Unsupervised Learning – Part 5

ESM3081 Programming for Data Science

Seokho Kang

성균관대학교
SUNG KYUN KWAN UNIVERSITY(SKKU)

Learning algorithms covered in this course

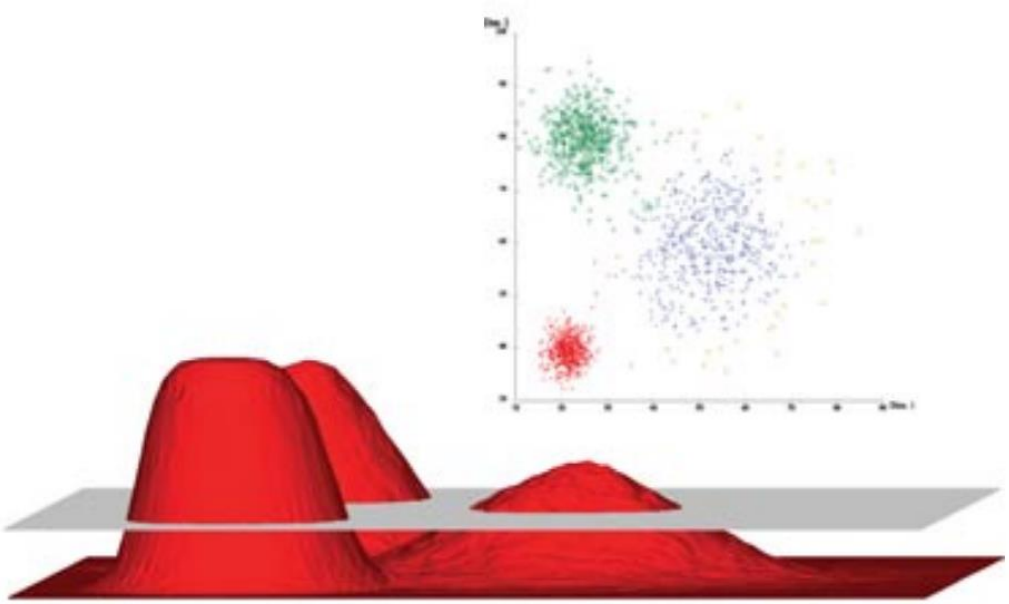
- **Data Preprocessing and Scaling**

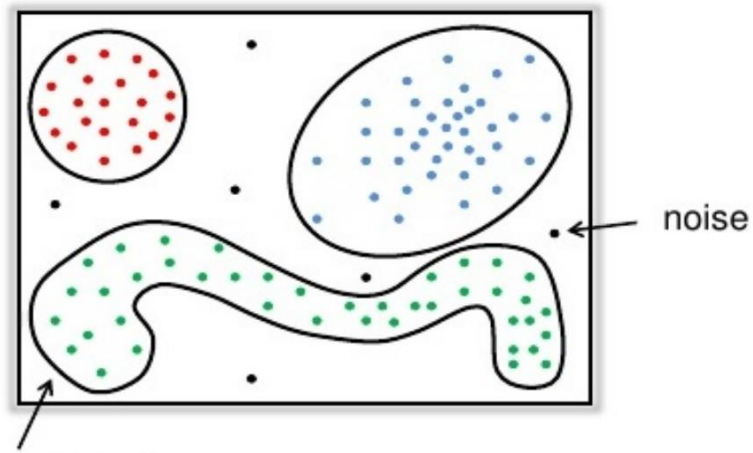
- **Unsupervised Learning**
  - **Dimensionality Reduction & Visualization**
    - (Projection) Principal Component Analysis (PCA)
    - (Manifold Learning) t-distributed Stochastic Neighbor Embedding (t-SNE)
    - …
  - **Clustering**
    - K-Means
    - Hierarchical Clustering
    - **DBSCAN**
   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

DBSCAN

Density-Based Clustering

- **A cluster is a dense region of data points, which is separated by low-density regions, from other high-density regions.**
- **Used when the clusters are irregular or intertwined, and when noise and outliers are present.**





DBSCAN

- **Density-Based Spatial Clustering of Applications with Noise (DBSCAN)**
  - DBSCAN works by identifying points that are in “crowded” regions of the feature space, where many data points are close together.
  - These regions are referred to as *dense* regions in feature space.
  - The idea behind DBSCAN is that clusters form dense regions of data, separated by regions that are relatively empty.

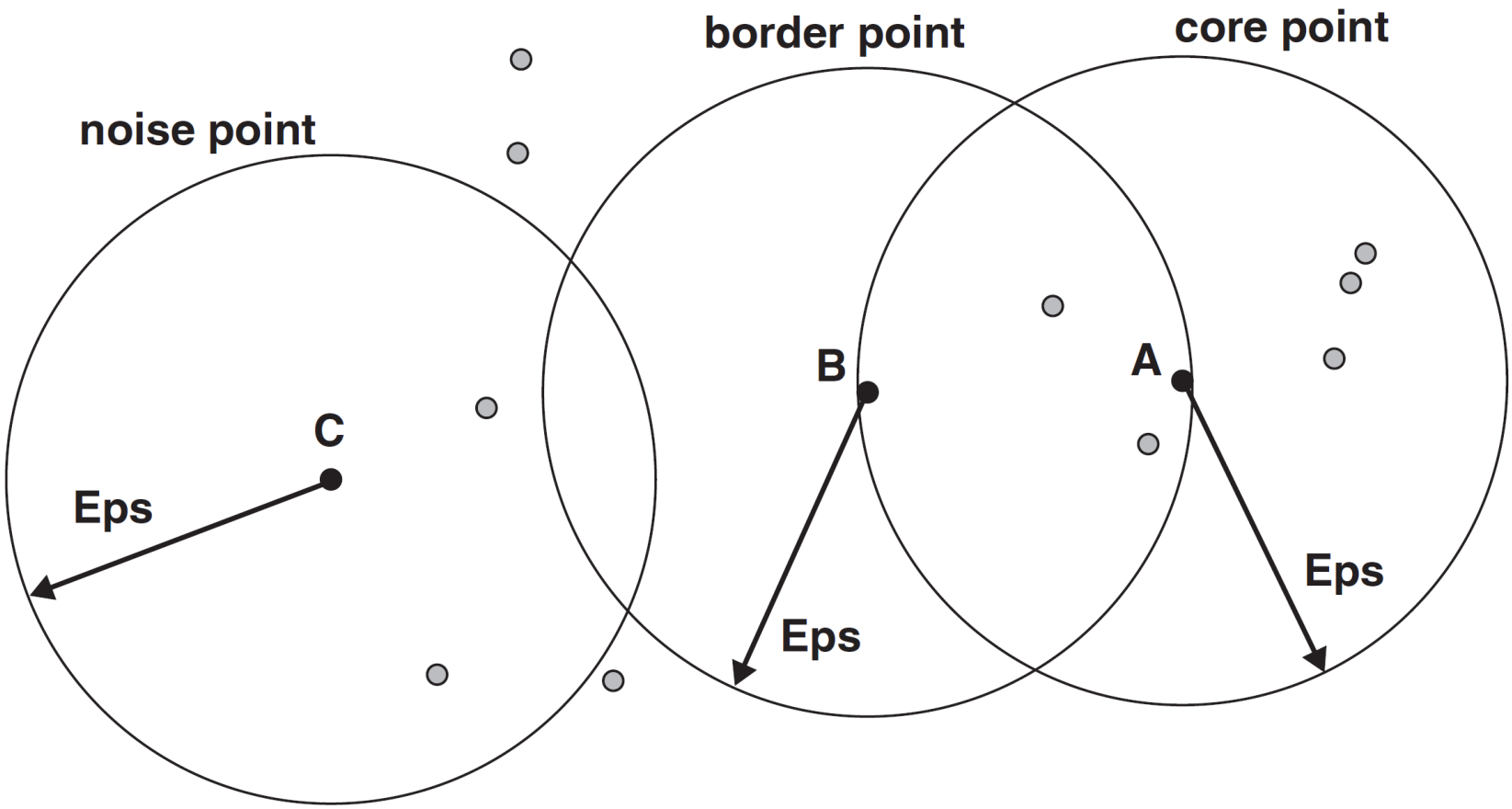
DBSCAN

- **Main hyperparameters in DBSCAN:** *eps* and *min_samples*
  - ***eps***: Maximum radius of the neighborhood
  - ***min_samples***: Minimum number of points in an ***eps***-neighborhood

- **Terminology**
  - A point is a **core point** if the number of points within the distance ***eps*** is greater than or equal to ***min_samples***
    - Core points are at the interior of a cluster.
    - Core points that are closer to each other than the distance ***eps*** are put into the same cluster.
  - A **border point** is not a core point, but is in the neighborhood of a core point
  - A **noise point** is any point that is not a core point or a border point

DBSCAN

- Examples of core, border, and noise points (min_samples=5)



DBSCAN

- **Clustering 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. Label all points as core, border, or noise points.
  2. Put an edge between all core points within a distance ***eps*** of each other.
  3. Make each group of connected core points into a separate cluster.
  4. Assign each border point to one of the clusters of its associated core points.

- **Determining the number of clusters**
  - DBSCAN doesn't require setting the number of clusters explicitly.
  - Setting *eps* implicitly controls how many clusters will be found.

  * Finding a good setting for *eps* is sometimes easier after feature scaling

scikit-learn Practice: *DBSCAN*

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

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.

New in version 0.17: metric *precomputed* to accept precomputed sparse matrix.

metric_params : *dict, default=None*
Additional keyword arguments for the metric function.

scikit-learn Practice: *DBSCAN*

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

Attributes:

core_sample_indices_ : *ndarray of shape (n_core_samples,)*
Indices of core samples.

components_ : *ndarray of shape (n_core_samples, n_features)*
Copy of each core sample found by training.

labels_ : *ndarray of shape (n_samples)*
Cluster labels for each point in the dataset given to fit(). Noisy samples are given the label -1.

- DBSCAN has no predict method

Methods

`fit`(X[, y, sample_weight])	Perform DBSCAN clustering from features, or distance matrix.
`fit_predict`(X[, y, sample_weight])	Perform DBSCAN clustering from features or distance matrix, and return cluster labels.
`get_params`([deep])	Get parameters for this estimator.
`set_params`(**params)	Set the parameters of this estimator.

scikit-learn Practice: *DBSCAN*

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

In [2]:
```
from sklearn.datasets import make_blobs
from sklearn.cluster import DBSCAN

X_train, _ = make_blobs(random_state=0, n_samples=12)
print('X_train.shape:', X_train.shape)
```

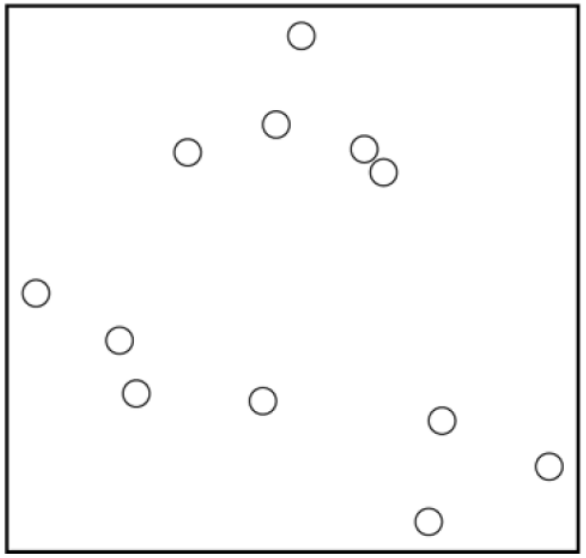
```
X_train.shape: (12, 2)
```

In [3]:
```
dbscan = DBSCAN()
dbscan.fit(X_train)
```

Out[3]:
```
▾ DBSCAN
DBSCAN()
```

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

```
[-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
```



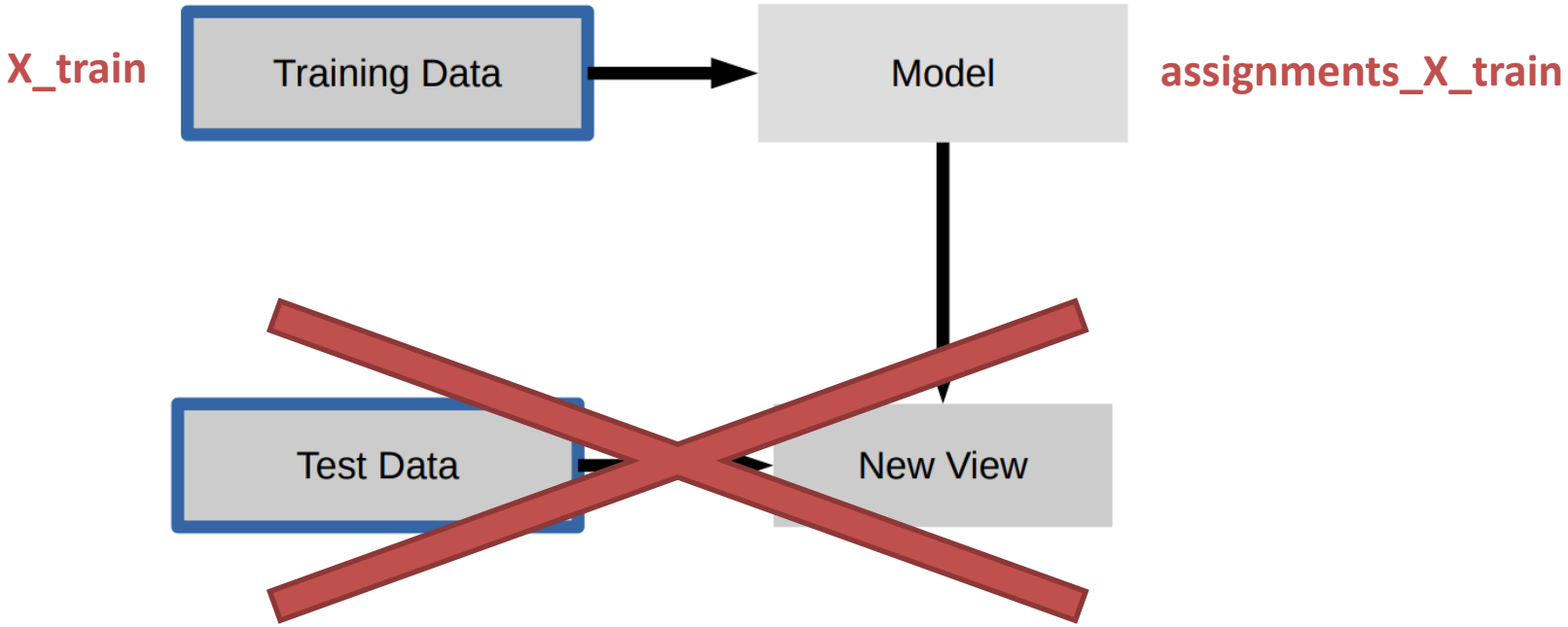
scikit-learn Practice: *DBSCAN*

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

```
X_train, _ = make_blobs(random_state=0, n_samples=12)
```

```
dbscan = DBSCAN()
dbscan.fit(X_train)
```

```
assignments_X_train = dbscan.labels_
```



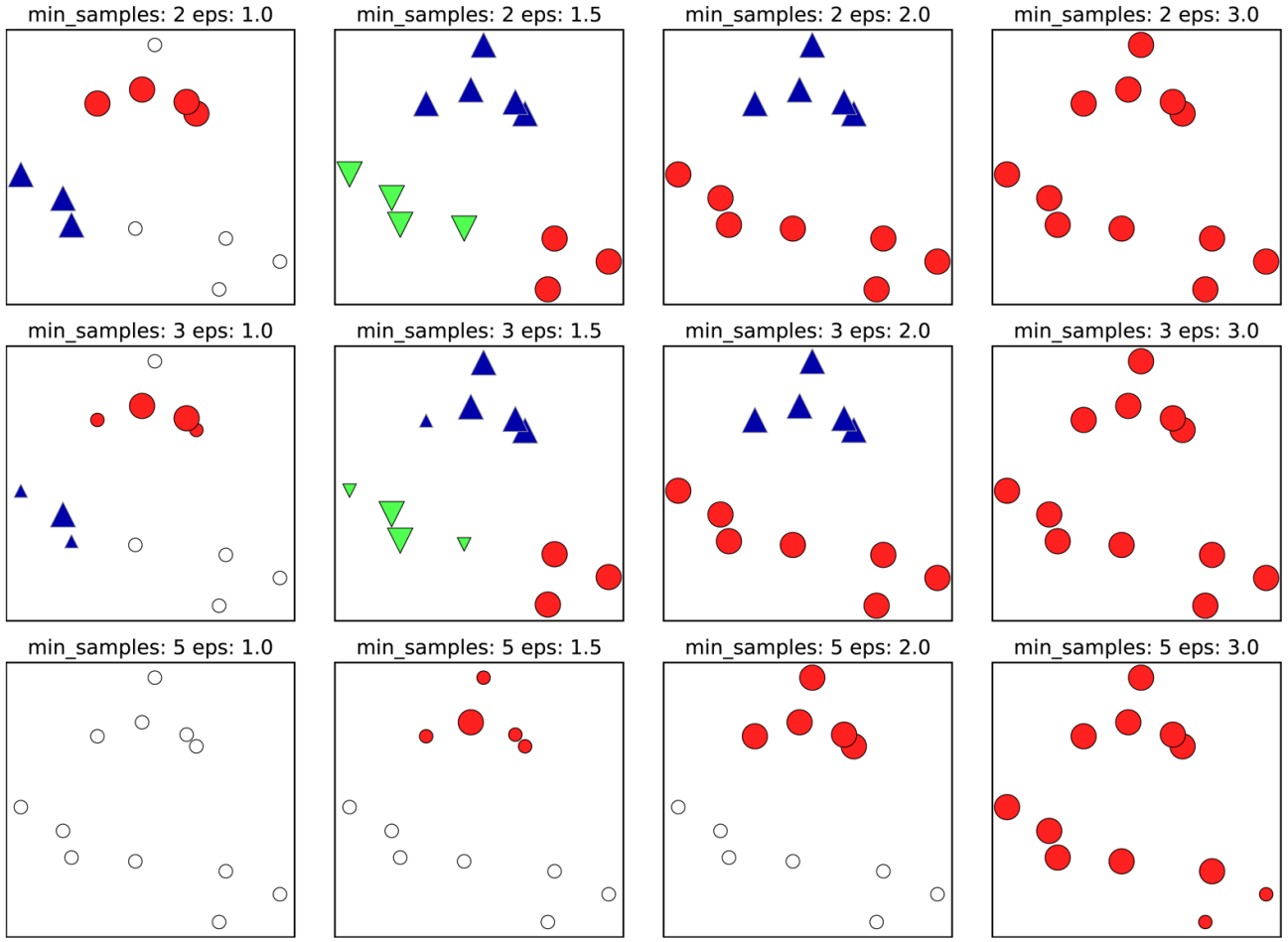
scikit-learn Practice: *DBSCAN*

- **Example (*blobs* dataset) with varying hyperparameter settings**
  - Increasing *eps* means that more points will be included in a cluster. This makes clusters grow, but might also lead to multiple clusters joining into one.
  - Increasing *min_samples* means that fewer points will be core points, and more points will be labeled as noise.

```
min_samples: 2 eps: 1.000000  cluster: [-1  0  0 -1  0 -1  1  1  0  1 -1 -1]
min_samples: 2 eps: 1.500000  cluster: [0 1 1 1 1 0 2 2 1 2 2 0]
min_samples: 2 eps: 2.000000  cluster: [0 1 1 1 1 0 0 0 1 0 0 0]
min_samples: 2 eps: 3.000000  cluster: [0 0 0 0 0 0 0 0 0 0 0 0]
min_samples: 3 eps: 1.000000  cluster: [-1  0  0 -1  0 -1  1  1  0  1 -1 -1]
min_samples: 3 eps: 1.500000  cluster: [0 1 1 1 1 0 2 2 1 2 2 0]
min_samples: 3 eps: 2.000000  cluster: [0 1 1 1 1 0 0 0 1 0 0 0]
min_samples: 3 eps: 3.000000  cluster: [0 0 0 0 0 0 0 0 0 0 0 0]
min_samples: 5 eps: 1.000000  cluster: [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
min_samples: 5 eps: 1.500000  cluster: [-1  0  0  0  0 -1 -1 -1  0 -1 -1 -1]
min_samples: 5 eps: 2.000000  cluster: [-1  0  0  0  0 -1 -1 -1  0 -1 -1 -1]
min_samples: 5 eps: 3.000000  cluster: [0 0 0 0 0 0 0 0 0 0 0 0]
```

scikit-learn Practice: *DBSCAN*

- **Example (*blobs* dataset) with varying hyperparameter settings**
  - Points that belong to clusters are solid; noise points are shown in white; core points are shown as large markers; boundary points are displayed as smaller markers.



scikit-learn Practice: *DBSCAN*

- **Example (*blobs* dataset) with feature scaling**

In [5]:
```
from sklearn.datasets import make_blobs
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import DBSCAN

X_train, _ = make_blobs(random_state=0, n_samples=12)
print('X_train.shape:', X_train.shape)
```

```
X_train.shape: (12, 2)
```

In [6]:
```
scaler = MinMaxScaler((-1, 1))
X_train_scaled = scaler.fit_transform(X_train)
```

In [7]:
```
dbscan = DBSCAN()
dbscan.fit(X_train_scaled)
```

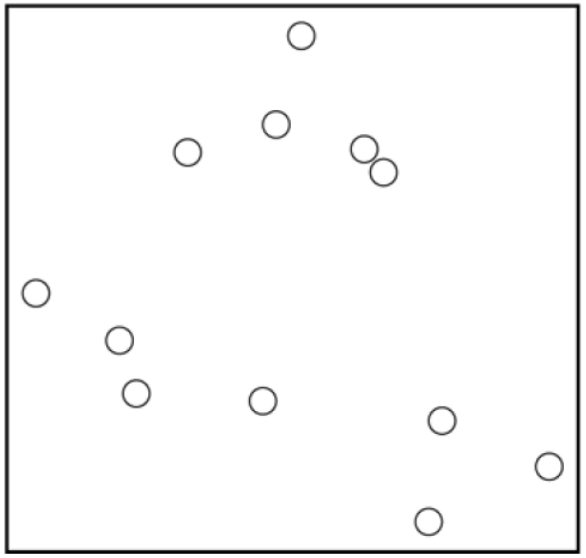
Out[7]:

▾ DBSCAN

DBSCAN()

In [8]:
```
assignments_X_train_scaled = dbscan.labels_
print(assignments_X_train_scaled)
```

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



Discussion

- **The main hyperparameters of DBSCAN**
  - *eps, min_samples*
  - *metric (distance metric)*

  * It's important to preprocess your data (including *feature scaling* and *one-hot encoding*)
- **Strengths**
  - It does not require the user to set the number of clusters a priori.
  - It can capture clusters of complex shapes.
  - It can identify points that are not part of any cluster, and is robust to outliers.
- **Weaknesses**
  - It cannot cluster data well with large differences in densities.
  - It does not work well with high-dimensional data.
  - The performance depends highly on scaling of features.

Comparing and Evaluating Clustering Algorithms

Summary of Clustering Algorithms

- Applying and evaluating clustering is a highly qualitative procedure, and often most helpful in the exploratory phase of data analysis.
- We looked at three clustering algorithms: *k*-means, hierarchical clustering, and DBSCAN.
  - All three have a way of controlling the granularity of clustering.
  - All three algorithms can be used on large, real-world datasets, are relatively easy to understand, and allow for clustering into many clusters.
  - Each of the algorithms has somewhat different strengths.

Summary of Clustering Algorithms

- ***k*-means** allows for a characterization of the clusters using the cluster means. It can also be viewed as a decomposition method, where each data point is represented by its cluster center.
- **Hierarchical clustering** can provide a whole hierarchy of possible partitions of the data, which can be easily inspected via dendrograms.
- **DBSCAN** allows for the detection of “noise points” that are not assigned any cluster, and it can help automatically determine the number of clusters. In contrast to the other two algorithms, it allow for complex cluster shapes. It sometimes produces clusters of very differing size, which can be a strength or a weakness.

Validating Clusters

- **When cluster validation is needed?**
  - Determining the clustering tendency of a set of data, *i.e.*, distinguishing whether non-random structure actually exists in the data.
  - Determining the ‘correct’ number of clusters.
  - Comparing the results of two different sets of clustering results to determine which is better.

Validating Clusters

- **For (supervised) classification we have a variety of measures to evaluate how good our model is.**
  - *e.g.*, accuracy, precision, recall, AUROC, …

- **For (unsupervised) clustering, the analogous question is how to evaluate the "goodness" of the resulting clusters.**
  - The validation of the clusters is the most difficult and frustrating part of clustering.
  - But, without a strong effort in this direction, clustering results will have no meaning.

Quantitative Evaluation With Ground Truth

- Given the knowledge of the ground truth class assignments, there are measures that can be used to assess the outcome of a clustering algorithm.

- The most common measure is the *adjusted rand index* (ARI), which provides a quantitative measure with an optimum of 1 and a value of 0 for unrelated clusterings (though the ARI can become negative).

- A common mistake when evaluating clustering in this way is to use *accuracy_score*.
  - It requires the assigned cluster labels to exactly match the ground truth.
  - However, the cluster labels themselves are meaningless—the only thing that matters is which points are in the same cluster.

scikit-learn Practice: *adjusted_rand_score*

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html

sklearn.metrics.adjusted_rand_score

sklearn.metrics.**adjusted_rand_score**(*labels_true*, *labels_pred*) [source]

Rand index adjusted for chance.

The Rand Index computes a similarity measure between two clusterings by considering all pairs of samples and counting pairs that are assigned in the same or different clusters in the predicted and true clusterings.

The raw RI score is then "adjusted for chance" into the ARI score using the following scheme:

```
ARI = (RI - Expected_RI) / (max(RI) - Expected_RI)
```

The adjusted Rand index is thus ensured to have a value close to 0.0 for random labeling independently of the number of clusters and samples and exactly 1.0 when the clusterings are identical (up to a permutation).

ARI is a symmetric measure:

```
adjusted_rand_score(a, b) == adjusted_rand_score(b, a)
```

Read more in the User Guide.

Parameters:

labels_true : *int array, shape = [n_samples]*
Ground truth class labels to be used as a reference

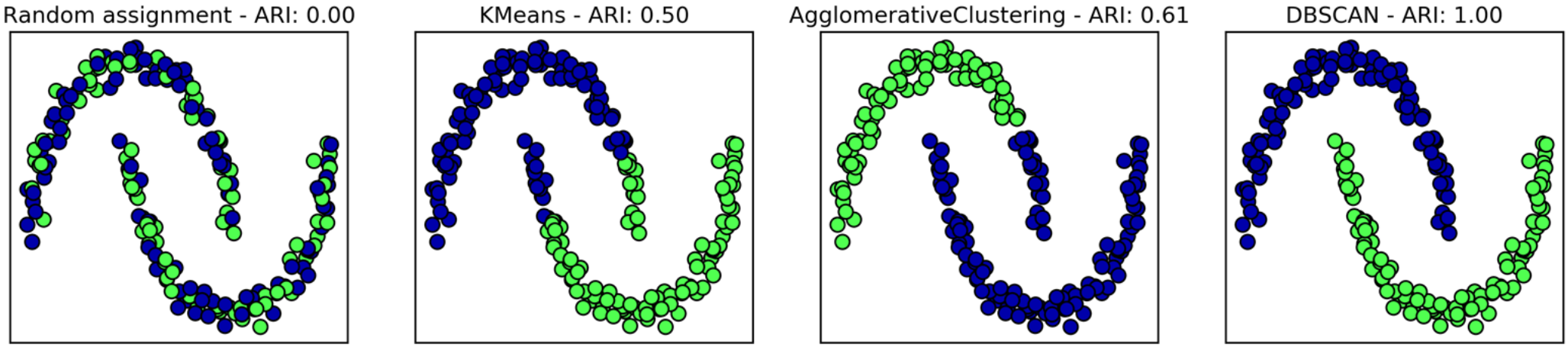
labels_pred : *array-like of shape (n_samples,)*
Cluster labels to evaluate

Returns:

ARI : *float*
Similarity score between -1.0 and 1.0. Random labelings have an ARI close to 0.0. 1.0 stands for perfect match.

scikit-learn Practice: *adjusted_rand_score*

- ***Example: Comparing random assignment, k-means, agglomerative clustering, and DBSCAN on the two_moons dataset using the supervised ARI score***
  - The adjusted rand index provides intuitive results:
    Random assignment having a score of 0;
    DBSCAN having a score of 1.



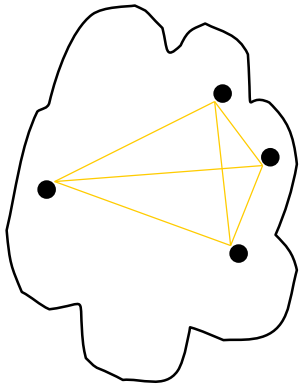
Quantitative Evaluation Without Ground Truth

- When applying clustering algorithms, there is usually **no ground truth** to which to compare the results.
  - If we knew the right clustering of the data, we could use this information to build a supervised model like a classifier.
  - Using measures like ARI usually only helps in developing algorithms, not in assessing success in an application.

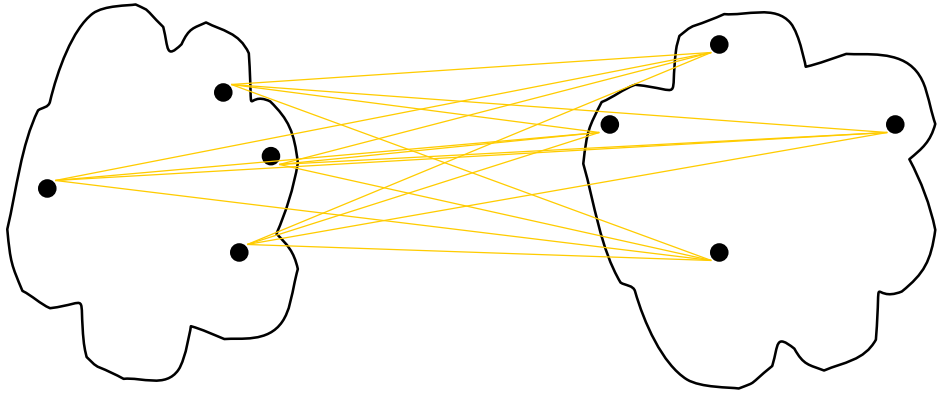
- There are scoring measures for clustering that don't require ground truth, like the *silhouette coefficient*.
  - The silhouette score computes the compactness of a cluster, where higher is better, with a perfect score of 1. While compact clusters are good, compactness doesn't allow for complex shapes.
  - It often doesn't work well in practice.

Quantitative Evaluation Without Ground Truth

- **Cluster Cohesion:** How closely related are data points in a cluster
- **Cluster Separation:** How distinct or well-separated a cluster is from other clusters



cohesion



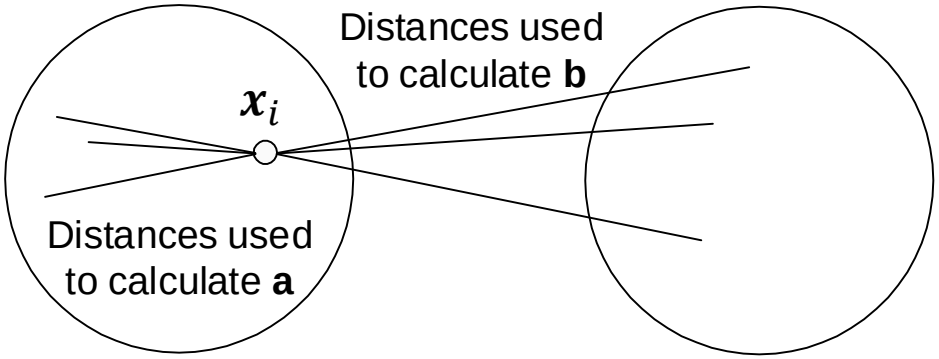
separation

Quantitative Evaluation Without Ground Truth

- **Example: Silhouette Coefficient**
  - Silhouette coefficient combines ideas of both cohesion and separation
  - For each individual data point $\boldsymbol{x}_i$,
    - Calculate a($\boldsymbol{x}_i$) = average distance of $\boldsymbol{x}_i$ to points in its cluster ← cohesion
    - Calculate b($\boldsymbol{x}_i$) = min (average distance of $\boldsymbol{x}_i$ to points in another cluster) ← separation
    - The silhouette coefficient for $\boldsymbol{x}_i$ is then given by

$$s(\boldsymbol{x}_i) = (b(\boldsymbol{x}_i) - a(\boldsymbol{x}_i)) / \max(a(\boldsymbol{x}_i), b(\boldsymbol{x}_i))$$

  - Calculate the average over the entire dataset
  - Value can vary between -1 and 1, typically between 0 and 1, and the closer to 1 the better



scikit-learn Practice: *silhouette_score*

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html

sklearn.metrics.silhouette_score

`sklearn.metrics.` **`silhouette_score`**(*X, labels, *, metric='euclidean', sample_size=None, random_state=None, **kwds*) [source]

Compute the mean Silhouette Coefficient of all samples.

The Silhouette Coefficient is calculated using the mean intra-cluster distance (`a`) and the mean nearest-cluster distance (`b`) for each sample. The Silhouette Coefficient for a sample is `(b - a) / max(a, b)`. To clarify, `b` is the distance between a sample and the nearest cluster that the sample is not a part of. Note that Silhouette Coefficient is only defined if number of labels is `2 <= n_labels <= n_samples - 1`.

This function returns the mean Silhouette Coefficient over all samples. To obtain the values for each sample, use `silhouette_samples`.

The best value is 1 and the worst value is -1. Values near 0 indicate overlapping clusters. Negative values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar.

Read more in the User Guide.

Parameters:

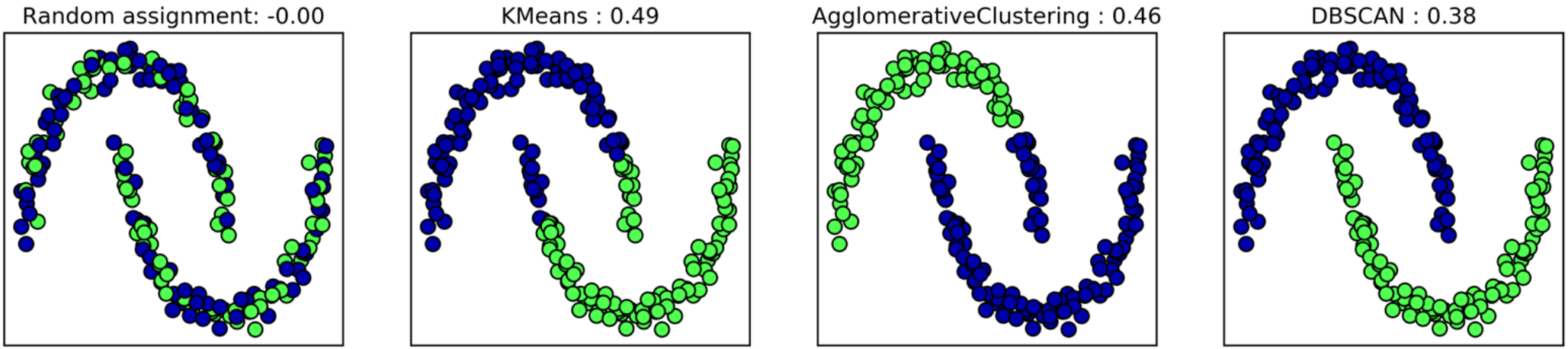
X : *array-like of shape (n_samples_a, n_samples_a) if metric == "precomputed" or (n_samples_a, n_features) otherwise*
An array of pairwise distances between samples, or a feature array.

labels : *array-like of shape (n_samples,)*
Predicted labels for each sample.

metric : *str or callable, default='euclidean'*
The metric to use when calculating distance between instances in a feature array. If metric is a string, it must be one of the options allowed by `metrics.pairwise.pairwise_distances`. If `X` is the distance array itself, use `metric="precomputed"`.

scikit-learn Practice: *silhouette_score*

- ***Example: Comparing random assignment, k-means, agglomerative clustering, and DBSCAN on the two_moons dataset using the unsupervised silhouette score***
  - *k*-means gets the highest silhouette score, even though we might prefer the result produced by DBSCAN.



Qualitative Evaluation

- **Interpretation (most important)**
  - **Goal:** obtain meaningful and useful clusters
  - **Caveats:**
    - (1) Random chance can often produce apparent clusters
    - (2) Different clustering algorithms produce different results
  - **Solutions:**
    - Obtain summary statistics
    - Review clusters in terms of features
    - Label the clusters **(based on the knowledge of human experts)**

Clustering Checklist

(a) What is a cluster?
(b) What features should be used?
(c) Should the data be normalized?
(d) Does the data contain any outliers?
(e) How do we define the pair-wise similarity?
(f) How many clusters are present in the data?
(g) Which clustering method should be used?
(h) Does the data have any clustering tendency?
(i) Are the discovered clusters and partition valid?

Summary and Outlook

Takeaway

- **Things to keep in mind**
  - **Setting the right hyperparameters** is important for good performance.
  - Some of the algorithms are also sensitive to how we represent the input data.
  - Blindly applying an algorithm to a dataset without understanding the assumptions the model makes and the meanings of the hyperparameter settings will rarely lead to an accurate model.
  - One can usually do much better with a correct application of a commonplace algorithm than by sloppily applying an obscure algorithm.

Quick Summary

- This chapter introduced a range of **unsupervised learning algorithms** (data preprocessing and scaling, dimensionality reduction & visualization, clustering) that can be applied for **exploratory data analysis** and **preprocessing**.
  - They are essential tools to further your understanding of your data, and can be the only ways to make sense of your data in the absence of supervision information.
  - It is hard to quantify the usefulness of an unsupervised algorithm, though this shouldn't deter you from using them to gather insights from your data.
  - Even in a supervised setting, exploratory tools are important for a better understanding of the properties of the data.