Homework #2

Implementing a simple clustering program

To Do (1/4)

- Implement a simple ***clustering*** program that works as follows:

① Print the menu

```
[ Student ID: your student ID ]
[ Name: your name ]

1. K-means
2. Hierarchical clustering
3. DBSCAN
4. Quit
>_
```

② Generate a ***synthetic*** dataset for clustering

– As described in page 6

To Do (2/4)

③ If the user enters 1, perform ***k-means*** as follows:

```
1. K-means
2. Hierarchical clustering
3. DBSCAN
4. Quit
>1
n_clusters: 4
random_state: 0
<Clustering result>
[1 0 3 ...] // cluster labels of each point (print(labels))

[ Student ID: your student ID ]
[ Name: your name ]

1. K-means
2. Hierarchical clustering
3. DBSCAN
4. Quit

>_
```

To Do (3/4)

④ If the user enters 2, perform ***hierarchical clustering*** as follows:

```
1. K-means
2. Hierarchical clustering
3. DBSCAN
4. Quit
>2
n_clusters: 4
linkage: single // ward, complete, average, or single
<Clustering result>
[3 2 0 ...] // cluster labels of each point (print(labels))

[ Student ID: your student ID ]
[ Name: your name ]

1. K-means
2. Hierarchical clustering
3. DBSCAN
4. Quit

>_
```

To Do (4/4)

⑤ If the user enters 3, perform ***DBSCAN*** as follows:

```
1. K-means
2. Hierarchical clustering
3. DBSCAN
4. Quit
>3
eps: 0.5
min_samples: 7
<Clustering result>
[1 4 3 ...] // cluster labels of each point (print(labels))

[ Student ID: your student ID ]
[ Name: your name ]

1. K-means
2. Hierarchical clustering
3. DBSCAN
4. Quit

>_
```

Requirements (*Important!*)

- As data for clustering, you ***must*** generate a synthetic dataset in your Python code as follows:

```
from sklearn.datasets import make_blobs
X, _ = make_blobs(n_samples=100, centers=5,
                  n_features=2, random_state=0)
```

- You ***must*** use the ***default*** values for all parameters for clustering except for the input parameters
  - The input parameters for *k*-means: `n_clusters`, `random_state`
  - The input parameters for hierarchical: `n_clusters`, `linkage`
  - The input parameters for DBSCAN: `eps`, `min_samples`

Evaluation Criteria (15 pts)

- Correctness of your program (12 pts)
  - K-means: 2 test cases $\times$ 2 pts = 4 pts
  - Hierarchical clustering: 2 test cases $\times$ 2 pts = 4 pts
  - DBSCAN: 2 test cases $\times$ 2 pts = 4 pts

- Accordance with the requirements (3 pts)
  - Menu
  - Input
  - Output
  - Other program requirements

Submission

- Submit your Python code (`.py`) to Snowboard
  - SnowBoard → 데이터마이닝및분석 → 14주차 → Homework #2

- Due: **2023.6.23 (Fri) 23:59**
  - 1-day delay: 80% credit
  - 2-day delay or more: 0% credit

Homework Support

- **[Off-line]** If you need help from me, I recommend you to request an off-line meeting with me
  - The number of students ≥ 1
  - Schedule a meeting time with me

- **[On-line]** Slack or emails
  - Of course, you can use the Slack for Q&A, discussion, etc.

- **[Teaching Assistant]** You can get help from the T.A.
  - Eunjo Jang (wkddmswh99@sookmyung.ac.kr)
  - Seongyeon Yang (yeon55@sookmyung.ac.kr)

Scikit-Learn

Clustering

Clustering with Scikit-Learn

- You can use the ***sklearn.cluster*** module
  - Gathers popular unsupervised ***clustering*** algorithms

Classes	
`cluster.AffinityPropagation`(*[, damping, ...])	Perform Affinity Propagation Clustering of data.
`cluster.AgglomerativeClustering`([...])	Agglomerative Clustering.
`cluster.Birch`(*[, threshold, ...])	Implements the BIRCH clustering algorithm.
`cluster.DBSCAN`([eps, min_samples, metric, ...])	Perform DBSCAN clustering from vector array or distance matrix.
`cluster.FeatureAgglomeration`([n_clusters, ...])	Agglomerate features.
`cluster.KMeans`([n_clusters, init, n_init, ...])	K-Means clustering.
`cluster.BisectingKMeans`([n_clusters, init, ...])	Bisecting K-Means clustering.
`cluster.MiniBatchKMeans`([n_clusters, init, ...])	Mini-Batch K-Means clustering.
`cluster.MeanShift`(*[, bandwidth, seeds, ...])	Mean shift clustering using a flat kernel.
`cluster.OPTICS`(*[, min_samples, max_eps, ...])	Estimate clustering structure from vector array.
`cluster.SpectralClustering`([n_clusters, ...])	Apply clustering to a projection of the normalized Laplacian.
`cluster.SpectralBiclustering`([n_clusters, ...])	Spectral biclustering (Kluger, 2003).
`cluster.SpectralCoclustering`([n_clusters, ...])	Spectral Co-Clustering algorithm (Dhillon, 2001).

sklearn.cluster.KMeans

- Performs *k*-means clustering

```
class sklearn.cluster.KMeans(...)
```

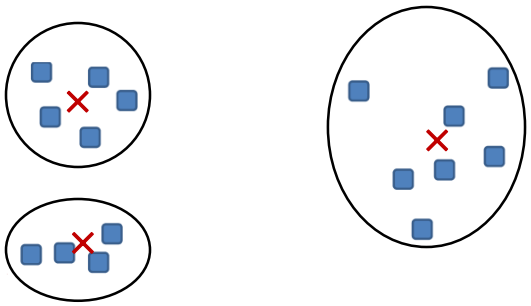
  - **Parameters**
    - `n_clusters`: the number of clusters to form (default = `8`)
    - `init`: method for initialization (default = `'k-means++'`)
      - `'k-means++'`: select initial cluster centroids using *k*-means++
      - `'random'`: selects *k* points at random from data for the initial centroids
    - `n_init`: the number of time the *k*-means algorithm will be run with different centroid seeds (default = `10`)
    - `max_iter`: the maximum number of iterations (default = `300`)
    - `tol`: relative tolerance to declare convergence (default = `1e-4`)
    - `random_state`: the seed used by the random number generator

sklearn.cluster.KMeans

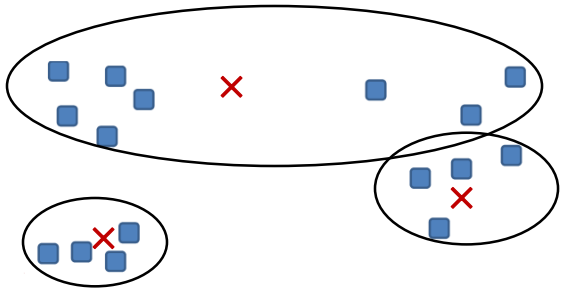
- Performs *k*-means clustering

```
class sklearn.cluster.KMeans(...)
```

  - **Attributes**
    - `cluster_centers_`: coordinates of cluster centers (i.e., centroids)
    - `labels_`: cluster labels for each point
    - `inertia_`: sum of squared distances of samples to their closest centroid



Good clustering



Bad clustering

sklearn.cluster.KMeans

- Performs *k*-means clustering

```
class sklearn.cluster.KMeans(...)
```

  - **Methods**
    - `fit(X)`: compute *k*-means clustering
      - `X`: instances to cluster (array-like of shape `(n_samples, n_features))`
    - `fit_predict(X)`: compute and predict cluster index for each sample
      - `X`: instances to cluster (array-like of shape `(n_samples, n_features))`
    - `predict(X)`: predict the closest cluster each sample in `X` belongs to
      - `X`: new data to predict (array-like of shape `(n_samples, n_features)`)

sklearn.cluster.KMeans

- Performs *k*-means clustering

```
class sklearn.cluster.KMeans(...)
```

  - **Example**

```
>>> from sklearn.cluster import KMeans
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [10, 2], [10, 4], [10, 0]])
>>> kmeans = KMeans(n_clusters=2, random_state=0, n_init="auto").fit(X)
>>> kmeans.labels_
array([1, 1, 1, 0, 0, 0], dtype=int32)
>>> kmeans.predict([[0, 0], [12, 3]])
array([1, 0], dtype=int32)
>>> kmeans.cluster_centers_
array([[10.,  2.],
       [ 1.,  2.]])
```

sklearn.cluster.AgglomerativeClustering

- Recursively merges pair of clusters of sample data using linkage distance

```
class sklearn.cluster.AgglomerativeClustering(...)
```

  - **Parameters**
    - `n_clusters`: the number of clusters to find (default = `2`)
    - `metric`: metric used to compute the linkage (default = None)
      - "`euclidean`", "`l1`", "`l2`", "`manhattan`", "`cosine`", or "`precomputed`"
    - `linkage`: which linkage criterion to use (default = '`ward`')
      - '`ward`': the variance of the clusters being merged
      - '`average`': the average of the distances of each point of the two clusters
      - '`complete`' or '`maximum`': the maximum distances between all points of the two clusters
      - '`single`': the minimum of the distances between all points of the two clusters

sklearn.cluster.AgglomerativeClustering

- Recursively merges pair of clusters of sample data using linkage distance

```
class sklearn.cluster.AgglomerativeClustering(...)
```

  - **Attributes**
    - `n_clusters_`: the number of clusters found by the algorithm
    - `labels_`: cluster labels for each point
    - `n_leaves_`: the number of leaves in the hierarchical tree
    - `children_`: the children of each non-leaf node

sklearn.cluster.AgglomerativeClustering

- Recursively merges pair of clusters of sample data using linkage distance

```
class sklearn.cluster.AgglomerativeClustering(...)
```

  - **Methods**
    - `fit(X)`: fit the hierarchical clustering from features
      - `X`: instances to cluster (array-like of shape `(n_samples, n_features)`)
    - `fit_predict(X)`: fit and return each sample's clustering assignment
      - `X`: instances to cluster (array-like of shape `(n_samples, n_features)`)

sklearn.cluster.AgglomerativeClustering

- Recursively merges pair of clusters of sample data using linkage distance

```
class sklearn.cluster.AgglomerativeClustering(...)
```

  - **Example**

```
>>> from sklearn.cluster import AgglomerativeClustering
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [4, 2], [4, 4], [4, 0]])
>>> clustering = AgglomerativeClustering().fit(X)
>>> clustering
AgglomerativeClustering()
>>> clustering.labels_
array([1, 1, 1, 0, 0, 0])
```

sklearn.cluster.DBSCAN

- Performs DBSCAN clustering

```
class sklearn.cluster.DBSCAN(...)
```

  - **Parameters**
    - `eps`: the maximum distance between two samples for one to be considered as in the neighborhood of the other (default = `0.5`)
    - `min_samples`: the number of samples in a neighborhood for a point to be considered as a core point (default = `5`)
    - `metric`: the metric to use when calculating distance between instances (default = `'euclidean'`)
      - `"euclidean"`, `"l1"`, `"l2"`, `"manhattan"`, `"cosine"`, or `"precomputed"`

`sklearn.cluster.DBSCAN`

- Performs DBSCAN clustering

```
class sklearn.cluster.DBSCAN(...)
```

  - **Attributes**
    - `core_sample_indices_`: indices of core samples
    - `labels_`: cluster labels for each point in the dataset given to `fit()`. Noisy samples are given the label `-1`

sklearn.cluster.DBSCAN

- Performs DBSCAN clustering

```
class sklearn.cluster.DBSCAN(...)
```

  - **Methods**
    - `fit(X)` : compute DBSCAN clustering
      - `X`: instances to cluster (array-like of shape `(n_samples, n_features))`
    - `fit_predict(X)` : compute clusters from a data and predict labels
      - `X`: instances to cluster (array-like of shape `(n_samples, n_features))`

sklearn.cluster.DBSCAN

- Performs DBSCAN clustering

```
class sklearn.cluster.DBSCAN(...)
```

– **Example**

```
>>> from sklearn.cluster import DBSCAN
>>> import numpy as np
>>> X = np.array([[1, 2], [2, 2], [2, 3],
...               [8, 7], [8, 8], [25, 80]])
>>> clustering = DBSCAN(eps=3, min_samples=2).fit(X)
>>> clustering.labels_
array([ 0,  0,  0,  1,  1, -1])
>>> clustering
DBSCAN(eps=3, min_samples=2)
```