173913  26.30434783 20.91304348]]
[4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4 1 4
 1 4 1 4 1 4 1 4 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 3 2 3 0 3 2 3 2 3 2 3 2 3 2 3 2 3 0 3 2 3 2 3
 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2
 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3]
```

Get the same clusters, but with changing cluster IDs.

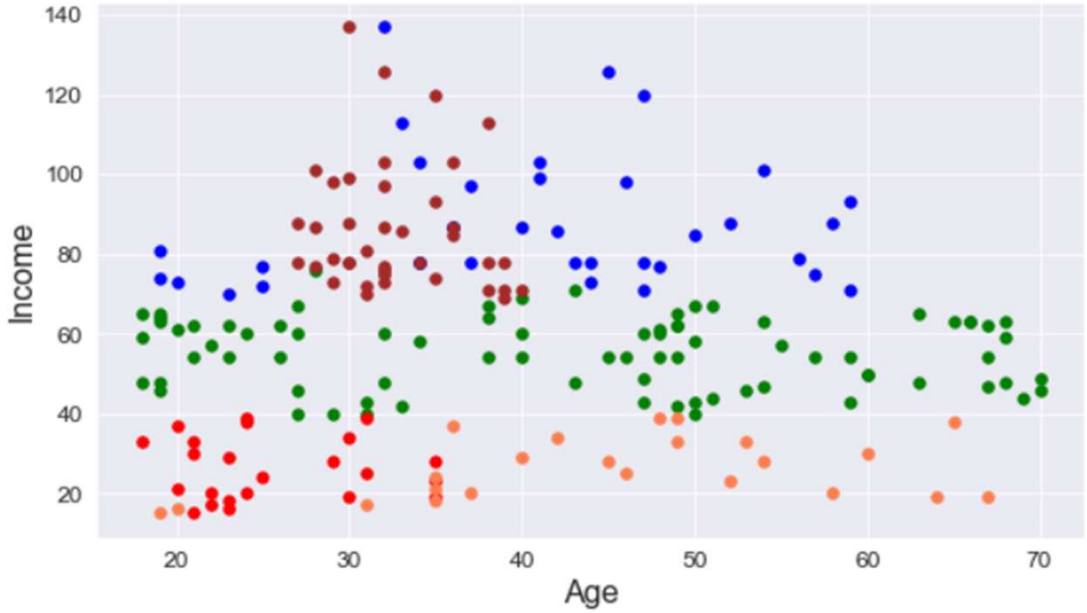
Add cluster ID into the dataframe

[20]:
```
# Add cluster ID into the data df
df['cluster'] = y_kmeans
print(df)
```

```
     Age  Income  Spending  cluster
0     19      15        39        4
1     21      15        81        1
2     20      16         6        4
3     23      16        77        1
4     31      17        40        4
..   ...     ...       ...      ...
195   35     120        79        3
196   45     126        28        2
197   32     126        74        3
198   32     137        18        2
199   30     137        83        3

[200 rows x 4 columns]
```

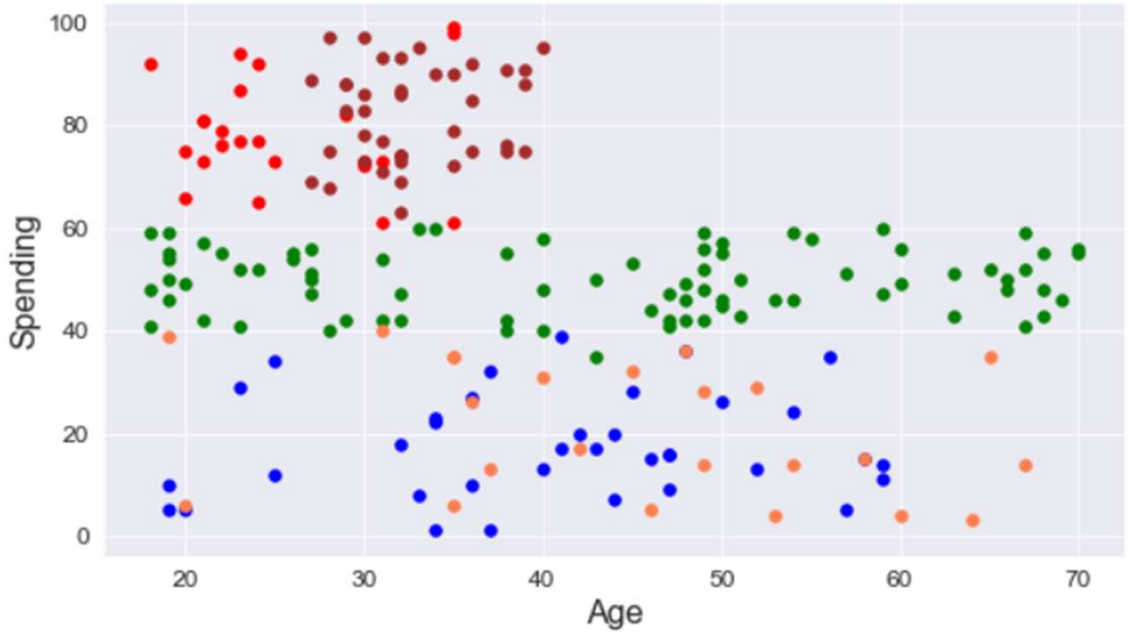
Visualize the Clusters – Age vs Annual Income

```
In [21]: fig = plt.figure(figsize=(9,5))
ax = fig.add_subplot(111)
ax.scatter(df.Age[df.cluster == 0], df["Income"][df.cluster == 0], c='green', s=30)
ax.scatter(df.Age[df.cluster == 1], df["Income"][df.cluster == 1], c='red',   s=30)
ax.scatter(df.Age[df.cluster == 2], df["Income"][df.cluster == 2], c='blue',  s=30)
ax.scatter(df.Age[df.cluster == 3], df["Income"][df.cluster == 3], c='brown', s=30)
ax.scatter(df.Age[df.cluster == 4], df["Income"][df.cluster == 4], c='coral', s=30)

plt.xlabel("Age")
plt.ylabel("Income")
plt.show()
```



Visualize the Clusters – Age vs Spending Score

In [22]:
```
fig = plt.figure(figsize=(9,5))
ax = fig.add_subplot(111)
ax.scatter(df.Age[df.cluster == 0], df["Spending"][df.cluster == 0], c='green', s=30)
ax.scatter(df.Age[df.cluster == 1], df["Spending"][df.cluster == 1], c='red', s=30)
ax.scatter(df.Age[df.cluster == 2], df["Spending"][df.cluster == 2], c='blue', s=30)
ax.scatter(df.Age[df.cluster == 3], df["Spending"][df.cluster == 3], c='brown', s=30)
ax.scatter(df.Age[df.cluster == 4], df["Spending"][df.cluster == 4], c='coral', s=30)
plt.xlabel("Age")
plt.ylabel('Spending')
plt.show()
```



Strengths and weaknesses of K-Mean algorithm

- Strengths of K-Mean Algorithm:
  - Relatively simple to implement.
  - Scales to large data sets.
  - Guarantees convergence.
  - Can warm-start the positions of centroids.

- Weaknesses of K-Mean Algorithm:
  - Choosing k manually
  - Requires numerical data. Categorical data may have to switch to *K-modes*, and *K-prototypes* for both types.
  - Being dependent on initial values
  - Clustering outliers?

Outline

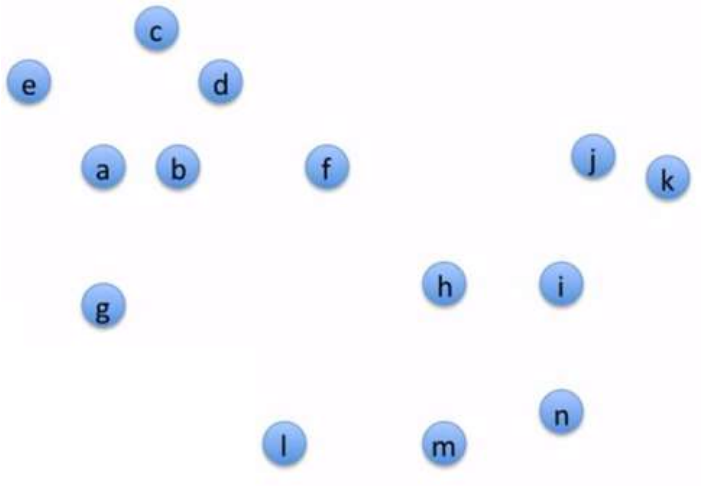
- Concept of cluster analysis
- K-mean clustering
- An application to customer segmentation
- Agglomerative clustering
- DBSCAN

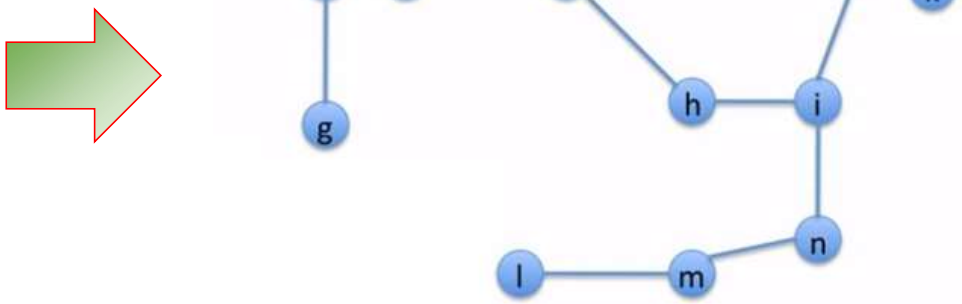
Hierarchical clustering

- K-Means create disjoint clusters, resulting in a “flat” data representation
- However, sometimes it is desirable to obtain a hierarchical representation of data, with clusters and sub-clusters arranged in a tree-structured fashion
  - Hierarchical representations are commonly used in the sciences (i.e., biological taxonomy)

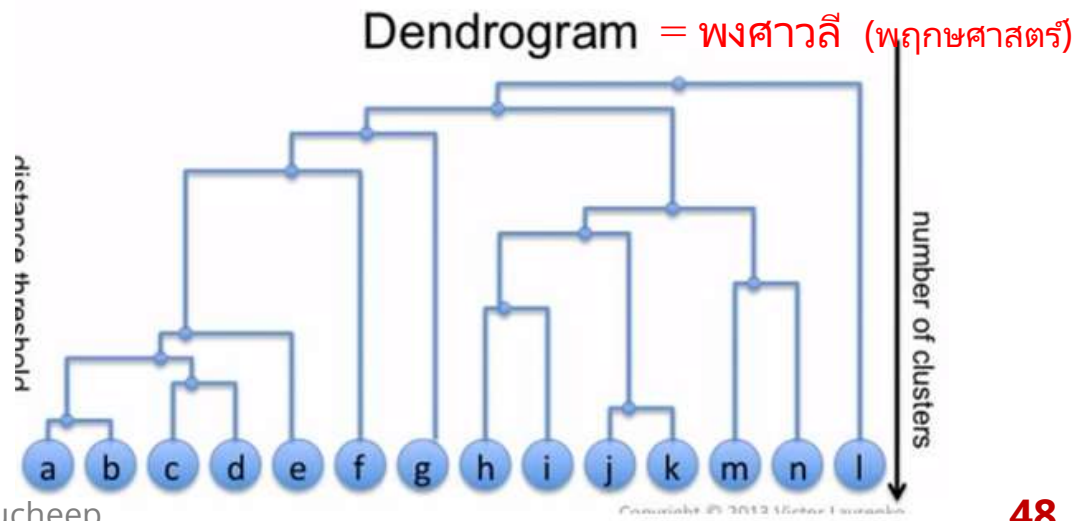
- Hierarchical clustering methods can be grouped in two general classes
  - *Agglomerative* (a.k.a. bottom-up)
    - Starting with N singleton clusters, successively merge clusters until one cluster is left
  - Divisive (a.k.a. top-down)
    - Starting with a unique cluster, successively split the clusters until N singleton examples are left.

Agglomerative hierarchical clustering



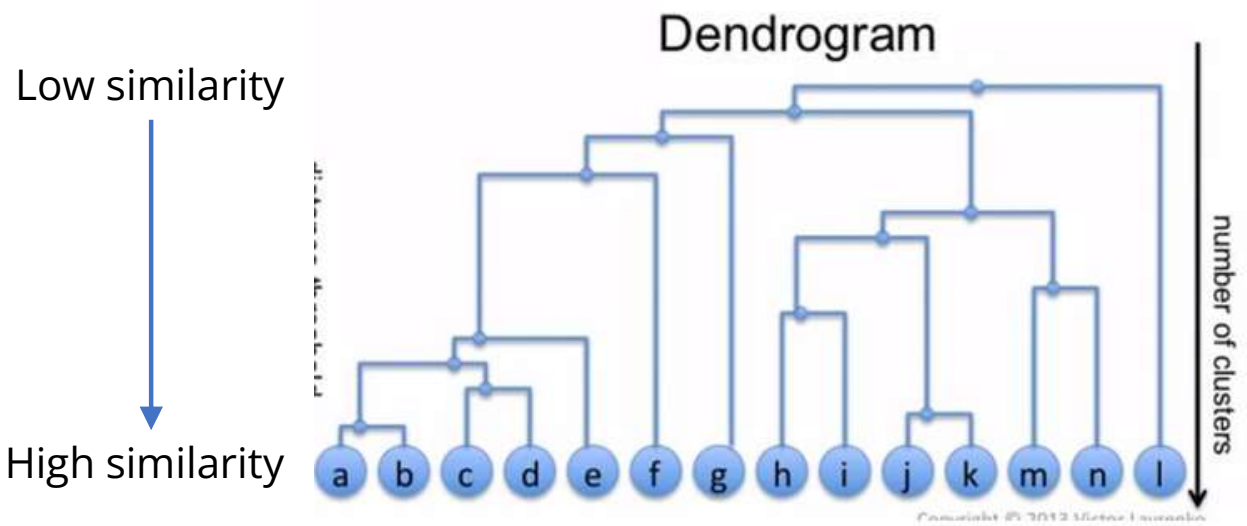


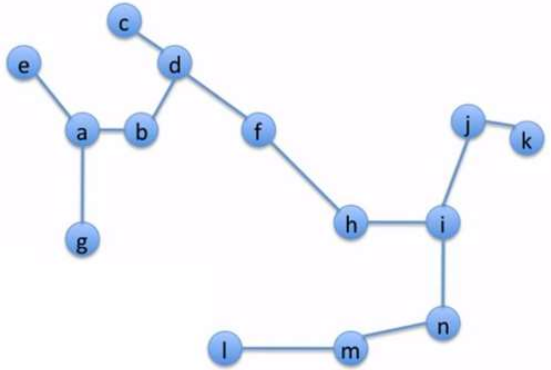
Bottom-up hierarchical clustering method



Dendrogram

- *Dendrogram* is a binary tree that shows the structure of the clusters.
- Preferred representation for hierarchical clusters.
- Dendrogram provides the similarity measure between clusters (the vertical axis).

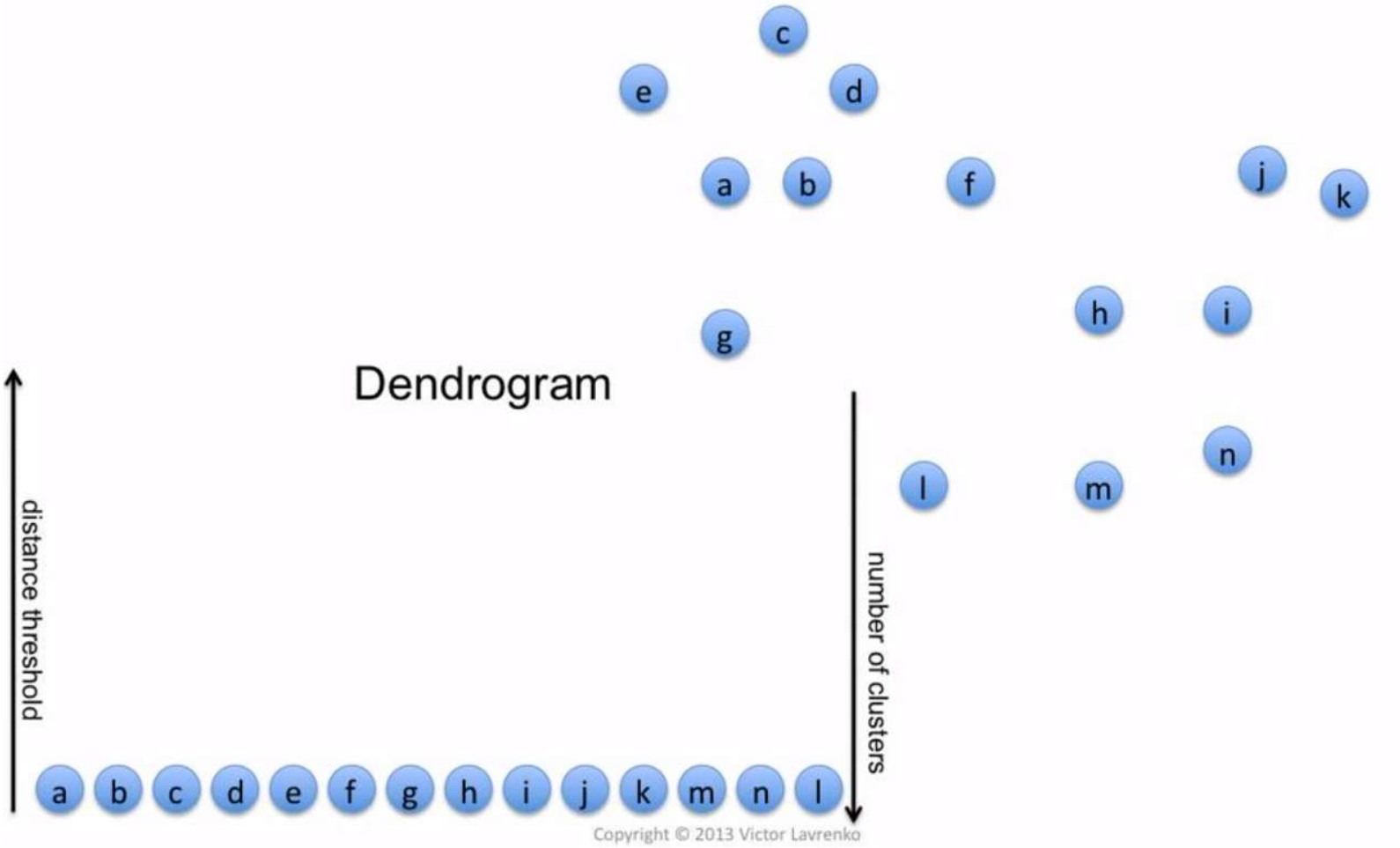




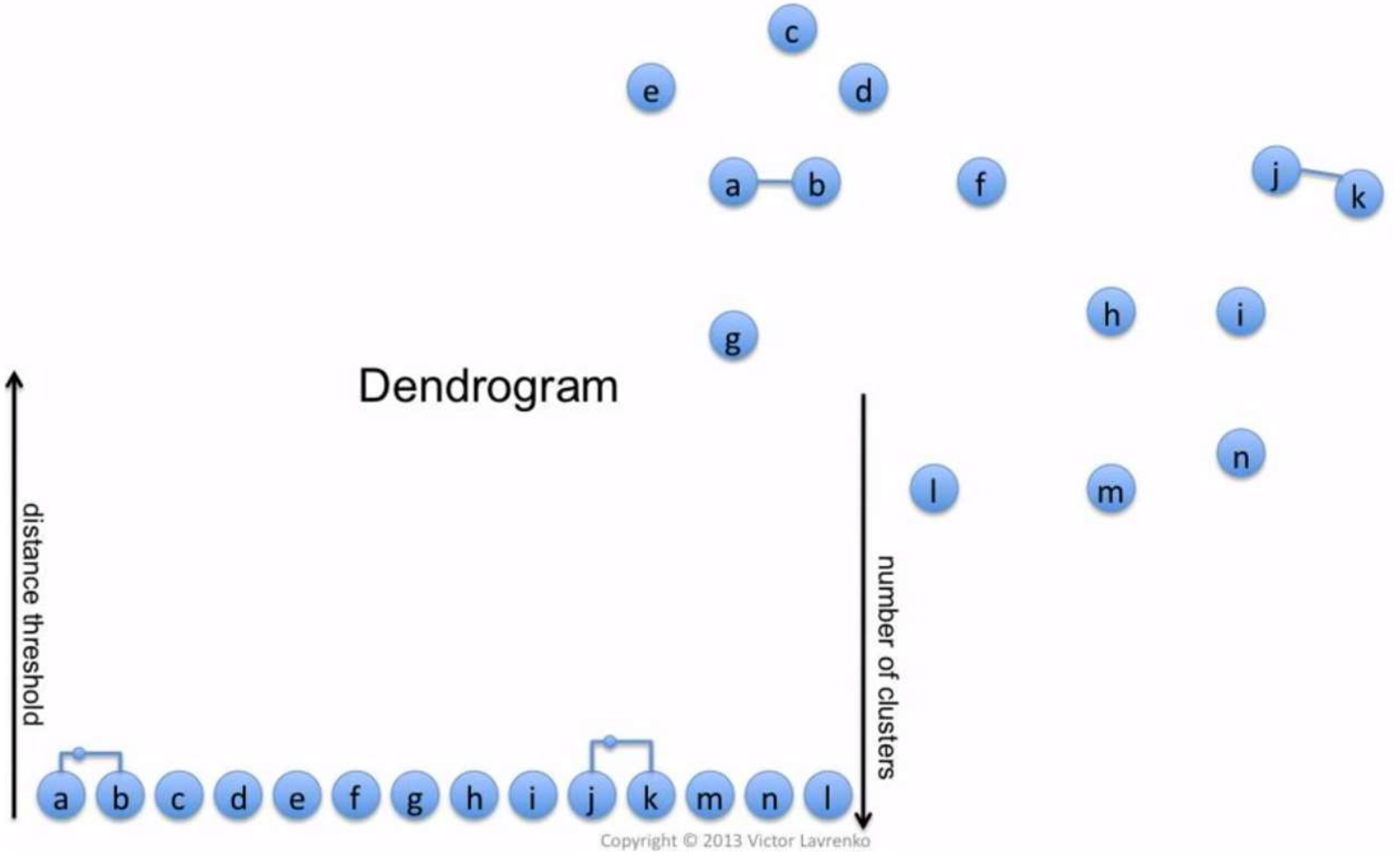
Agglomerative clustering

- Start: ให้แต่ละ item นับเป็น 1 กลุ่ม
- Repeat จนกว่าจะเหลือ cluster เดียว
  - หา 2 clusters ที่อยู่ใกล้กันที่สุด ... min D(c1, c2)
  - รวม 2 clusters นั้นกลายเป็น cluster ใหม่เดียว (และลบ 2 clusters เดิมนั้นทิ้งด้วย)
- Produce a dendrogram (hierarchical tree of clusters)

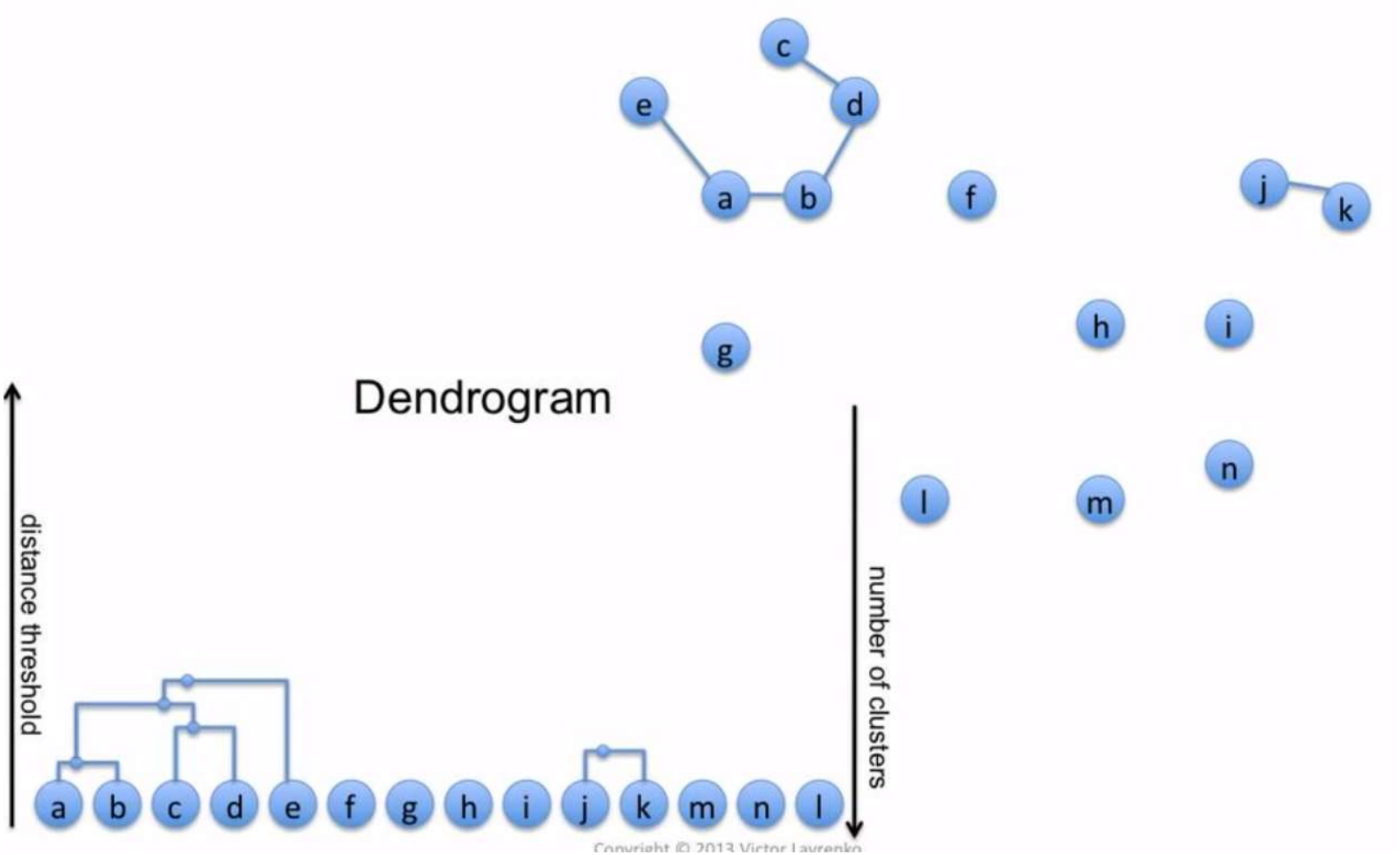
Agglomerative clustering – example



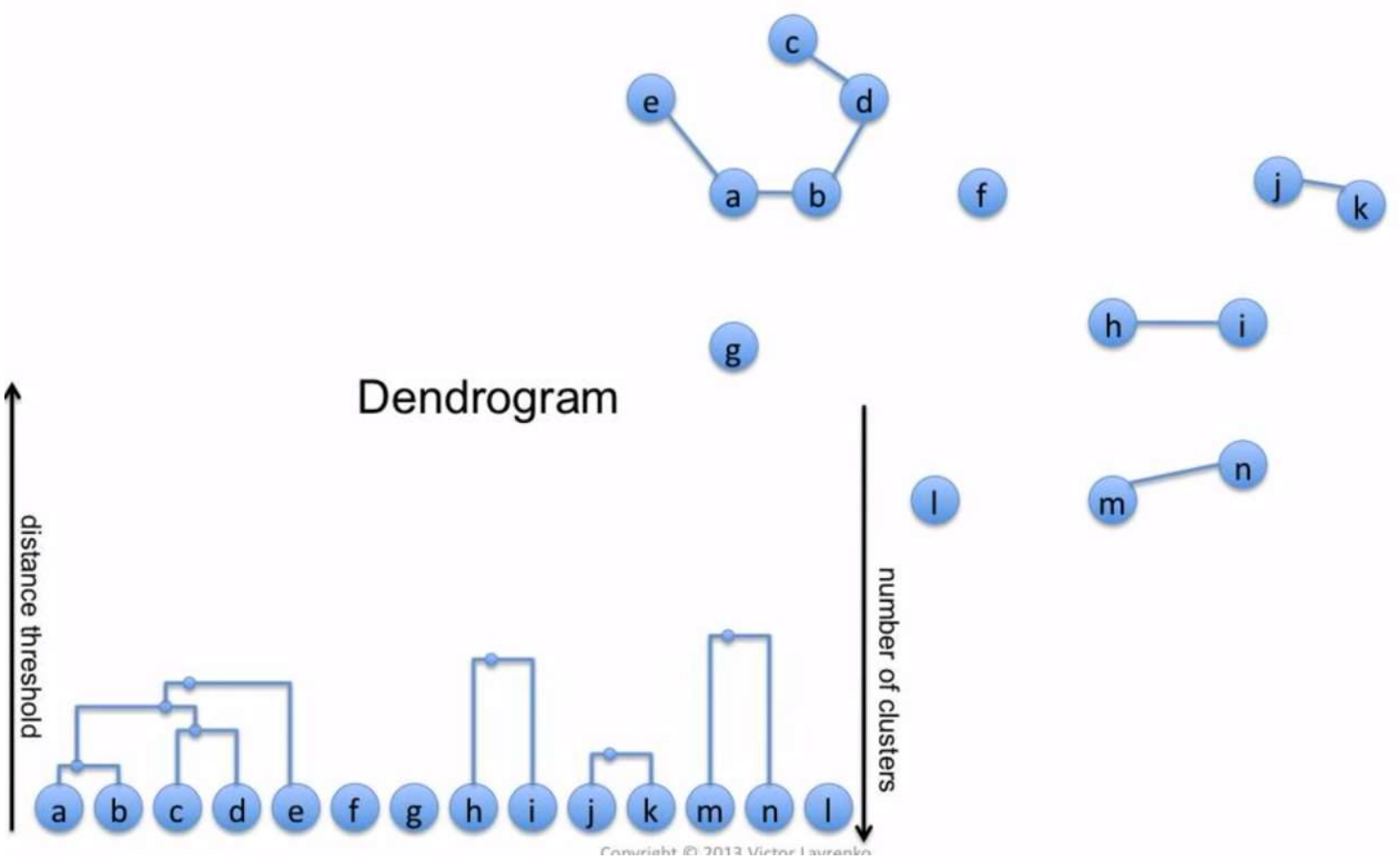
Agglomerative clustering – example – step 1



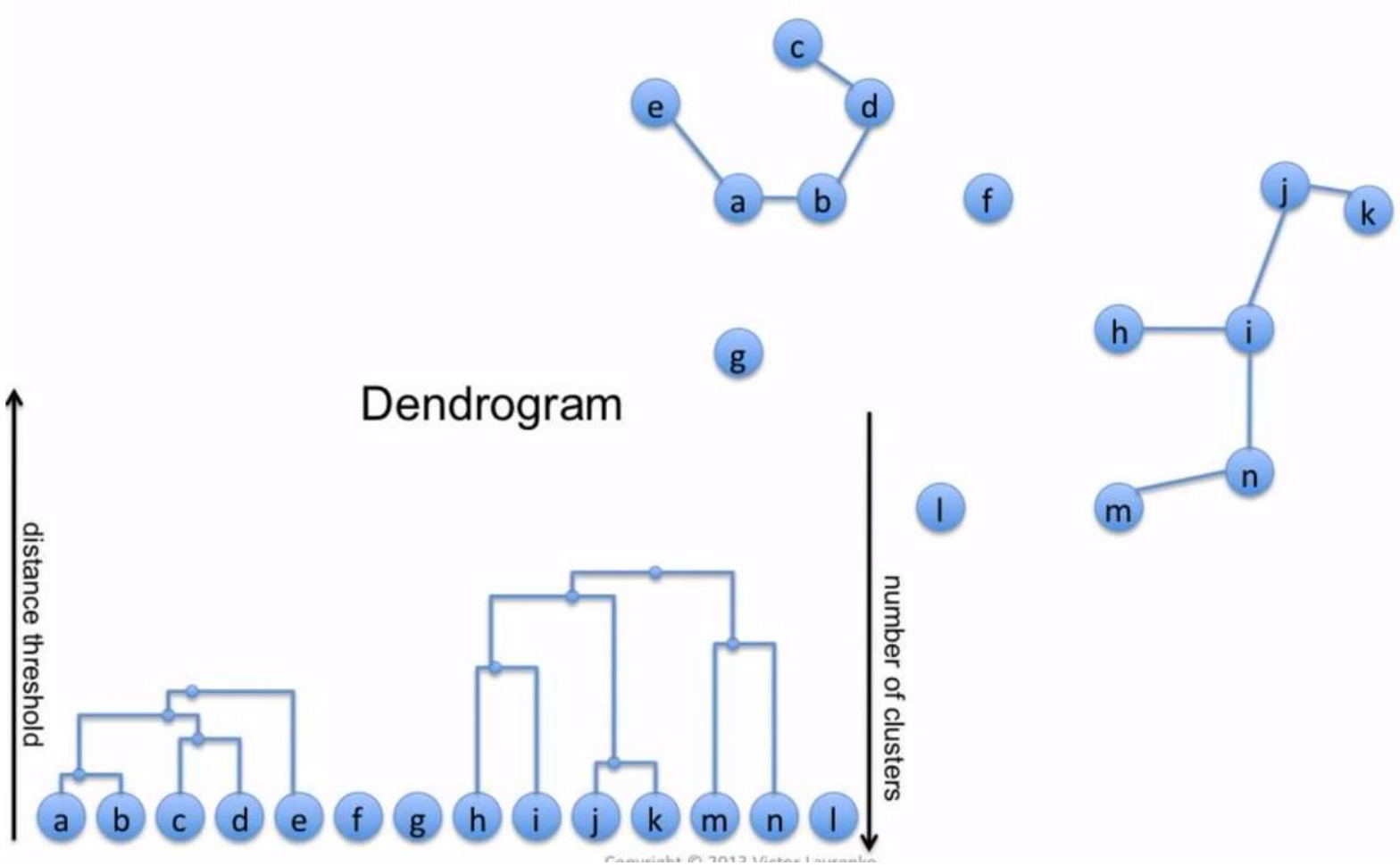
Agglomerative clustering – example – step 5



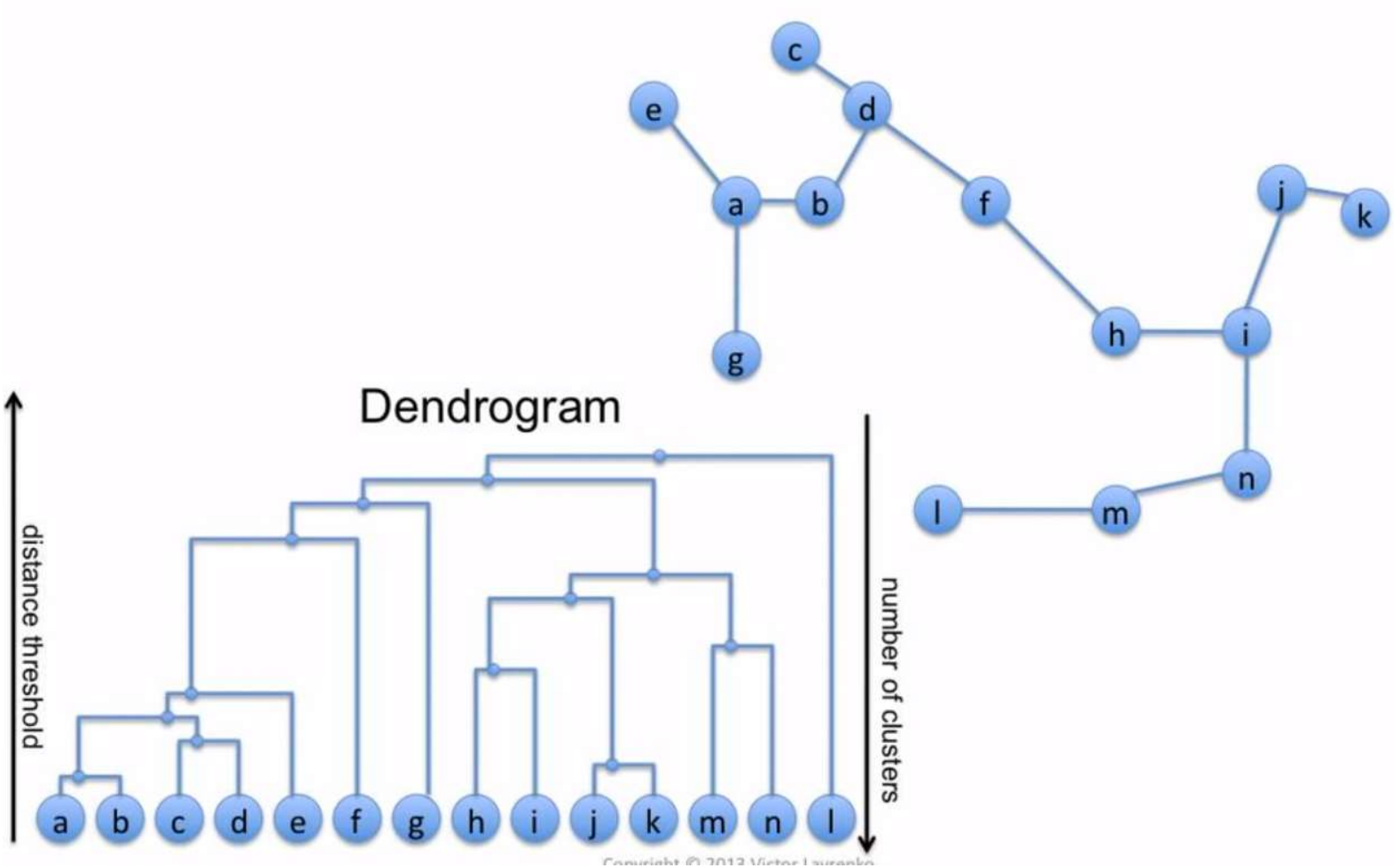
Agglomerative clustering – example – step 7



Agglomerative clustering – example – step 9

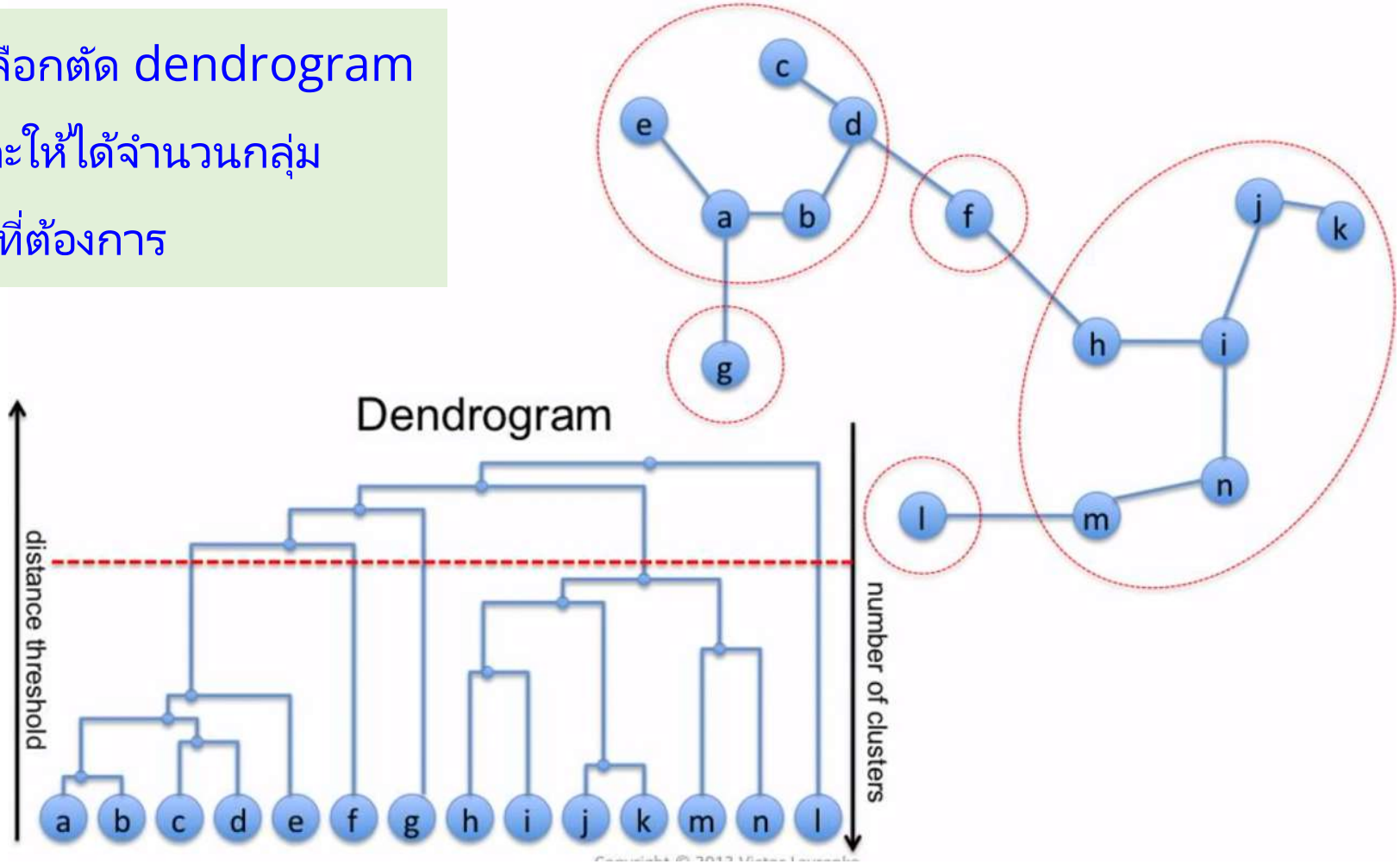


Agglomerative clustering – example – step 13



สุดท้าย.... ต้องการ flat clusters เพื่อเลือกจำนวนกลุ่ม (clusters)

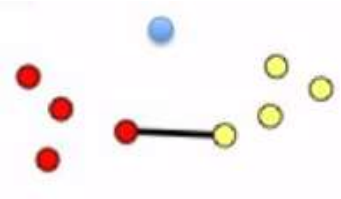
ทำได้โดยเลือกตัด dendrogram ณ ระดับที่จะให้ได้จำนวนกลุ่ม (cluster) ที่ต้องการ



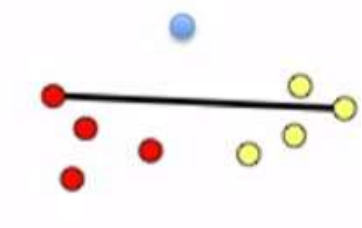
Mode or Cluster Distance Measures

- ใน Agglomerate hierarchical clustering การรวมกลุ่มกันจะค่อย ๆ รวมกลุ่มที่ใกล้กันมากที่สุดเข้าด้วยกัน. คำว่า*ใกล้กัน* มีนิยามอยู่มากกว่า 4 วิธี ได้แก่

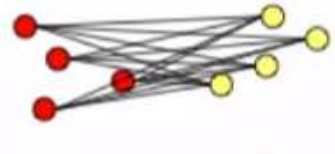
1. Single link (closest pair)
   - ระยะห่างระหว่างจุดที่ใกล้ที่สุดของสอง clusters
   - จะได้กลุ่มที่เชื่อมต่อกันเป็นสายยาว



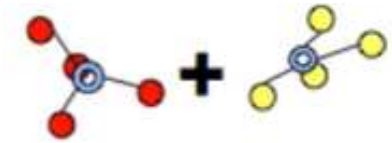
2. Complete link (farthest pair)
   - ระยะห่างระหว่างจุดที่ไกลที่สุดของสอง clusters
   - จะได้กลุ่มที่มีขนาดไม่ต่างกันมาก (เทียบกับ Single Link)



3. Average link
   - ค่าเฉลี่ยของระยะห่างระหว่างทุกคู่จากสอง clusters
   - ทนต่อ outliers มากขึ้น

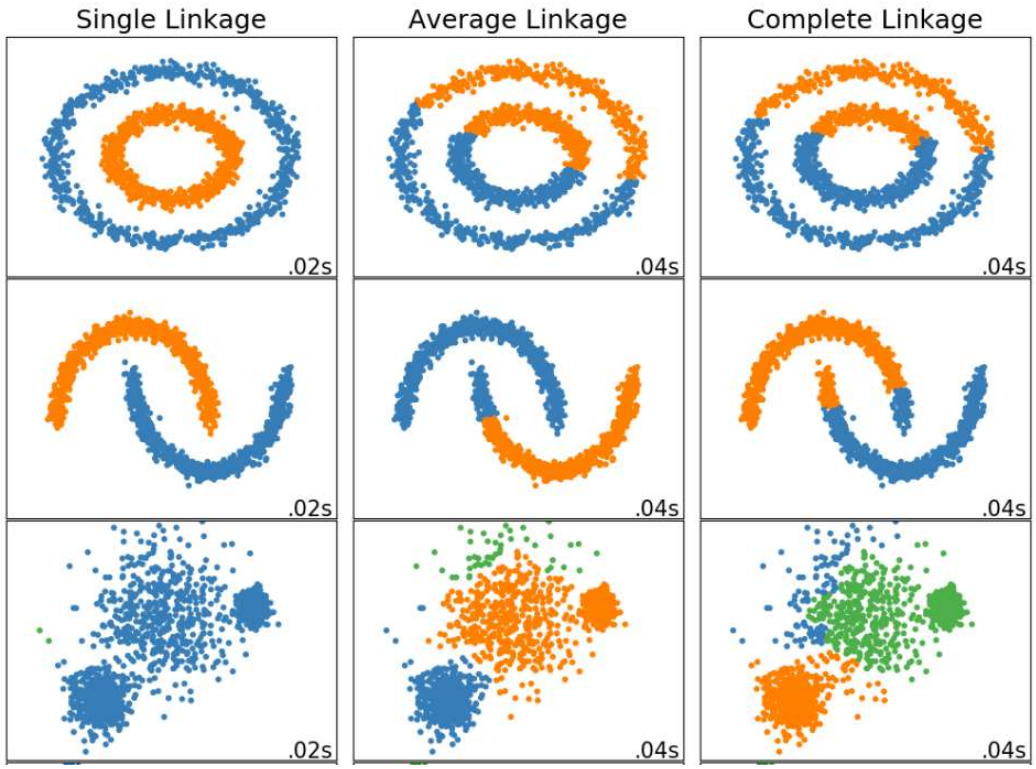


4. Ward link
   - minimal increase of sum-of-squares (MISSQ))

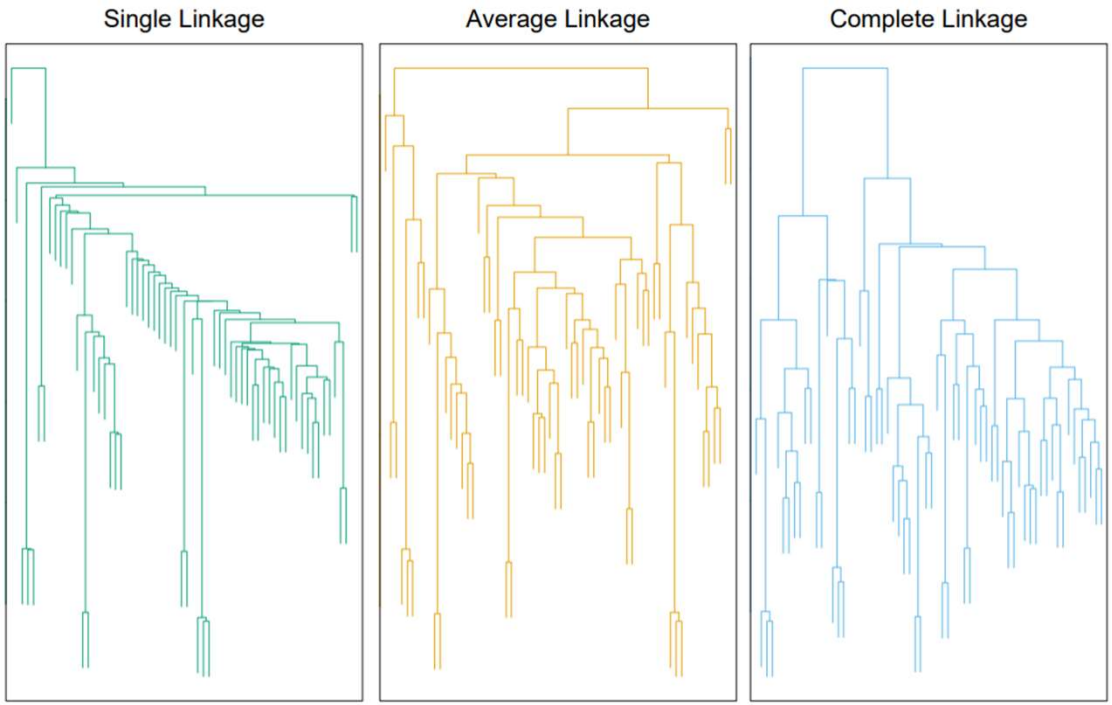


Ref: https://www.youtube.com/watch?v=vg1w5ZUF5lA
https://stats.stackexchange.com/questions/195446/choosing-the-right-linkage-method-for-hierarchical-clustering

เทียบ Mode or Cluster Distance Measures ของ Agglomerative



Source: https://scikit-learn.org/stable/modules/clustering.html



Dendrograms from agglomerative hierarchical clustering of human tumor microarray data. Pg 524 [Hastie, Tibshirani, Friedman, 2008]

sklearn.cluster.AgglomerativeClustering

sklearn.cluster.AgglomerativeClustering

class `sklearn.cluster.` **`AgglomerativeClustering`**(*n_clusters=2, *, affinity='euclidean', memory=None, connectivity=None, compute_full_tree='auto', linkage='ward', distance_threshold=None*) [source]

Agglomerative Clustering

Recursively merges the pair of clusters that minimally increases a given linkage distance.

Read more in the User Guide.

Parameters:

n_clusters : *int or None, default=2*

The number of clusters to find. It must be `None` if `distance_threshold` is not `None`.

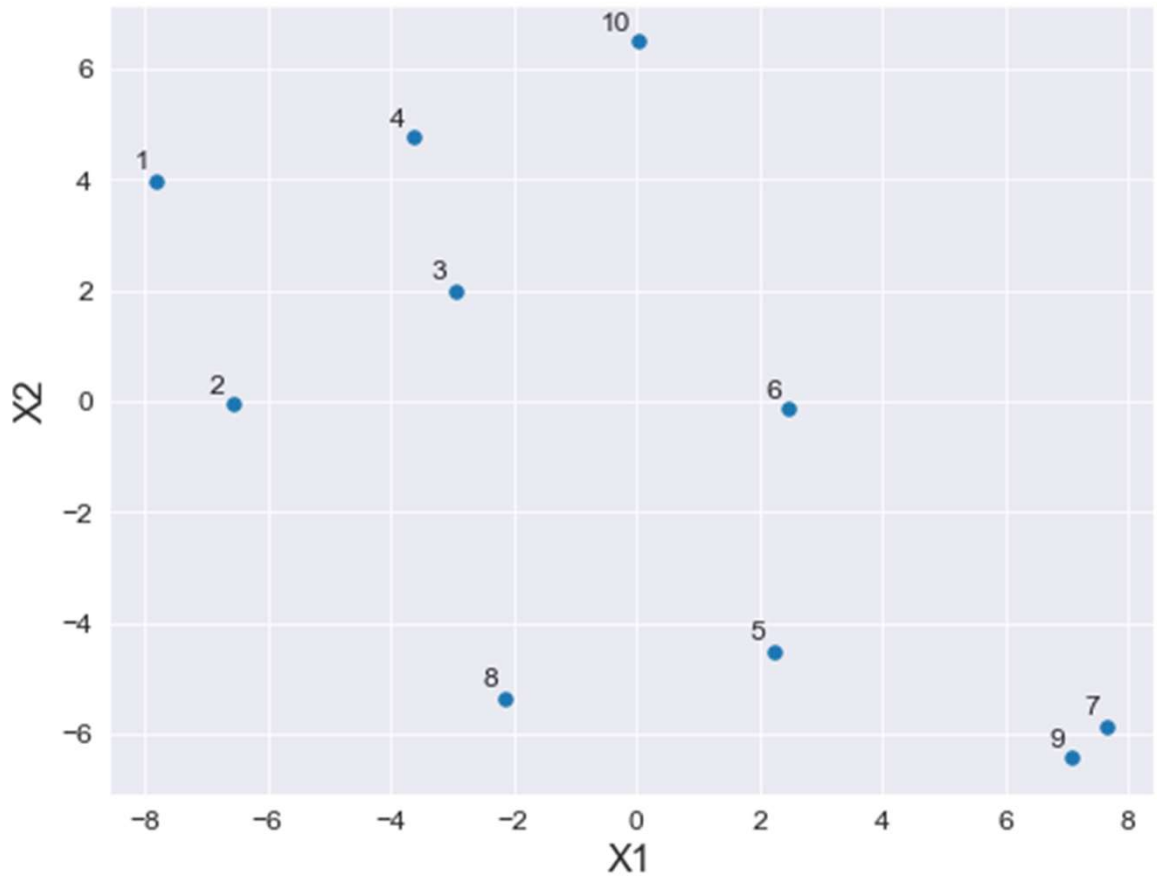
affinity : *str or callable, default='euclidean'*

Metric used to compute the linkage. Can be "euclidean", "l1", "l2", "manhattan", "cosine", or "precomputed". If linkage is "ward", only "euclidean" is accepted. If "precomputed", a distance matrix (instead of a similarity matrix) is needed as input for the fit method.

memory : *str or object with the joblib.Memory interface, default=None*

Used to cache the output of the computation of the tree. By default, no caching is done. If a string is given, it is the path to the caching directory.

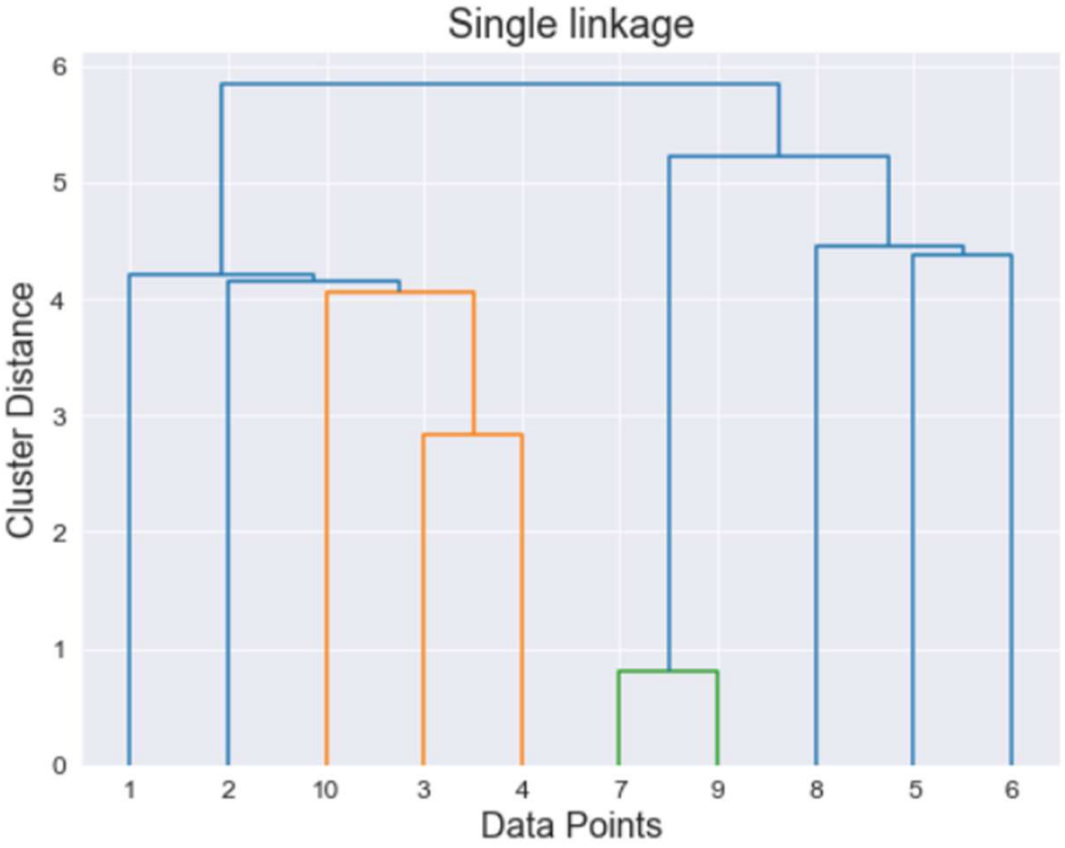
Try built dendrogram (AgglomerativeClustering)

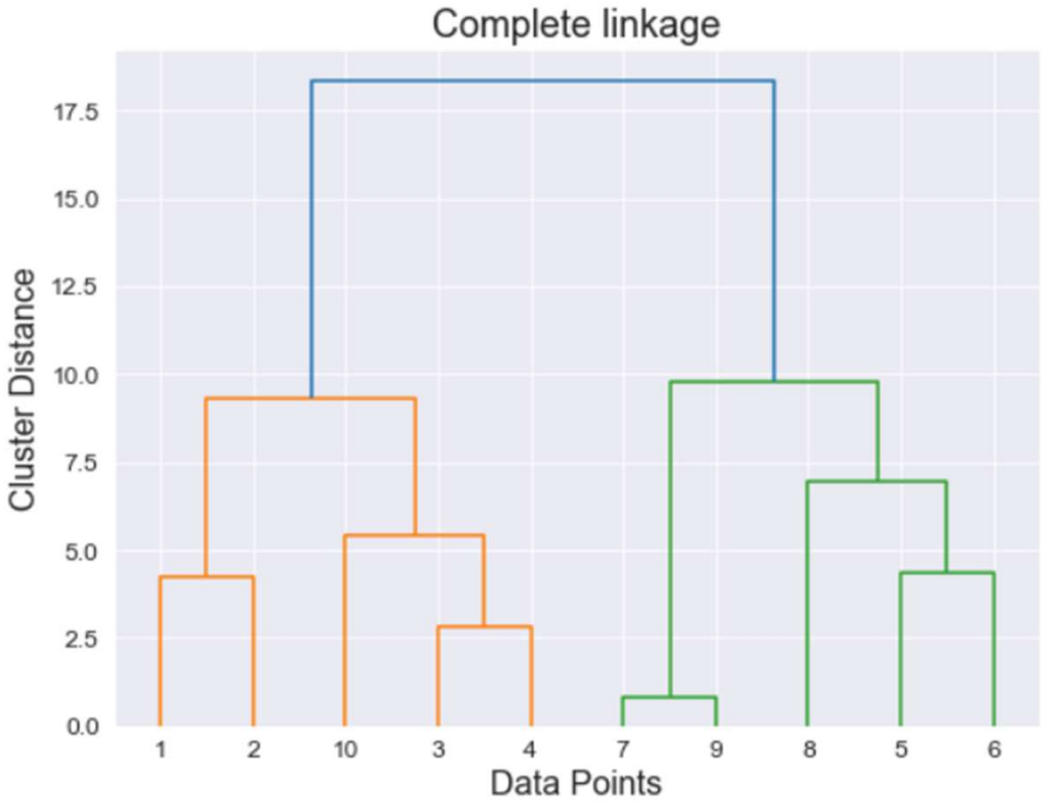


See dendrogram with ‘single’ linkage

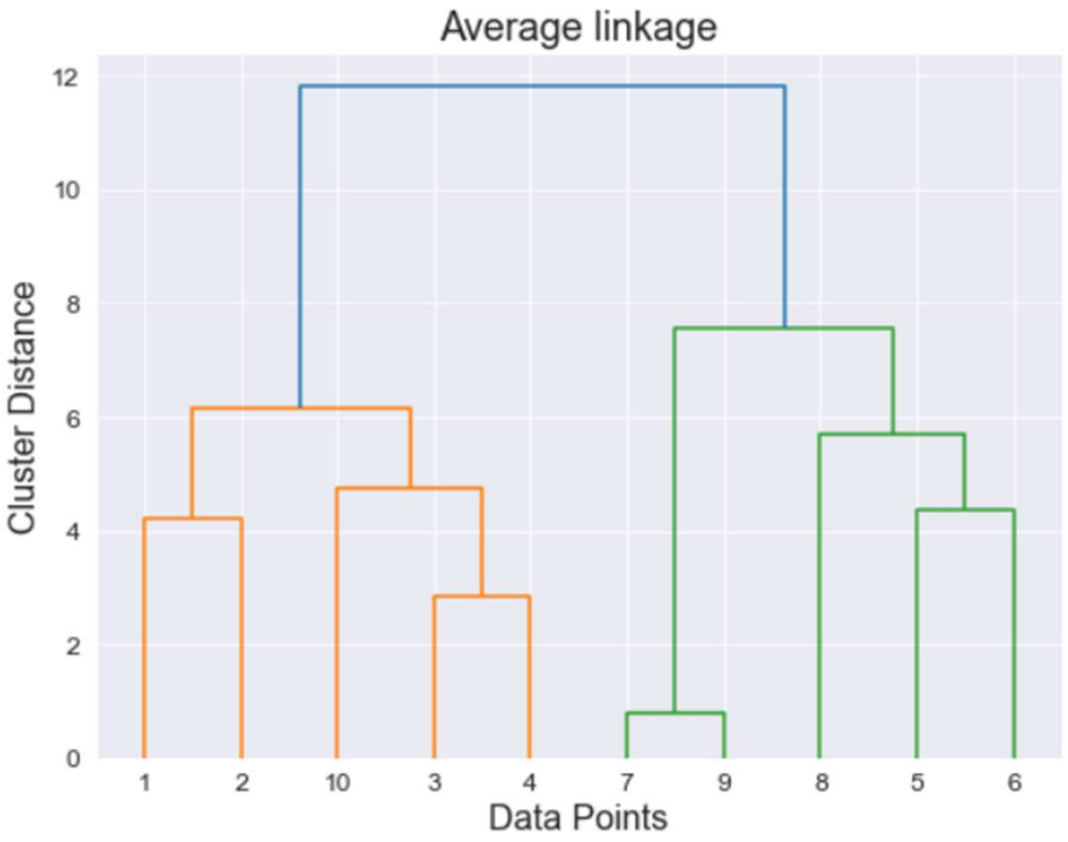
[24]:
```
# Import dendrogram and linkage library from scipy
from scipy.cluster.hierarchy import dendrogram, linkage

# Initial figure size
plt.figure(figsize=(8,6))

# Assign labels to represent all data points in X: here N points
labelList = range(1, N+1)

# Create Single-linkage of two points in X
linked = linkage(X, 'single')

# Compile dendrogram plot on linked X by using Euclidean distance
# Descending distance
dendrogram(linked,
           orientation='top',
           labels=labelList,
           distance_sort='descending',
           show_leaf_counts=True)

plt.title('Single linkage', fontsize=18)
plt.ylabel("Cluster Distance")
plt.xlabel("Data Points")
plt.show()
```

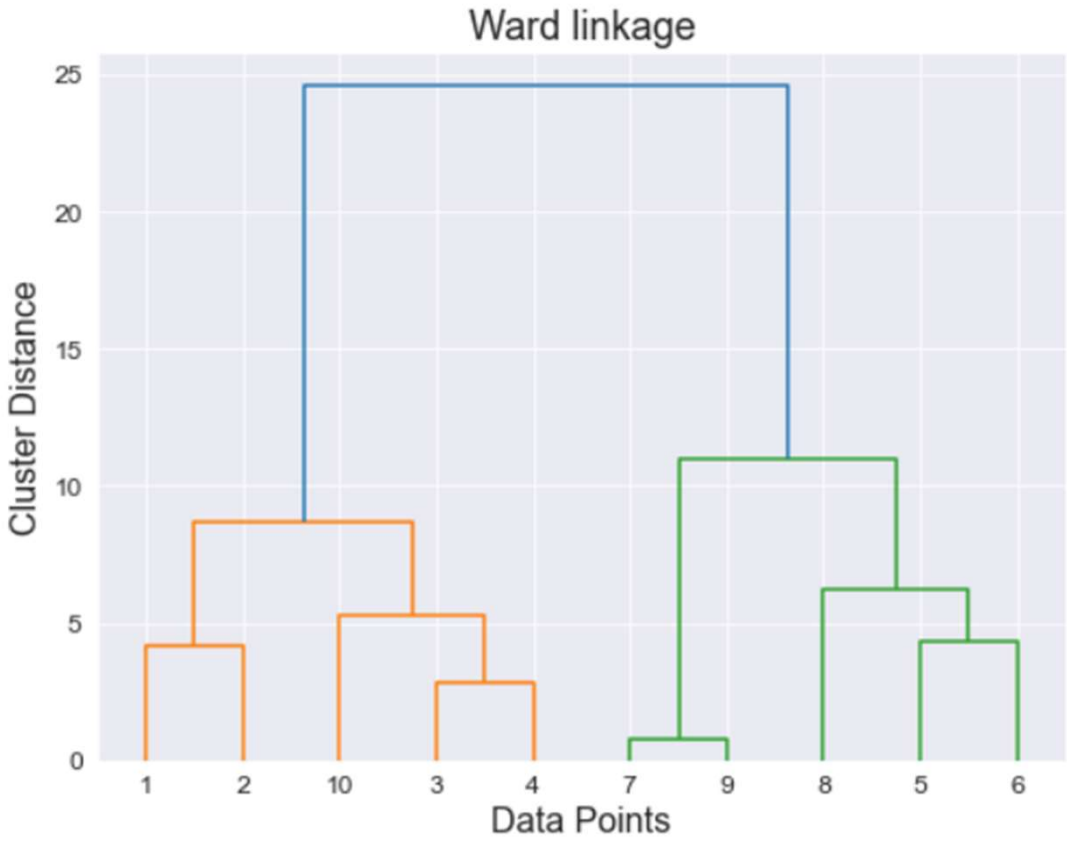
See dendrograms with ‘single’ and ‘complete’ linkages





See dendrograms with ‘average’ and ‘ward’ linkage





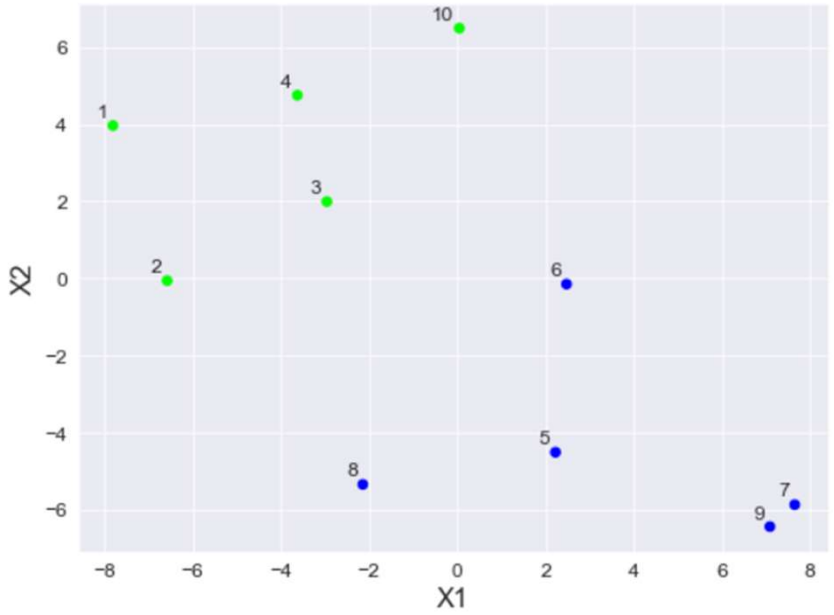
Now, let's do the actual clustering with AgglomerativeClustering

[29]:
```
from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n_clusters=2, linkage='ward')

# Call the fit_predict method to returns the cluster ID of each data point
cluster.fit_predict(X)

# Print labels
print(cluster.labels_)

# Plot Scatter
plt.figure(figsize=(8,6))
plt.scatter(X[:,0], X[:,1], c=cluster.labels_, s=30, cmap='brg')
plt.xlabel("X1")
plt.ylabel("X2")
for label, x, y in zip(labels, X[:, 0], X[:, 1]):
    plt.annotate(
        label,
        xy=(x,y), xytext=(-3,3),
        textcoords = 'offset points', ha='right', va='bottom')
plt.show()
```

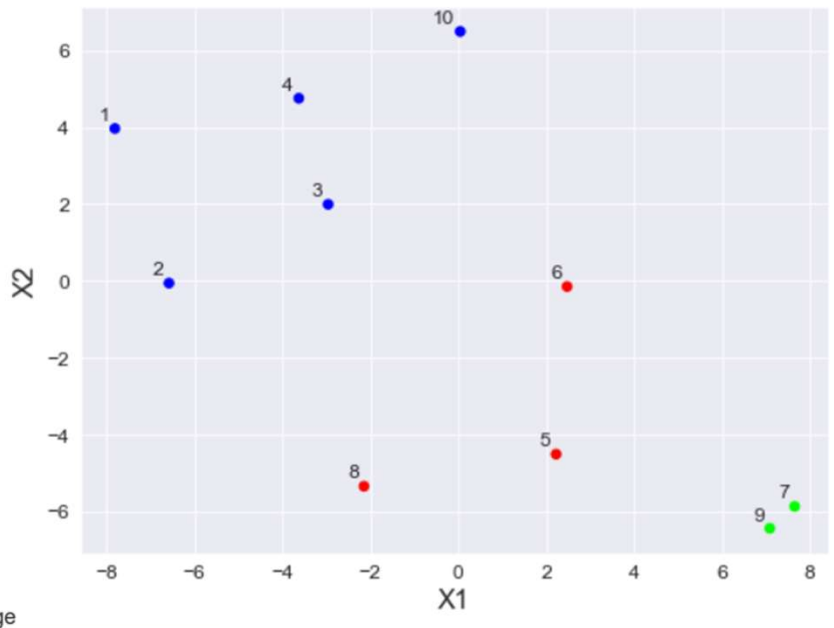
```
[1 1 1 1 0 0 0 0 0 1]
```

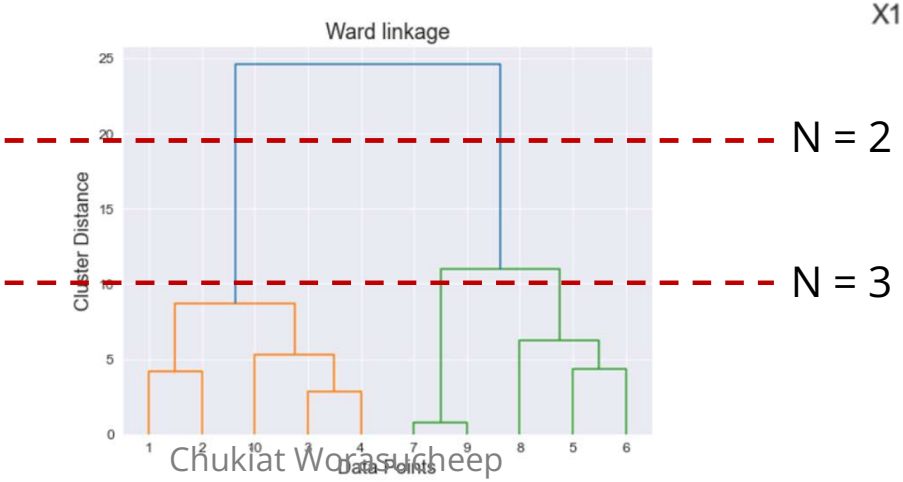
Compare the results using 2 (left) and 3 (right) clusters

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




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





Strengths and weaknesses of Agglomerative Clustering algorithm

- Strengths of AgglomerativeClustering Algorithm:
  - Useful for some underlying application requiring a hierarchy, e.g. creation of a taxonomy.
  - Can produce better-quality clusters

- Weaknesses of AgglomerativeClustering Algorithm:
  - Slow, high computing time using $O(n^2 + n^3)$, and storage requirement $O(n^2)$
  - Have some troubles with noisy, high-dimensional data, such as document data

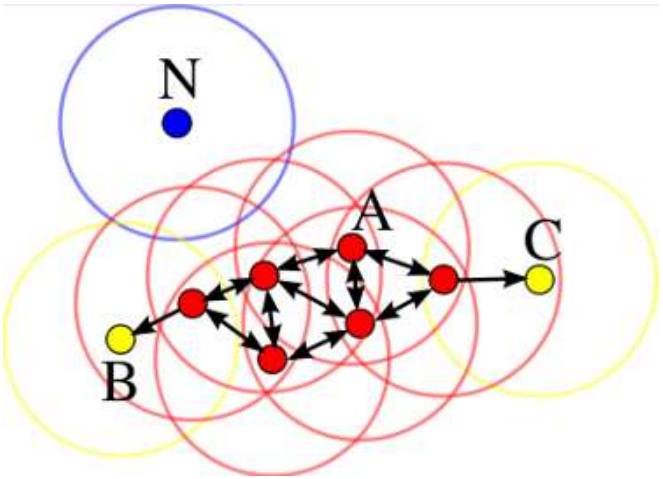
Outline

- Concept of cluster analysis
- K-mean clustering
- An application to customer segmentation
- Agglomerative clustering
- DBSCAN

DBSCAN

- Density-Based Spatial Clustering of Applications with Noise
- Two key parameters:
  - *eps*: The distance that specifies the neighborhoods. Two points are considered to be neighbors if the distance between them are less than or equal to eps.
  - *minPts*: Minimum number of data points to define a cluster.
- REPEAT
  - เลือกจุดที่ยังไม่ได้จัดเข้ากลุ่ม
  - พิจารณาระยะห่าง ε รอบๆ จุดนั้น
    - หากมีจุดมากถึง min_point ให้รวมจุดเหล่านั้นให้อยู่ในกลุ่มเดียวกัน และใช้จุดนั้นทำเช่นเดียวกัน
    - แต่หากจุดนั้นมีไม่ถึง min_point ในรัศมี ε ให้นับว่าเป็น noise
- UNTIL จะเพิ่มจุดไม่ได้ในกลุ่มนี้ และให้ไปเลือกจุดอื่นที่ยังไม่มีกลุ่ม

min_point = 4



Source: https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68

Chukiat Worasucheep

sklearn.cluster.DBSCAN

sklearn.cluster.DBSCAN

class `sklearn.cluster.` **DBSCAN**(*eps=0.5, *, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None*) [source]

Perform DBSCAN clustering from vector array or distance matrix.

DBSCAN - Density-Based Spatial Clustering of Applications with Noise. Finds core samples of high density and expands clusters from them. Good for data which contains clusters of similar density.

Read more in the User Guide.

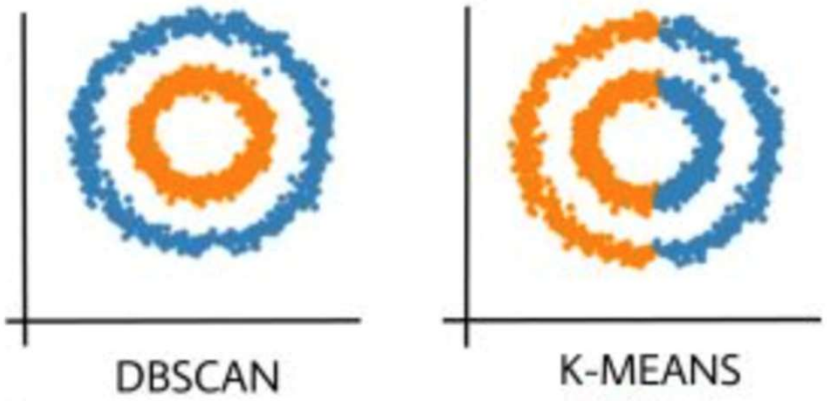
Parameters:

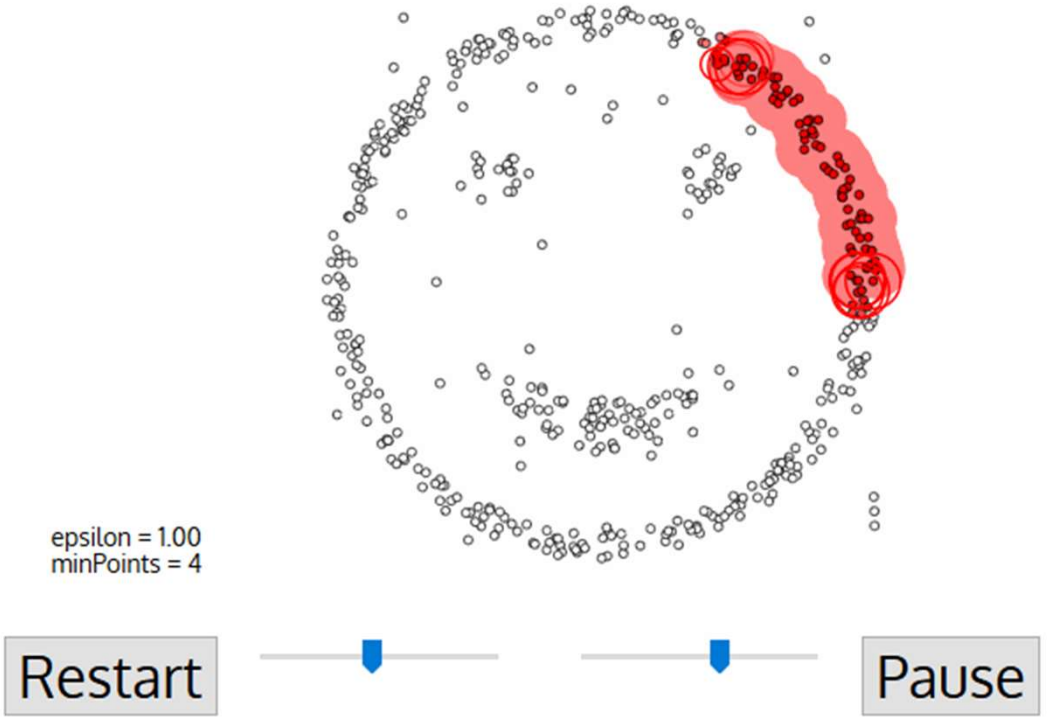
eps : *float, default=0.5*
The maximum distance between two samples for one to be considered as in the neighborhood of the other. This is not a maximum bound on the distances of points within a cluster. This is the most important DBSCAN parameter to choose appropriately for your data set and distance function.

min_samples : *int, default=5*
The number of samples (or total weight) in a neighborhood for a point to be considered as a core point. This includes the point itself.

metric : *string, or callable, default='euclidean'*
The metric to use when calculating distance between instances in a feature array. If metric is a string or callable, it must be one of the options allowed by `sklearn.metrics.pairwise_distances` for its metric parameter. If metric is "precomputed", X is assumed to be a distance matrix and must be square. X may be a Glossary, in which case only "nonzero" elements may be considered neighbors for DBSCAN.

DBSCAN vs K-Means in Some Complex Clustering





DBSCAN

- Advantages:
  - ไม่ต้องกำหนดจำนวนกลุ่ม
  - ได้ผลดีกับกลุ่มที่มีรูปร่างๆ แปลกๆ
  - ทนและสามารถหา outliner (noise) ได้เองอัตโนมัติ

- Disadvantages:
  - ต้องกำหนด ε, min_point ให้เหมาะสม
  - ไม่เหมาะกับกลุ่มมีการกระจายตัวหลายระดับ
  - มีความสุ่ม (จากการ random)

Criteria of Clustering Algorithms

- *Scalability*
  - to deal with large databases.
- *High dimensionality*
  - should able to handle the high dimensional space.
- *Ability to deal with different kinds of attributes*
  - such as interval-based (numerical) data, categorical, and binary data.
- *Ability to deal with noisy data*
  - Some algorithms are sensitive to noisy or missing data and may lead to poor quality clusters.
- *Interpretability*
  - The clustering results should be interpretable, comprehensible, and usable.

Source: https://www.tutorialspoint.com/data_mining/dm_cluster_analysis.htm

References

- Xu & Tian, A Comprehensive Survey of Clustering Algorithms, 2015.
- Velmurugan & Santhanam, A Survey of Partition based Clustering Algorithms in Data Mining: An Experimental Approach, 2011.
- https://www.gatevidyalay.com/k-means-clustering-algorithm-example/
- https://datafloq.com/read/7-innovative-uses-of-clustering-algorithms/6224
- https://www.tutorialspoint.com/data_mining/dm_cluster_analysis.htm
- https://en.wikipedia.org/wiki/Hierarchical_clustering
- https://towardsdatascience.com/dbscan-clustering-explained-97556a2ad556

Outline

- Concept of cluster analysis
- K-mean clustering
- An application to customer segmentation
- Agglomerative clustering
- DBSCAN