Unsupervised Learning – Part 4

ESM3081 Programming for Data Science

Seokho Kang

성균관대학교
SUNG KYUN KWAN UNIVERSITY(SKKU)

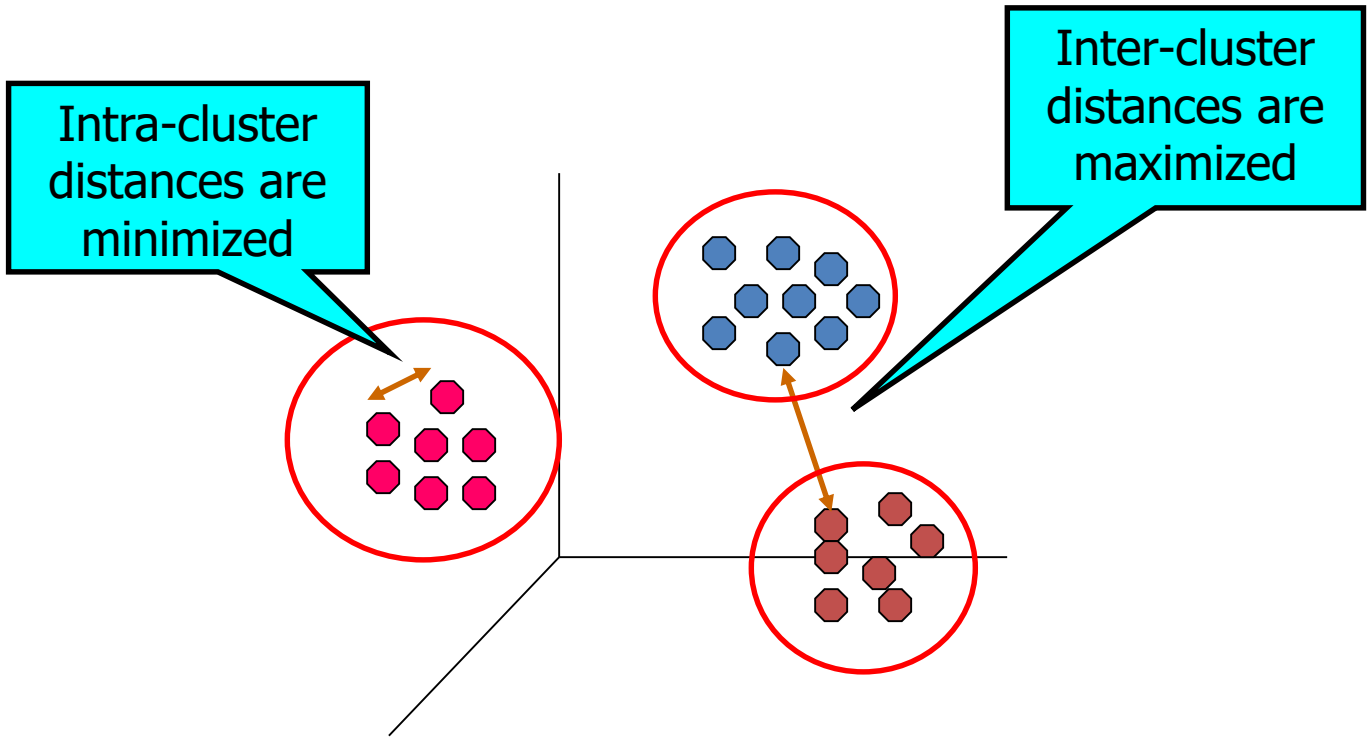
Learning algorithms covered in this course

- **Unsupervised Learning**
  - **Dimensionality Reduction & Visualization**
    - (Projection) Principal Component Analysis (PCA)
    - (Manifold Learning) t-distributed Stochastic Neighbor Embedding (t-SNE)
    - …
  - **Clustering**
    - **K-Means**
    - **Hierarchical Clustering**
    - DBSCAN
    - …

K-Means Clustering

Clustering

- ***Clustering* is the task of partitioning the dataset into groups, called clusters.**
  - The goal is to split up the data in such a way that points within a single cluster are very similar and points in different clusters are different.
  - Clustering is an exploratory tool, and is useful only when it produces meaningful clusters.
  - Data may not have definitive "real" clusters, so we need to be wary of chance results.
  - **Applications of Clustering:** *Data Understanding and Summarization (Compression)*



K-Means Clustering

- ***k*-means clustering finds *k cluster centers* that are representative of certain regions of the data based on an expectation-maximization procedure.**

  Given a (training) dataset $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n\}$ such that $\boldsymbol{x}_i = (x_{i1}, \dots, x_{id}) \in \mathbb{R}^d$ is the *i*-th input vector of *d* features

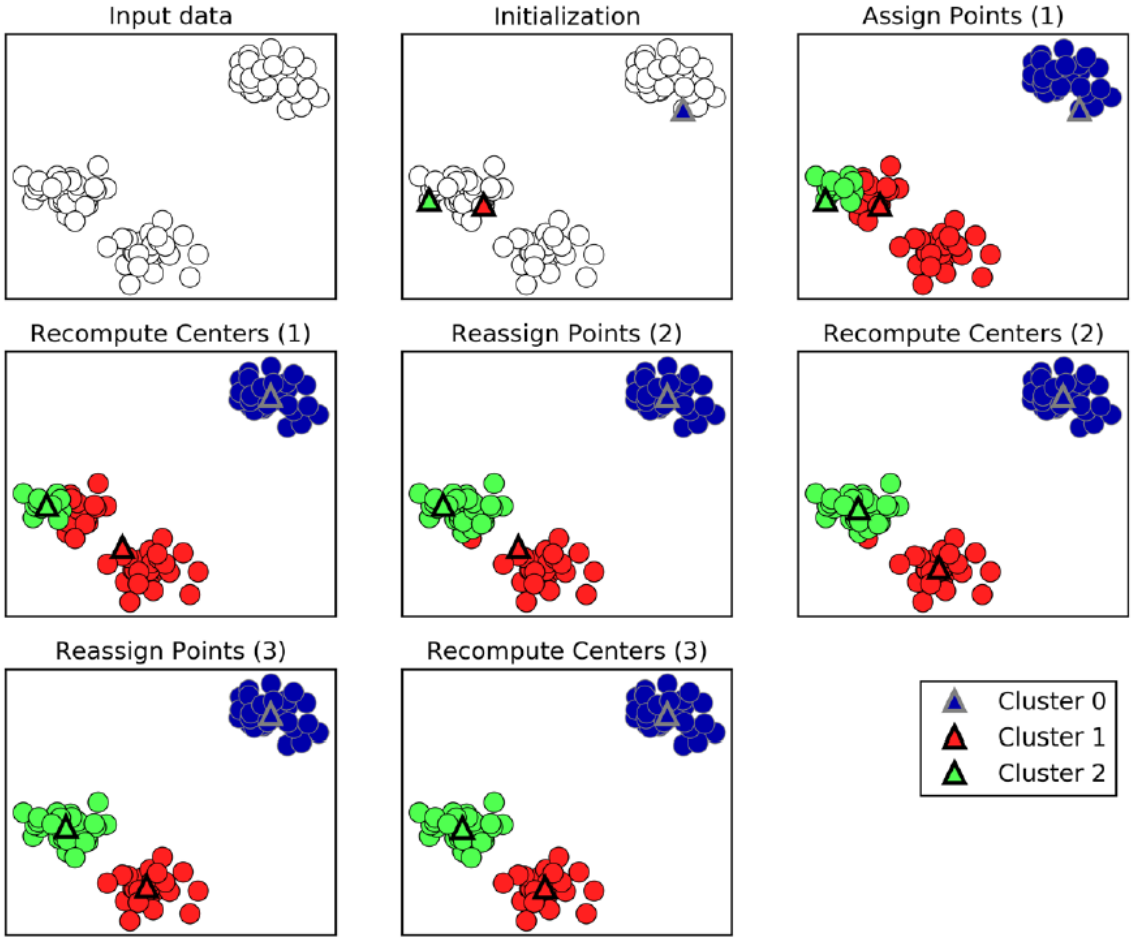
  1. The algorithm starts by selecting *k* points as the initial cluster centers.
  2. Then, alternates between two steps. It is finished when the assignment of data points to clusters no longer changes.
     - **(Expectation)** Assigning each data point to the closest cluster centers
     - **(Maximization)** Setting each cluster center as the mean of the data points that are assigned to it

       * Euclidean distance is used by default.

K-Means Clustering

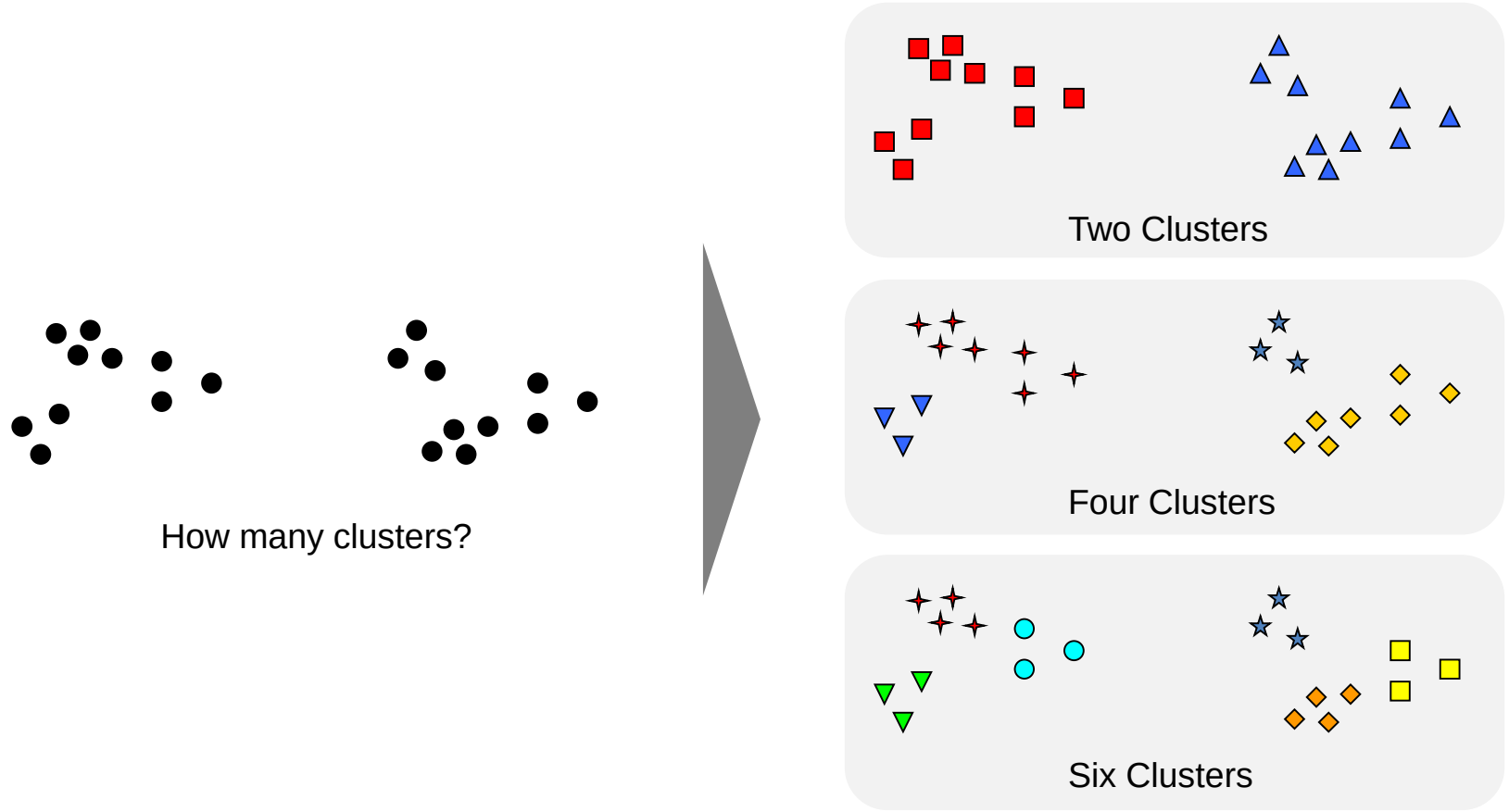
- **Example of *k*-means clustering**

  Example: Cluster centers are shown as triangles, while data points are shown as circles, Colors indicate cluster membership, We are looking for three clusters (k=3).



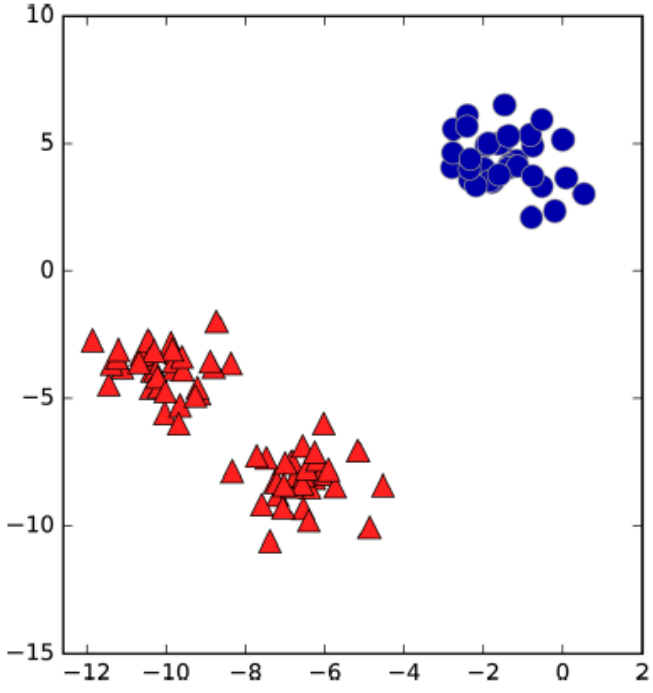
K-Means Clustering

- **Main hyperparameter**: the number of clusters *k*
  - Notion of a cluster can be ambiguous

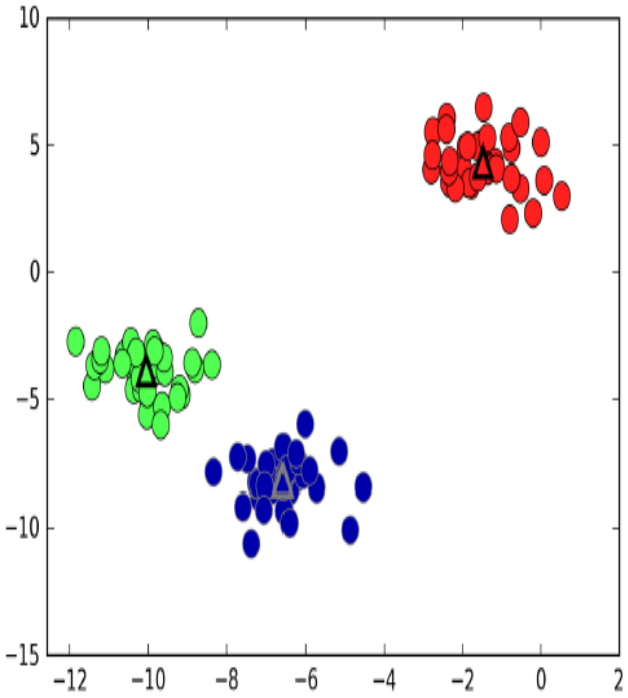


K-Means Clustering

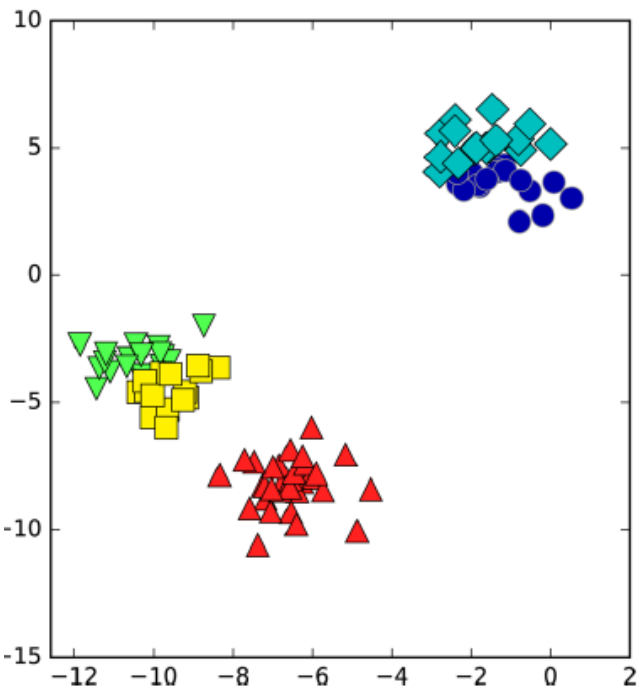
- **Main hyperparameter**: the number of clusters *k*
  - The effect of the hyperparameter *k* (n_clusters)
    Example: Cluster assignments and cluster centers found by k-means with different numbers of clusters



k = 2



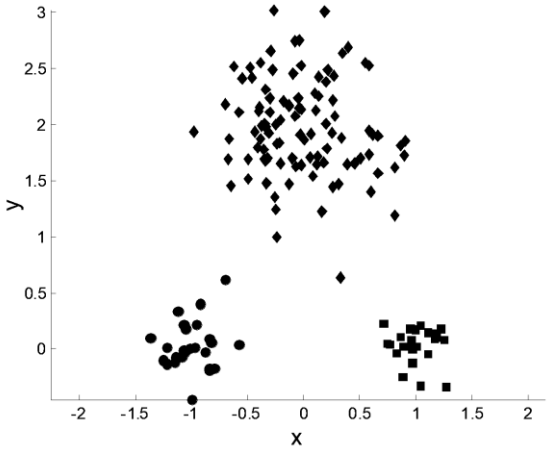
k = 3



k = 5

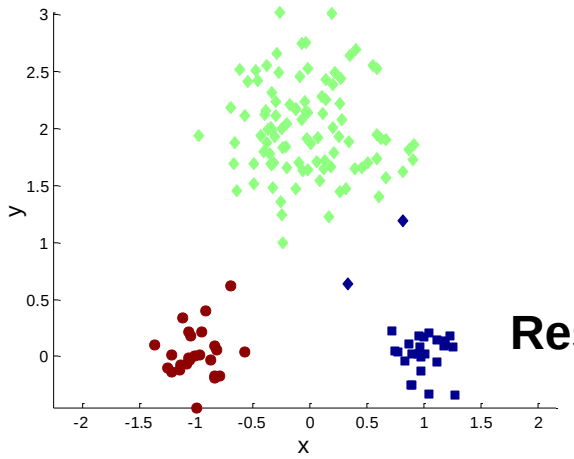
K-Means Clustering

- **The effect of random initialization**
  - It is recommended to run the algorithm multiple times with different random initializations, and returns the best result.



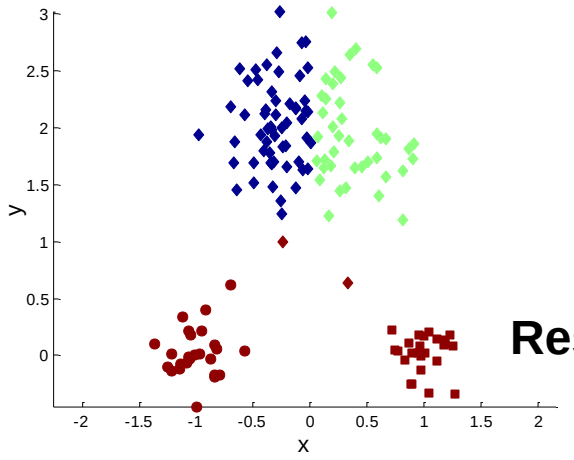
Original Points

1st run



Result 1

2nd run



Result 2

scikit-learn Practice: *KMeans*

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

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

K-Means clustering.

n_clusters	*int, default=8* The number of clusters to form as well as the number of centroids to generate. For an example of how to choose an optimal value for n_clusters refer to Selecting the number of clusters with silhouette analysis on KMeans clustering.

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

scikit-learn Practice: *KMeans*

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

Methods	
fit(*X*, *y*=*None*, *sample_weight*=*None*)	Compute k-means clustering.
fit_predict(*X*, *y*=*None*, *sample_weight*=*None*)	Compute cluster centers and predict cluster index for each sample. Convenience method; equivalent to calling fit(X) followed by predict(X).
fit_transform(*X*, *y*=*None*, *sample_weight*=*None*)	Compute clustering and transform X to cluster-distance space. Equivalent to fit(X).transform(X), but more efficiently implemented.
predict(*X*)	Predict the closest cluster each sample in X belongs to. In the vector quantization literature, cluster_centers_ is called the code book and each value returned by predict is the index of the closest code in the code book.
transform(*X*)	Transform X to a cluster-distance space. In the new space, each dimension is the distance to the cluster centers. Note that even if X is sparse, the array returned by transform will typically be dense.

scikit-learn Practice: *KMeans*

- **Example (*blobs* dataset)**

[1]:
```
from sklearn.datasets import make_blobs
from sklearn.cluster import Kmeans

X_train, _ = make_blobs(random_state=1)
print('X_train.shape:', X_train.shape)
```

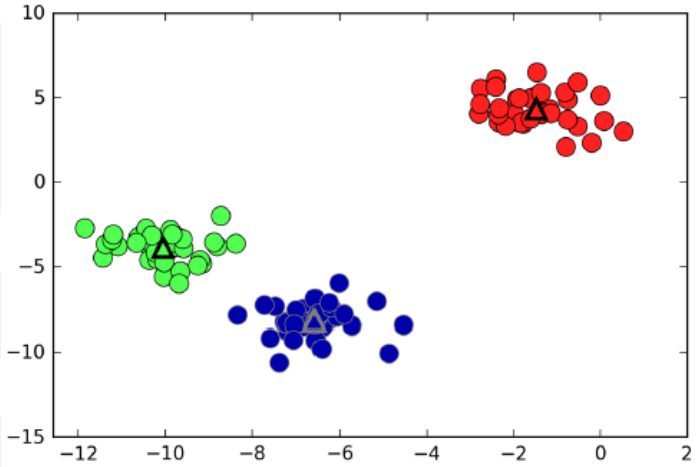
```
X_train.shape: (100, 2)
```

[2]:
```
kmeans = KMeans(n_clusters=3)
kmeans.fit(X_train)
```

```
KMeans(n_clusters=3)
```

[3]:
```
print(kmeans.cluster_centers_)
```

```
[[ -6.58197  -8.17239]
 [ -1.47108   4.33722]
 [-10.04935  -3.85954]]
```



[4]:
```
assignments_X_train = kmeans.labels_
print(assignments_X_train)
```

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

[5]:
```
X_new, _ = make_blobs()
assignments_X_new = kmeans.predict(X_new)
print(assignments_X_new)
```

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

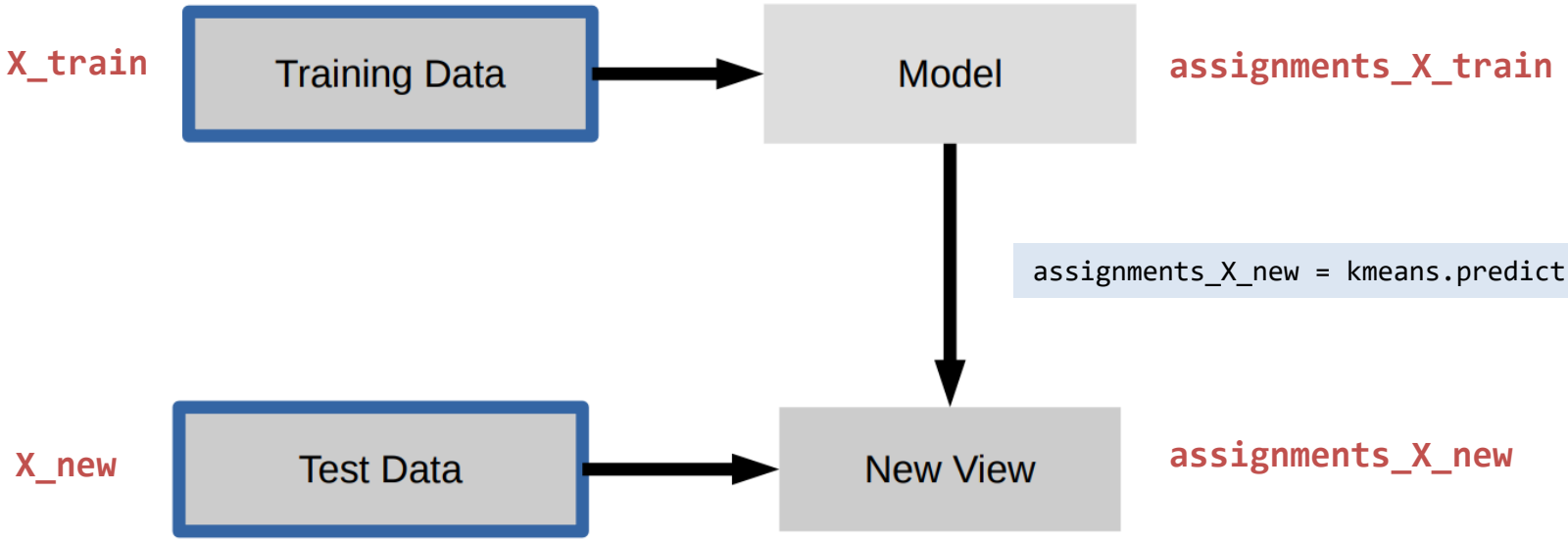
scikit-learn Practice: *KMeans*

- **Example (*blobs* dataset)**

```
X_train, _ = make_blobs(random_state=1)
```

```
kmeans = KMeans(n_clusters=3)
kmeans.fit(X_train)
```

```
assignments_X_train = kmeans.labels_
```



scikit-learn Practice: *KMeans*

Given the knowledge of the ground truth class assignments **labels_true** and our clustering algorithm assignments of the same data points **labels_pred**, the **adjusted Rand index** is a function that measures the **similarity** of the two assignments

- Score between -1.0 and 1.0.
- Random labelings have an ARI close to 0.0.
- 1.0 stands for perfect match.

```
sklearn.metrics.adjusted_rand_score(labels_true, labels_pred)
```

scikit-learn Practice: *KMeans*

Examples

Perfectly matching labelings have a score of 1 even

```
>>> from sklearn.metrics.cluster import adjusted_rand_score
>>> adjusted_rand_score([0, 0, 1, 1], [0, 0, 1, 1])
1.0
>>> adjusted_rand_score([0, 0, 1, 1], [1, 1, 0, 0])
1.0
```

Labelings that assign all classes members to the same clusters are complete but may not always be pure, hence penalized:

```
>>> adjusted_rand_score([0, 0, 1, 2], [0, 0, 1, 1])
0.57...
```

ARI is symmetric, so labelings that have pure clusters with members coming from the same classes but unnecessary splits are penalized:

```
>>> adjusted_rand_score([0, 0, 1, 1], [0, 0, 1, 2])
0.57...
```

If classes members are completely split across different clusters, the assignment is totally incomplete, hence the ARI is very low:

```
>>> adjusted_rand_score([0, 0, 0, 0], [0, 1, 2, 3])
0.0
```

scikit-learn Practice: *KMeans*

- **Example (*iris* dataset)**

[1]:
```
from sklearn.datasets import load_iris
from sklearn.cluster import Kmeans
from sklearn.metrics import adjusted_rand_score

iris = load_iris()
X_train, y_train = iris.data, iris.target
print('X_train.shape:', X_train.shape)
```

```
X_train.shape: (150, 4)
```

[2]:
```
kmeans = KMeans(n_clusters=3)
kmeans.fit(X_train)
```

```
KMeans(n_clusters=3)
```

[3]:
```
print(kmeans.cluster_centers_)
```

```
[[5.006   3.428   1.462   0.246  ]
 [6.85385 3.07692 5.71538 2.05385]
 [5.88361 2.74098 4.38852 1.43443]]
```

[4]:
```
assignments_X_train = kmeans.labels_
print('adjusted_rand_score: %.5f'%adjusted_rand_score(y_train, assignments_X_train))
```

```
adjusted_rand_score: 0.71634
```

scikit-learn Practice: *KMeans*

- **Example (*iris* dataset)**

[1]:
```
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import Kmeans
from sklearn.metrics import adjusted_rand_score

iris = load_iris()
X_train, y_train = iris.data, iris.target
print('X_train.shape:', X_train.shape)
```

```
X_train.shape: (150, 4)
```

[2]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

[3]:
```
kmeans = KMeans(n_clusters=3)
kmeans.fit(X_train_scaled)
```

```
KMeans(n_clusters=3)
```

[4]:
```
print(kmeans.cluster_centers_)
```

```
[[5.70435 2.63478 4.21522 1.33261]
 [5.01633 3.45102 1.46531 0.2449 ]
 [6.69636 3.06    5.41818 1.93818]]
```

[5]:
```
assignments_X_train = kmeans.labels_
print('adjusted_rand_score: %.5f'%adjusted_rand_score(y_train, assignments_X_train))
```

```
adjusted_rand_score: 0.64515
```

Discussion

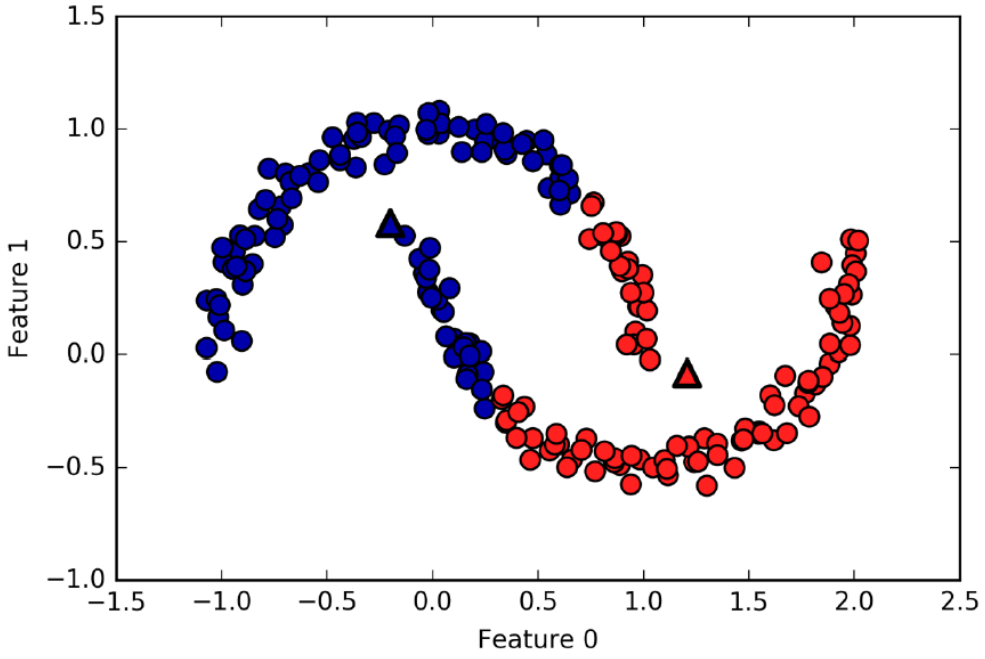
- **The main hyperparameters of K-Means**
  - *n_clusters (the number of clusters k), distance metric*

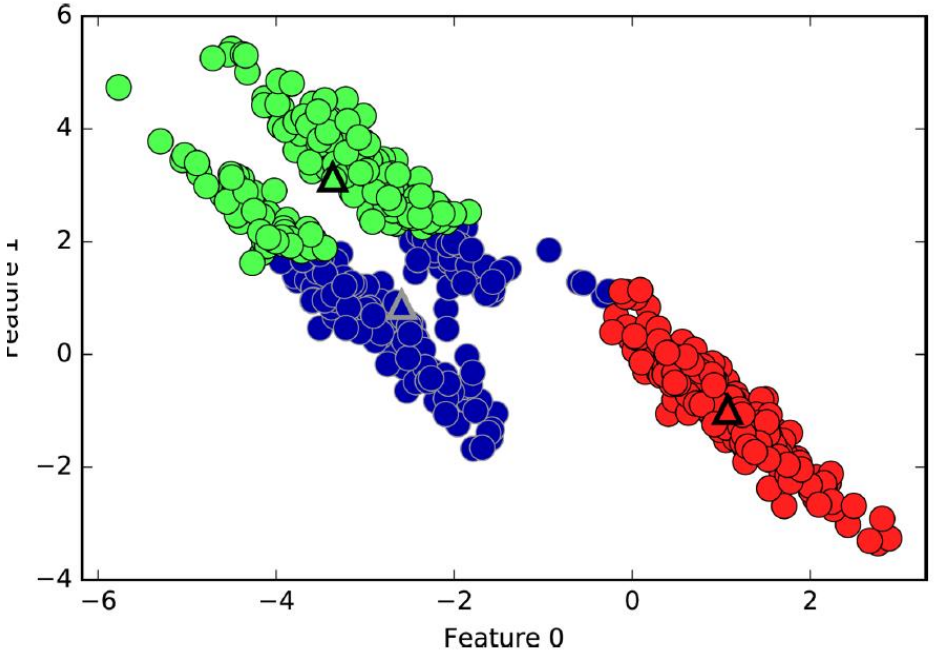
  * In scikit-learn, Euclidean distance is used by default.

  * It's important to preprocess your data (including *feature scaling* and *one-hot encoding*)
- **Strengths**
  - K-Means is relatively easy to understand and implement
  - It runs relatively quickly, and scales easily to large datasets
- **Weaknesses**
  - It relies on a random initialization, which means the outcome of the algorithm depends on a random seed.
  - It requires to specify the number of clusters you are looking for (which might not be known in a real-world application).
  - The performance depends highly on scaling of features.
  - The relatively restrictive assumptions are made on the shape of clusters.

Failure Cases of K-Means

- K-Means can only capture relatively simple (convex) shapes, as each cluster is defined solely by its center.
- K-Means fails to identify non-spherical or complex shaped clusters.
- Even if you know the “right” number of clusters for a given dataset, K-Means might not always be able to recover them.

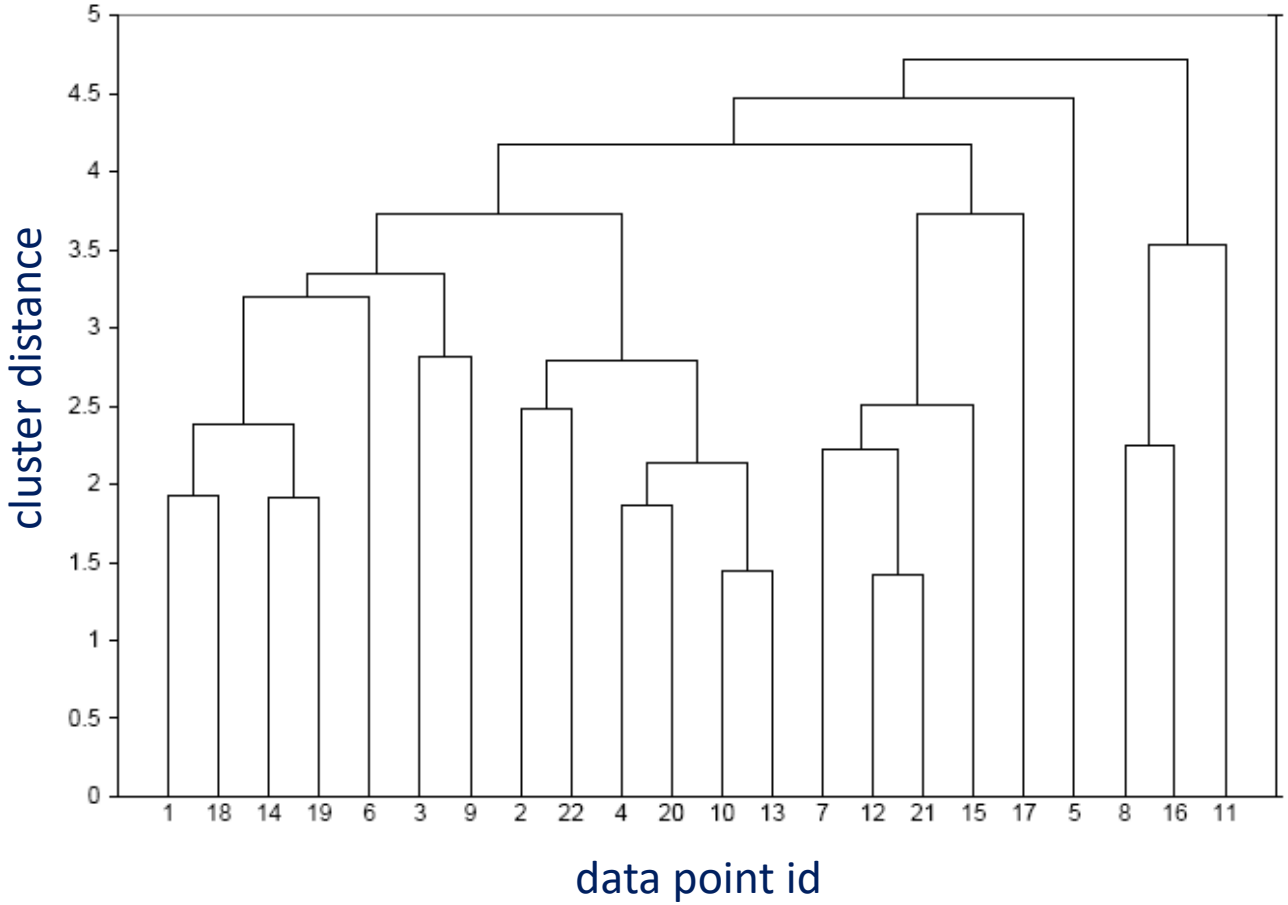




Hierarchical Clustering

Hierarchical Clustering

- **Hierarchical clustering produces a set of nested clusters organized as a hierarchical tree, which can be visualized as a dendrogram**
  - Dendrogram is a tree like diagram that records the sequences of merges or splits



Hierarchical Clustering

- **Two main types of hierarchical clustering**
  - **Agglomerative clustering (most popular)**
    - Start with the data points as individual clusters
    - At each step, merge the closest pair of clusters until only one cluster (or k clusters) left
  - **Divisive clustering**
    - Start with one all-inclusive cluster (the entire dataset as a cluster)
    - At each step, split a cluster until each cluster contains an individual point (or there are k clusters)

Agglomerative Clustering

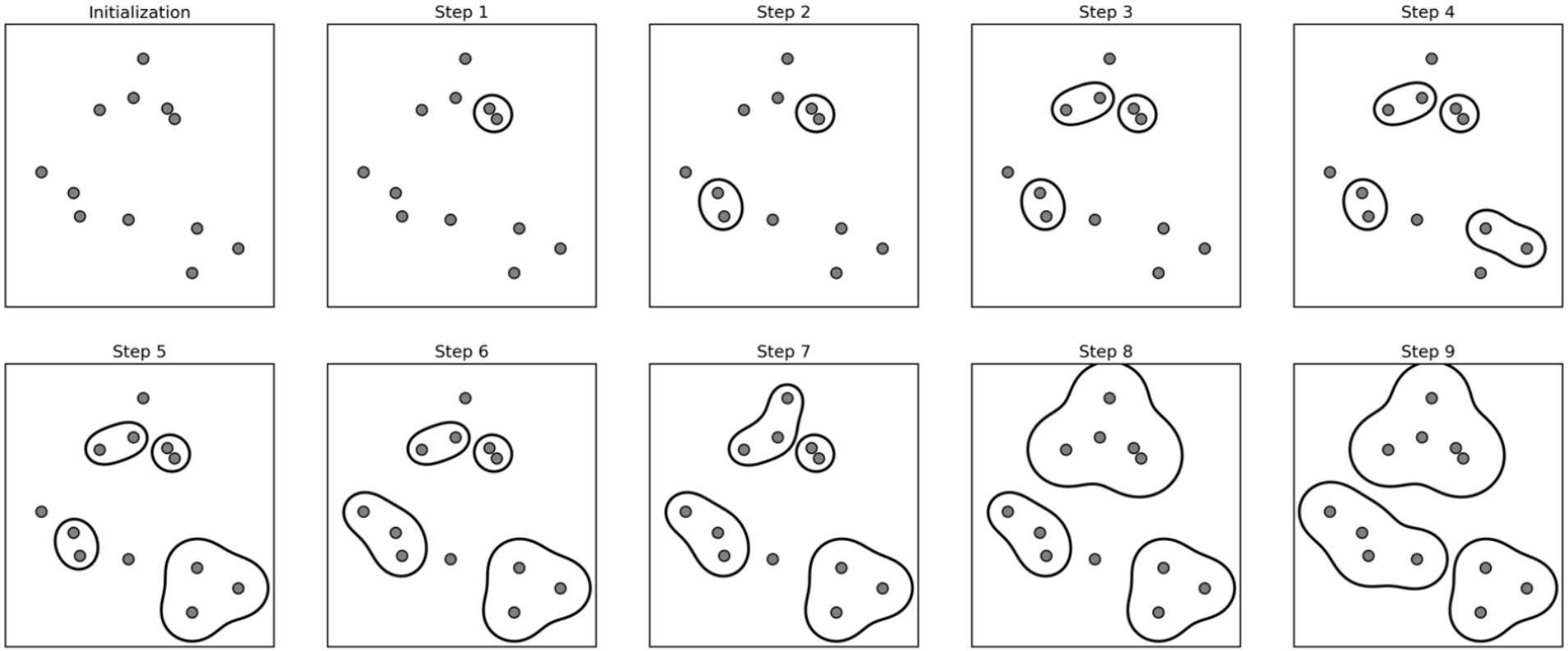
- ***Agglomerative clustering*** **refers to a collection of clustering algorithms that all build upon the following principles**

  Given a (training) dataset $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n\}$ such that $\boldsymbol{x}_i = (x_{i1}, \dots, x_{id}) \in \mathbb{R}^d$ is the *i*-th input vector of *d* features

  1. The algorithm starts by declaring each point its own cluster.
  2. Then, merges the two most similar clusters until some stopping criterion is satisfied.

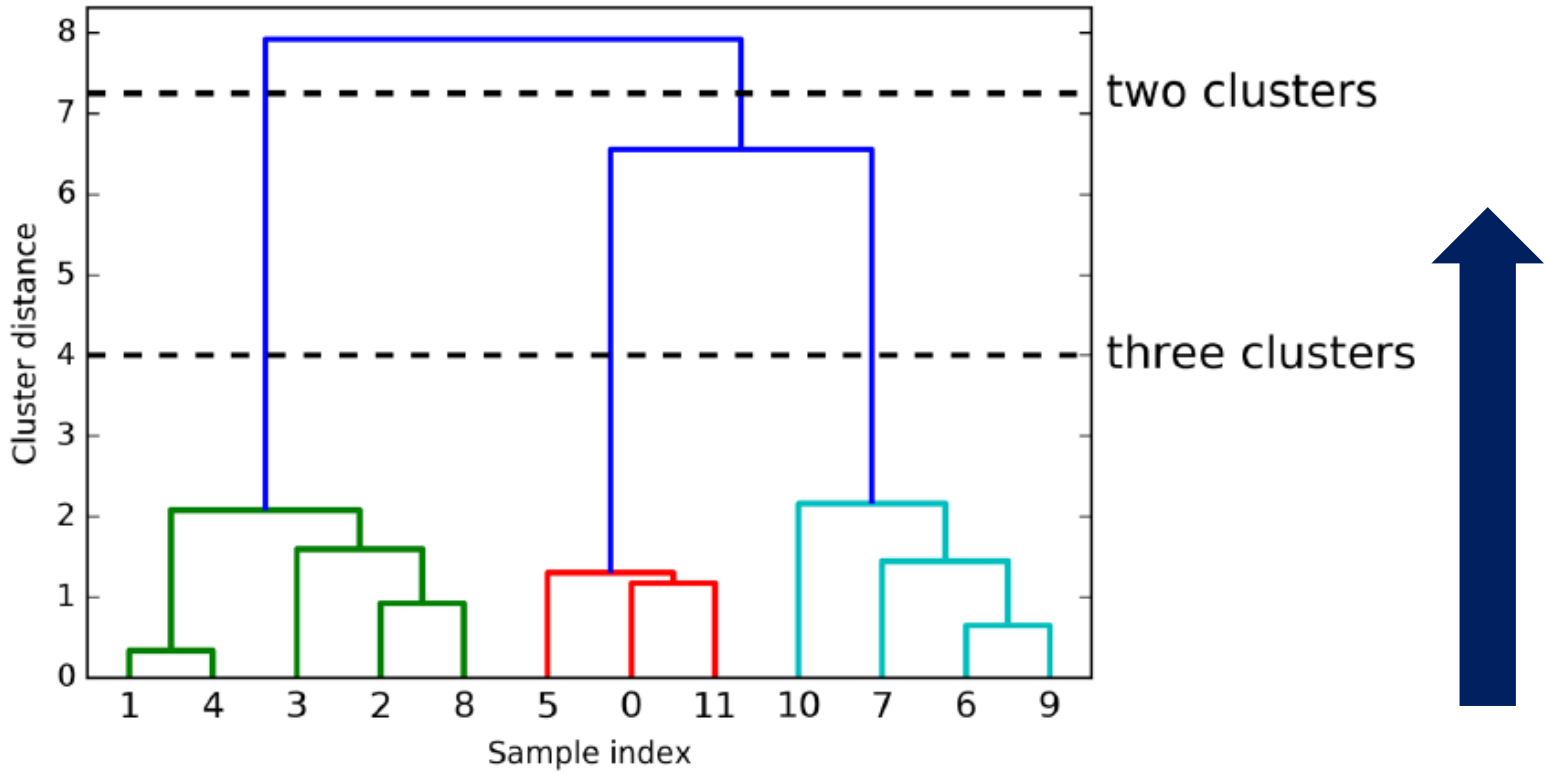
Agglomerative Clustering

- **Example of agglomerative clustering**
  - Initially, each point is its own cluster.
  - Then, in each step, the two clusters that are closest are merged.



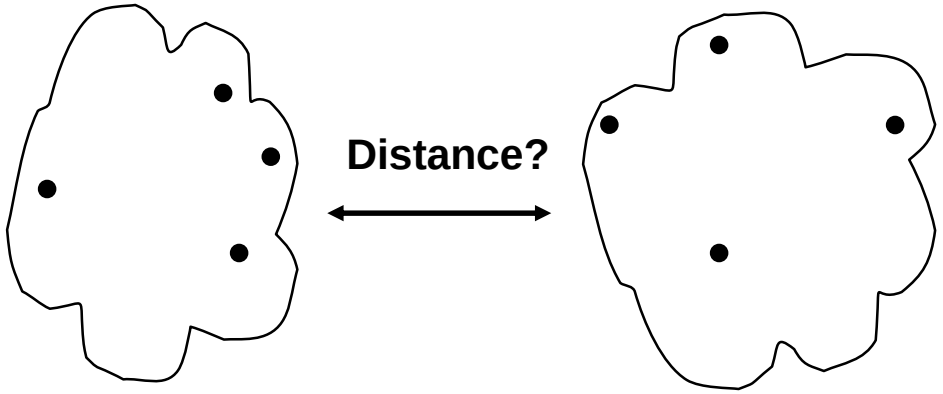
Agglomerative Clustering

- **Determining the number of clusters**
  - **Dendrogram** is helpful to look at all possible clusterings jointly
  - Each intermediate step provides a clustering of the data (with a different number of clusters).
  - A horizontal line intersects the clusters that are that far apart, to create clusters



Agglomerative Clustering

- There are several ***linkage*** criteria to evaluate the distance between two clusters.
  - This specifies how exactly the “most similar cluster” is selected.
  - Single (Minimum), Complete (Maximum), Average, Ward

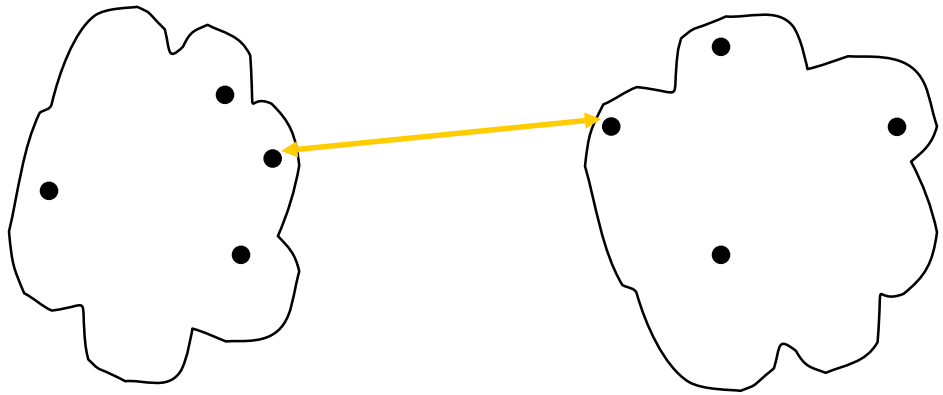


* Ward works on most datasets

* If the clusters have very dissimilar numbers of members (e.g., one is much bigger than all the others), average or complete might work better.

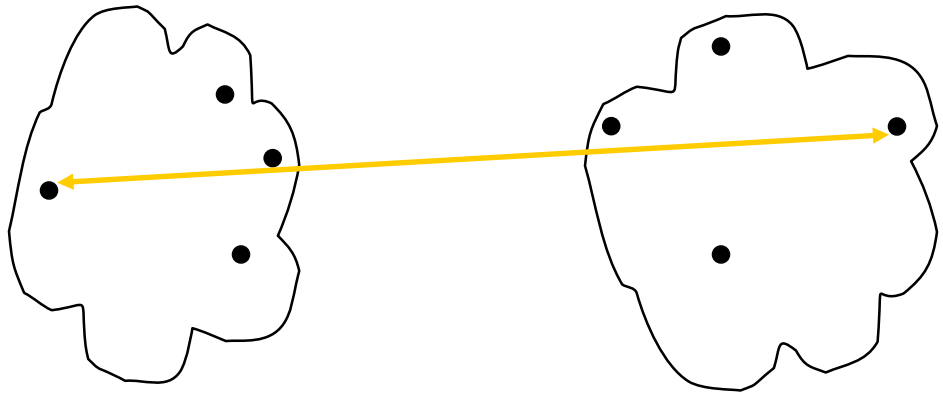
Agglomerative Clustering

- **Single (Minimum) Linkage**: the minimum of the distances between all data points of the two clusters.
  - Can handle non-globular shapes
  - Sensitive to noise and outliers



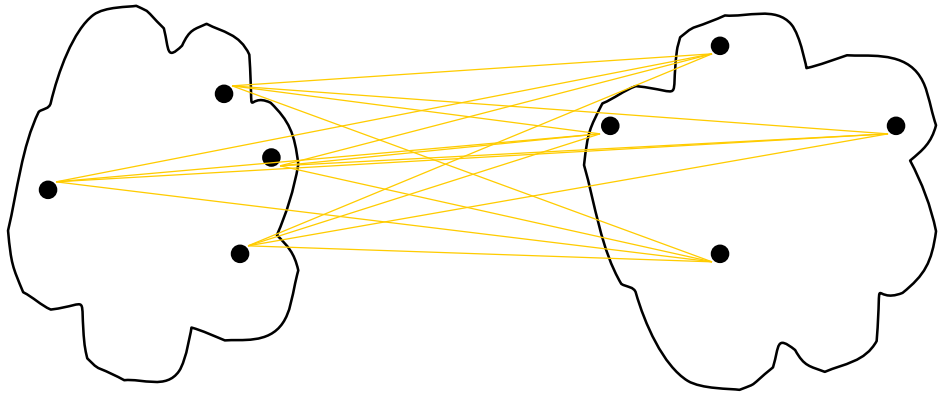
Agglomerative Clustering

- **Complete (Maximum) Linkage**: the maximum of the distances between all data points of the two clusters.
  - Tends to break large clusters
  - Biased towards globular clusters



Agglomerative Clustering

- **Average Linkage**: the average of the distances between all data points of the two clusters.
  - Compromise between Single and Complete Linkage
  - Less susceptible to noise and outliers
  - Biased towards globular clusters

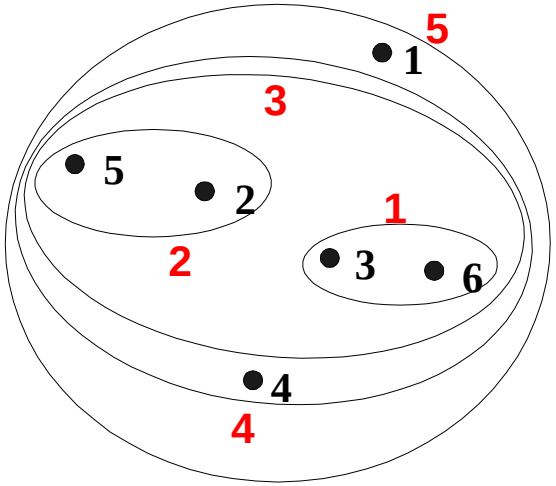


Agglomerative Clustering

- **Ward Linkage**: the minimum increase in variance when the two clusters are merged.
  - Less susceptible to noise and outliers
  - Biased towards globular clusters
  - Often leads to clusters that are relatively equally sized
  - Hierarchical analogue of K-means
  - The default choice in scikit-learn

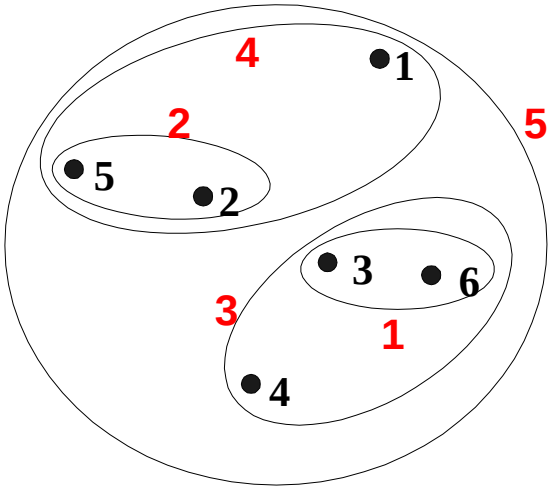
Agglomerative Clustering

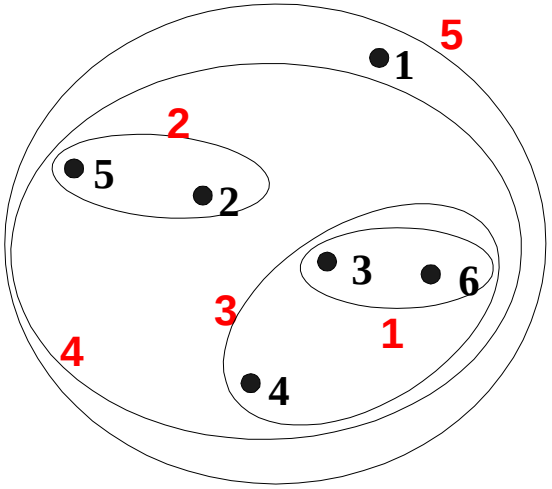
- **Example of Agglomerative Clustering with Different Linkages**



Single (Minimum)

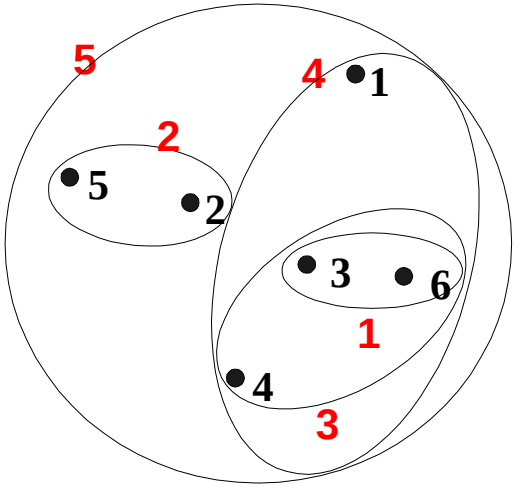
Complete (Maximum)





Average

Ward



scikit-learn Practice: *AgglomerativeClustering*

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

```
class sklearn.cluster.AgglomerativeClustering(n_clusters=2, *, metric='euclidean',
memory=None, connectivity=None, compute_full_tree='auto', linkage='ward',
distance_threshold=None, compute_distances=False)
```

Agglomerative Clustering.

Recursively merges pair of clusters of sample data; uses linkage distance.

*** The stopping criteria implemented in scikit-learn are *n_clusters* and *distance_threshold*.**

n_clusters	*int or None, default=2* The number of clusters to find. It must be None if distance_threshold is not None.
metric	*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 is needed as input for the fit method. If connectivity is None, linkage is "single" and affinity is not "precomputed" any valid pairwise distance metric can be assigned.
linkage	*{'ward', 'complete', 'average', 'single'}, default='ward'* Which linkage criterion to use. The linkage criterion determines which distance to use between sets of observation. The algorithm will merge the pairs of cluster that minimize this criterion. • 'ward' minimizes the variance of the clusters being merged. • 'average' uses the average of the distances of each observation of the two sets. • 'complete' or 'maximum' linkage uses the maximum distances between all observations of the two sets. • 'single' uses the minimum of the distances between all observations of the two sets. For examples comparing different linkage criteria, see Comparing different hierarchical linkage methods on toy datasets.

scikit-learn Practice: *AgglomerativeClustering*

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

distance_threshold	*float, default=None* The linkage distance threshold at or above which clusters will not be merged. If not None, n_clusters must be None and compute_full_tree must be True.

Attributes	
n_clusters_	*Int* The number of clusters found by the algorithm. If distance_threshold=None, it will be equal to the given n_clusters.
labels_	*ndarray of shape (n_samples)* Cluster labels for each point.

Methods	
fit(*X*, *y*=*None*)	Fit the hierarchical clustering from features, or distance matrix.
fit_predict(*X*, *y*=*None*)	Fit and return the result of each sample's clustering assignment. In addition to fitting, this method also return the result of the clustering assignment for each sample in the training set.

*** Agglomerative Clustering has no predict method**

scikit-learn Practice: *AgglomerativeClustering*

- **Example (*blobs* dataset)**

[1]:
```
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering

X_train, _ = make_blobs(random_state=1)
print('X_train.shape:', X_train.shape)
```

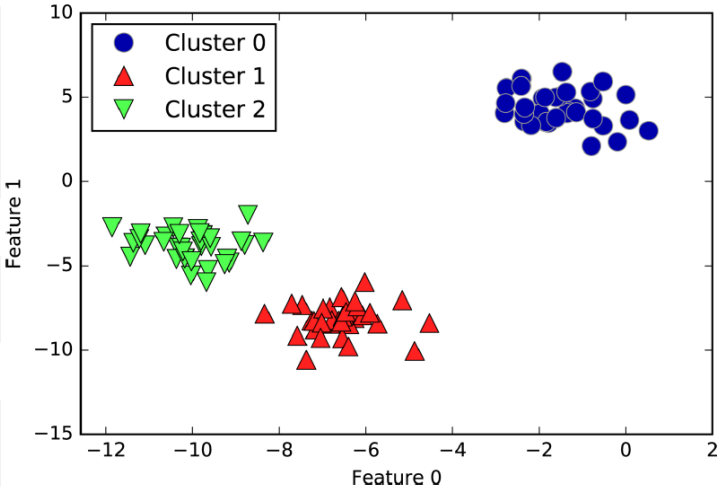
```
X_train.shape: (100, 2)
```

[2]:
```
agg = AgglomerativeClustering(n_clusters=3)
agg.fit(X_train)
```

```
AgglomerativeClustering(n_clusters=3)
```

[3]:
```
assignments_X_train = agg.labels_
print(assignments_X_train)
```

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



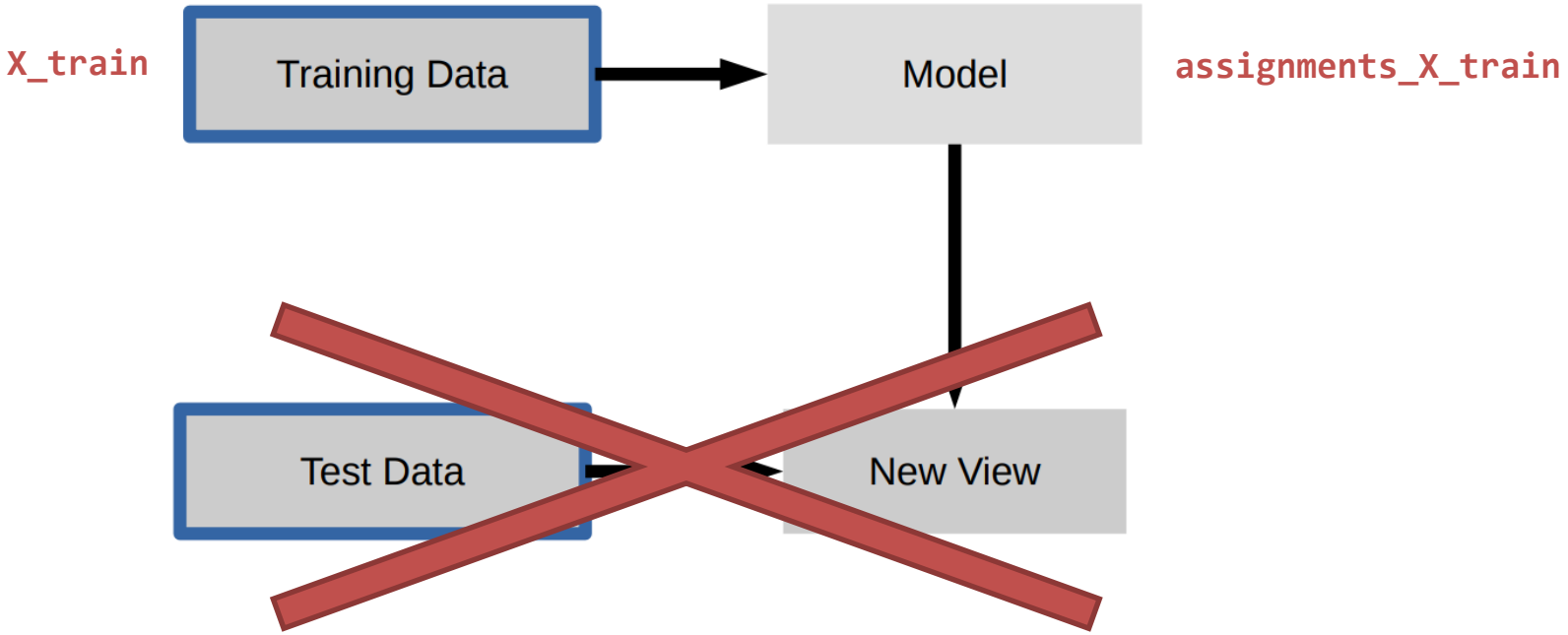
scikit-learn Practice: *AgglomerativeClustering*

- **Example (*blobs* dataset)**

```
X_train, _ = make_blobs(random_state=1)
```

```
agg = AgglomerativeClustering(n_clusters=3)
agg.fit(X_train)
```

```
assignments_X_train =agg.labels_
```



scikit-learn Practice: *AgglomerativeClustering*

- **Example (*iris* dataset) with different linkages**

[1]:
```
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import adjusted_rand_score

iris = load_iris()
X_train, y_train = iris.data, iris.target
print('X_train.shape:', X_train.shape)
```

```
X_train.shape: (150, 4)
```

[2]:
```
clustering_ari = []
linkage_settings = ['ward', 'average', 'single', 'complete']

for linkage in linkage_settings:
    # perform clustering
    agg = AgglomerativeClustering(n_clusters=3, linkage=linkage)
    agg.fit(X_train)

    # adjusted_random_index on the training set
    assignments_X_train = agg.labels_
    clustering_ari.append(adjusted_rand_score(y_train, assignments_X_train))
```

	linkage	ARI
0	ward	0.73120
1	average	0.75920
2	single	0.56375
3	complete	0.64225

scikit-learn Practice: *AgglomerativeClustering*

- **Example (*iris* dataset) with different linkages (feature scaling)**

[1]:
```
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import adjusted_rand_score

iris = load_iris()
X_train, y_train = iris.data, iris.target
print('X_train.shape:', X_train.shape)
```

```
X_train.shape: (150, 4)
```

[2]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

[3]:
```
clustering_ari = []
linkage_settings = ['ward', 'average', 'single', 'complete']

for linkage in linkage_settings:
    # perform clustering
    agg = AgglomerativeClustering(n_clusters=3, linkage=linkage)
    agg.fit(X_train_scaled)

    # adjusted_random_index on the training set
    assignments_X_train = agg.labels_
    clustering_ari.append(adjusted_rand_score(y_train, assignments_X_train))
```

	linkage	ARI
0	ward	0.61532
1	average	0.56214
2	single	0.55837
3	complete	0.57263

Discussion

- **The main hyperparameters of (agglomerative) hierarchical clustering**
  - *metric, linkage* (measuring the distances between clusters)
  - *n_clusters* or *distance_threshold* (termination condition)

  * It's important to preprocess your data (including *feature scaling* and *one-hot encoding*)
- **Strengths**
  - Hierarchical clustering gives visual representation (dendrogram) of different levels of clustering, which may correspond to meaningful taxonomies.
  - It does not have to assume any particular number of clusters.
- **Weaknesses**
  - Once a decision is made to combine two clusters, it cannot be undone.
  - It is computationally expensive.
  - The performance depends highly on scaling of features.
  - It fails at separating complex shapes.