Unsupervised Learning

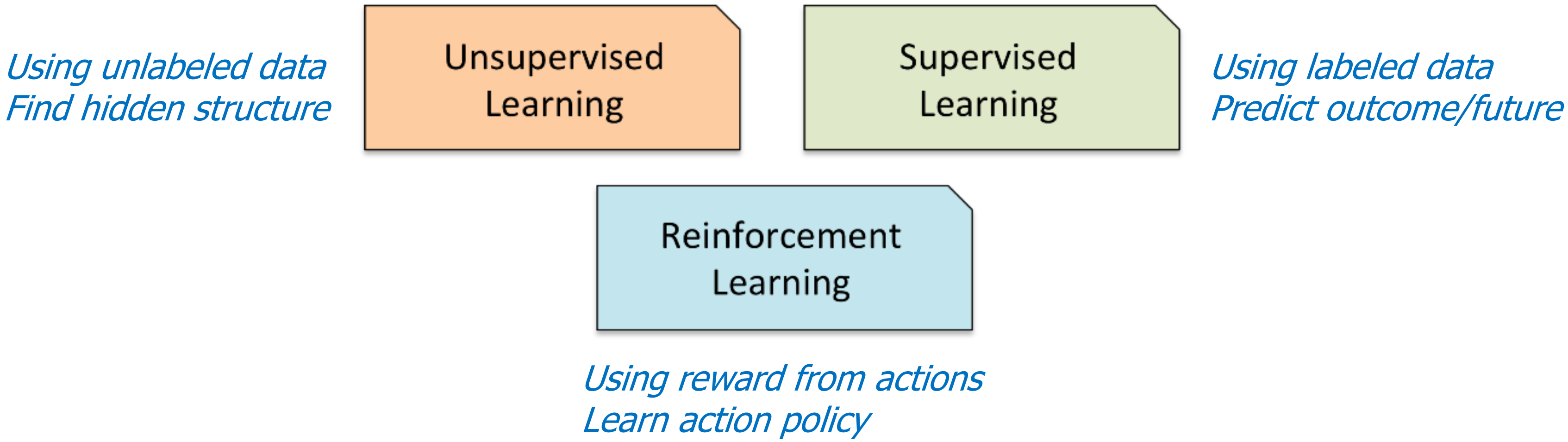
Machine Learning

Contents

- K-means clustering
  - K-means
  - K-means++
  - Distortion, Silhouette : Quantifying the quality of cluster
- DBSCAN(Density-Based Spatial Clustering of Application with Noise)
  - DBSCAN
  - K-means vs DBSCAN
  - Outlier Detection

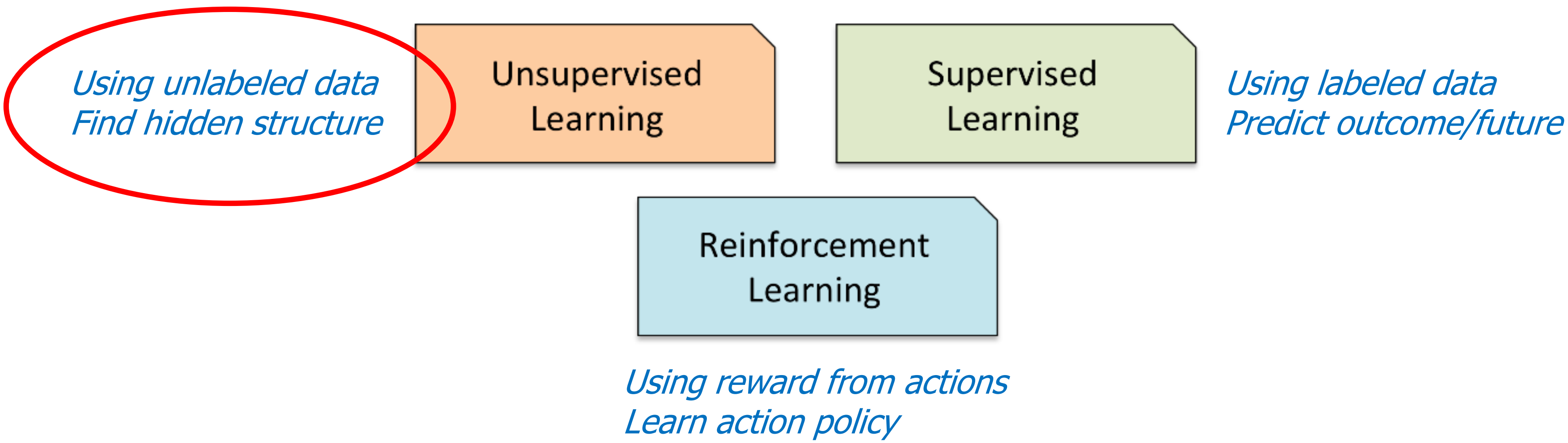
Machine Learning

- Types of learning
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning



Machine Learning

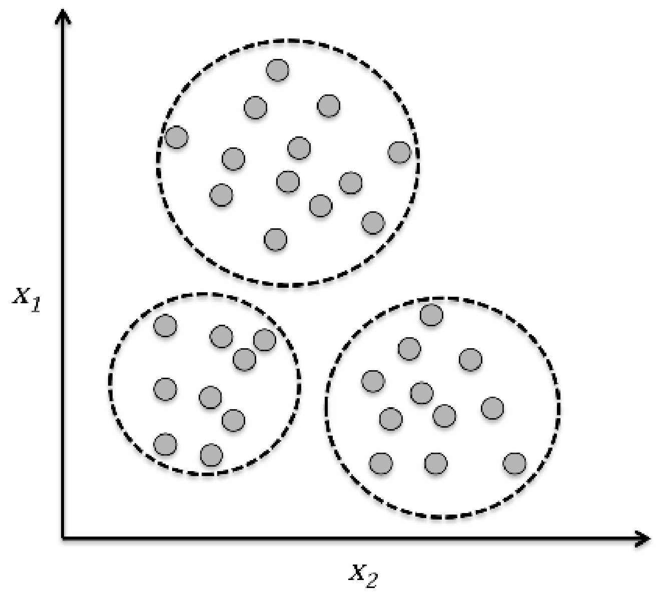
- Types of learning
  - Supervised learning
  - Unsupervised learning
  - Reinforcement learning



Unsupervised Learning

- Finding subgroups with clustering
  - Given: **<data>** examples
  - Learning: **<groups>** of data

H	W
185	65
160	90
180	70
165	95
…	…



K-means clustering

- K-means clustering
  - Our goal is to group the samples based on their feature similarities
  - Algorithm
    1. Randomly pick k centroids from the sample points as initial clusters
    2. Assign each sample to the nearest centroid $\mu^{(j)}, j \in \{1, \ldots, k\}$
    3. Move the centroids to the center of the samples that were assigned to it
    4. Repeat steps 2 and 3 until the cluster assignments do not change or a user-defined tolerance or a maximum number of iterations is reached

K-means clustering

- K-means clustering
  - We can define similarity as the opposite of distance, "square Euclidean distance" between 2 points x and y in m-dimensional space

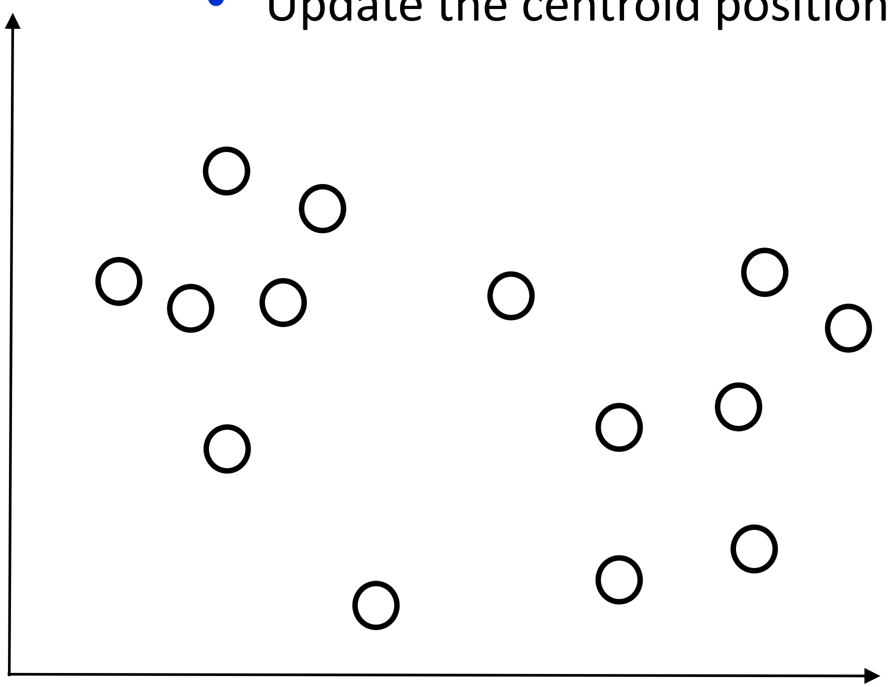
$$d(x,y) = \sum_{j=1}^{m} (x_j - y_j)^2 = \|x - y\|_2^2$$

  - Based on this Euclidean distance metric, we can describe the k-means algorithm as a simple optimization problem
  - An iterative approach for minimizing the within-cluster sum of squared errors(SSE)

$$SSE = \sum_{i=1}^{n} \sum_{j=1}^{k} w^{(i,j)} \left\| x^{(i)} - \mu^{(j)} \right\|_2^2$$

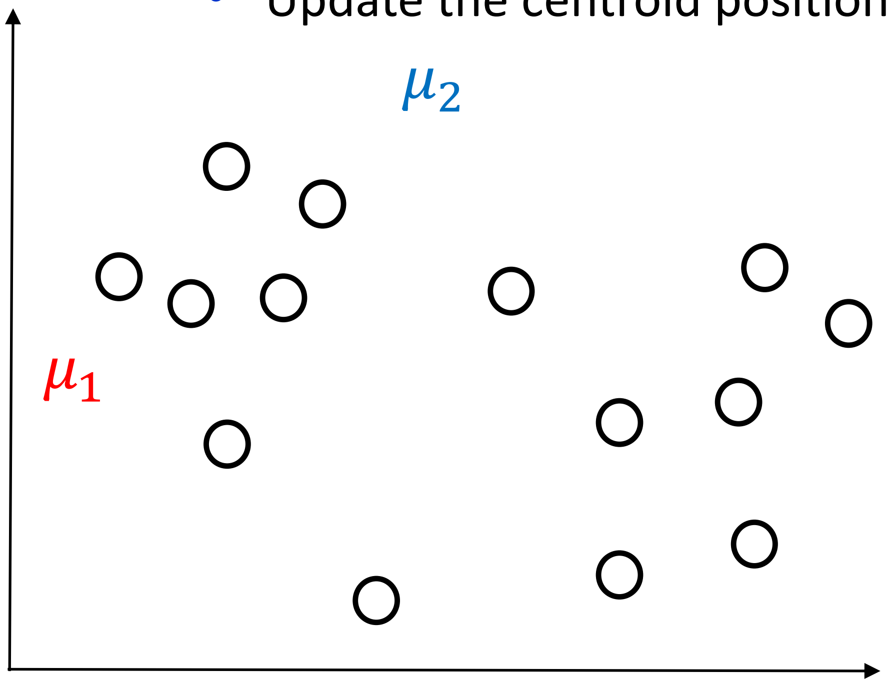
K-means clustering

- EM(Expectation and Maximization) algorithm for K-means
  - Expectation
    - Assign the data points to the nearest centroid
  - Maximization
    - Update the centroid positions given the assignments



K-means clustering

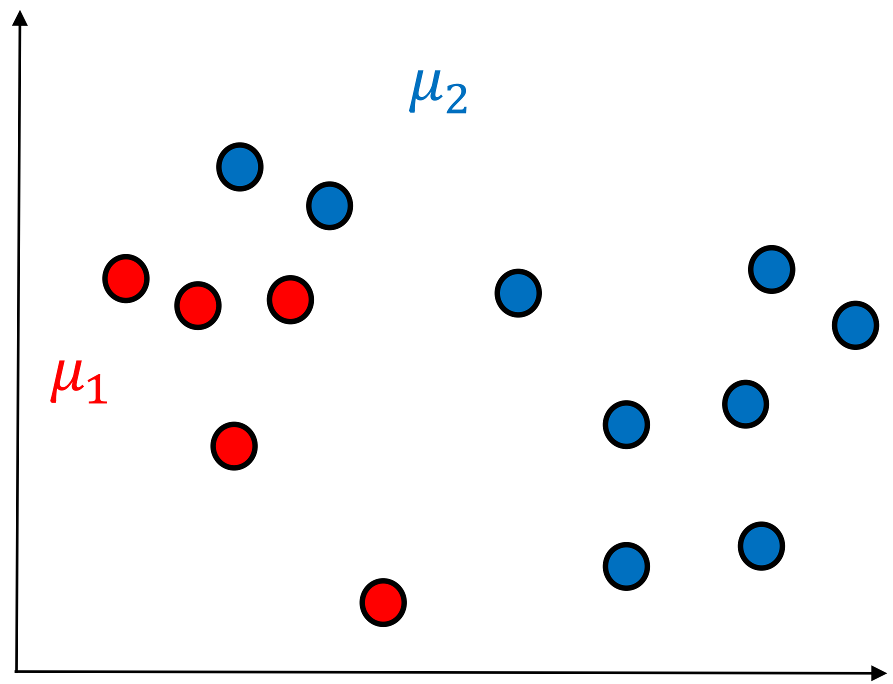
- EM(Expectation and Maximization) algorithm for K-means
  - Expectation
    - Assign the data points to the nearest centroid
  - Maximization
    - Update the centroid positions given the assignments



K initial means chosen randomly

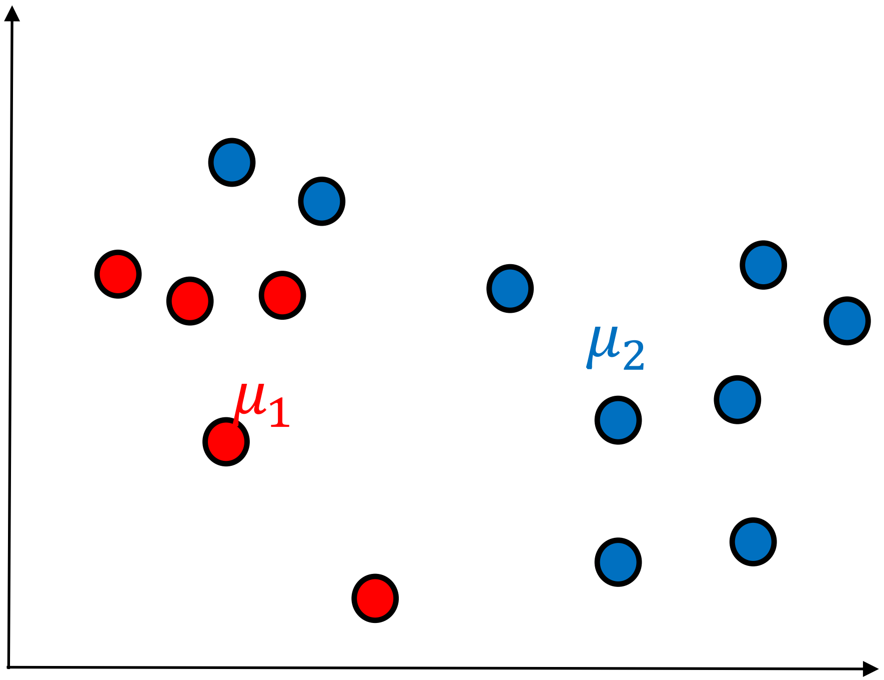
K-means clustering

- EM(Expectation and Maximization) algorithm for K-means
  - Expectation
    - Assign the data points to the nearest centroid



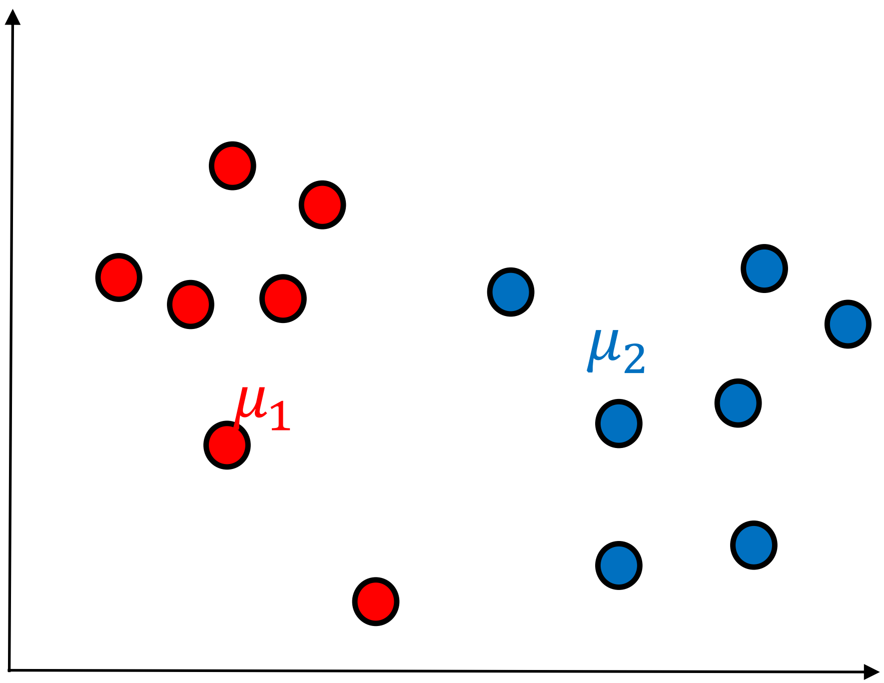
K-means clustering

- EM(Expectation and Maximization) algorithm for K-means
  - Maximization
    - Update the centroid positions given the assignments



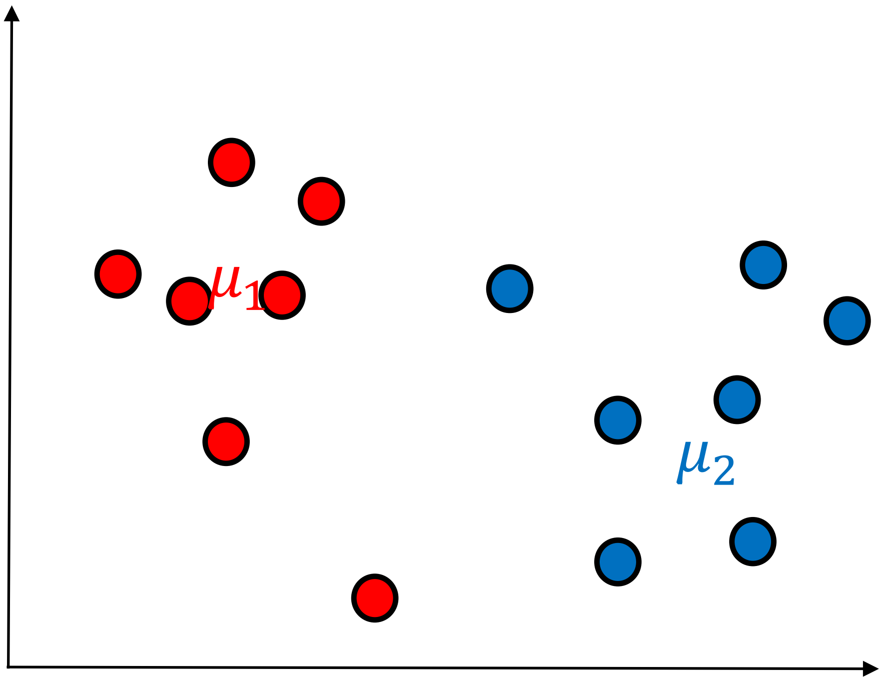
K-means clustering

- EM(Expectation and Maximization) algorithm for K-means
  - Expectation
    - Assign the data points to the nearest centroid



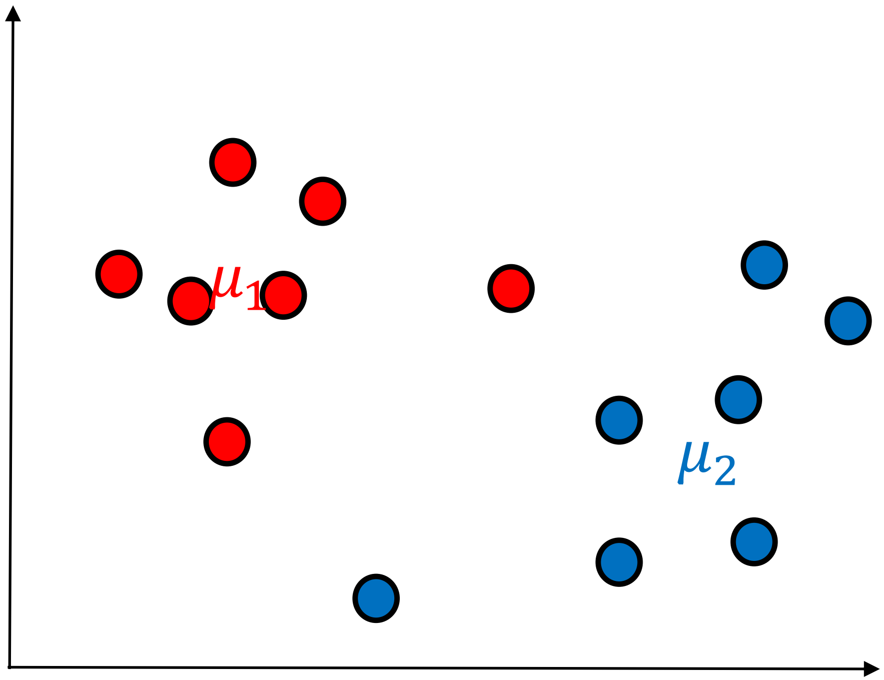
K-means clustering

- EM(Expectation and Maximization) algorithm for K-means
  - Maximization
    - Update the centroid positions given the assignments



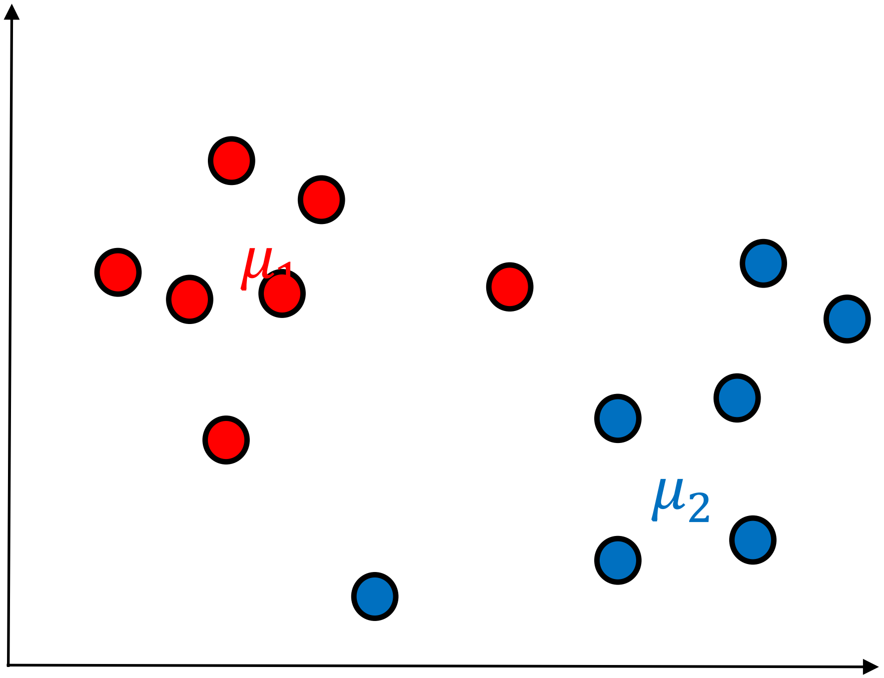
K-means clustering

- EM(Expectation and Maximization) algorithm for K-means
  - Expectation
    - Assign the data points to the nearest centroid



K-means clustering

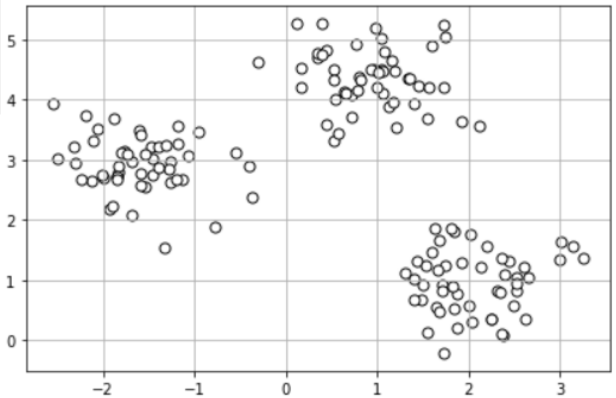
- EM(Expectation and Maximization) algorithm for K-means
  - Maximization
    - Update the centroid positions given the assignments



K-means clustering using scikit-learn

- Make simple data and plotting

```
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=150,
                  n_features=2,
                  centers=3,
                  cluster_std=0.5,
                  shuffle=True,
                  random_state=0)

import matplotlib.pyplot as plt

plt.scatter(X[:, 0], X[:, 1],
            c='white', marker='o', edgecolor='black', s=50)
plt.grid()
plt.tight_layout()

plt.show()
```



K-means clustering using scikit-learn

- Train the K-means model and Plot

```
from sklearn.cluster import KMeans

# KMeans with k = 2
km = KMeans(n_clusters=2,
            init='random',
            max_iter=300,
            random_state=0)

# cluster assignment
y_km = km.fit_predict(X)
y_km
```

```
array([0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1,
       1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0,
       1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1,
       0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0])
```

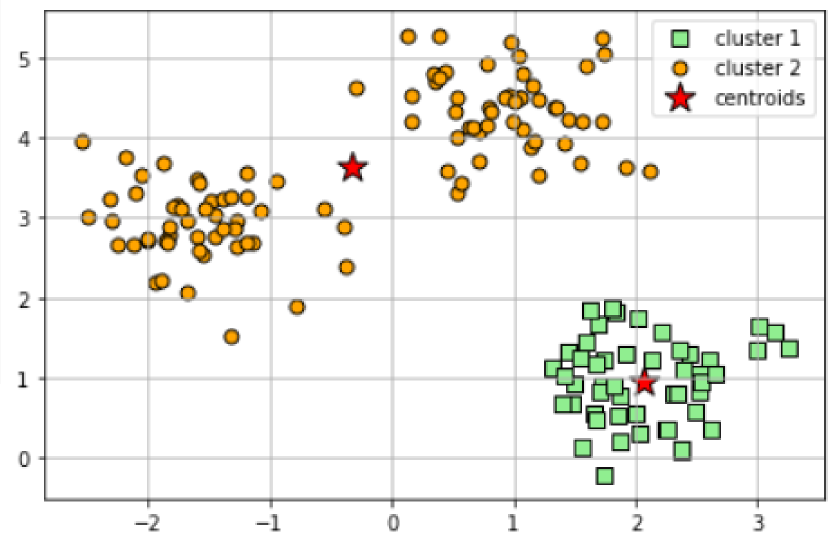
```
# cluster centers
print('<Cluster centers>\n', km.cluster_centers_)
```

```
<Cluster centers>
 [[ 2.06521743  0.96137409]
 [-0.33088235  3.63828839]]
```

K-means clustering using scikit-learn

- Train the K-means model and Plot

```
plt.scatter(X[y_km == 0, 0],
            X[y_km == 0, 1],
            s=50, c='lightgreen',
            marker='s', edgecolor='black',
            label='cluster 1')
plt.scatter(X[y_km == 1, 0],
            X[y_km == 1, 1],
            s=50, c='orange',
            marker='o', edgecolor='black',
            label='cluster 2')
plt.scatter(km.cluster_centers_[:, 0],
            km.cluster_centers_[:, 1],
            s=250, marker='*',
            c='red', edgecolor='black',
            label='centroids')

plt.legend(scatterpoints=1)
plt.grid()
plt.tight_layout()
plt.show()
```



K-means clustering using scikit-learn

- Train the K-means model and Plot

```
from sklearn.cluster import Kmeans

km = KMeans(n_clusters=3,
            init='random',
            max_iter=300,
            random_state=0)

y_km = km.fit_predict(X)
y_km
```

```
array([2, 1, 1, 1, 2, 1, 1, 2, 0, 1, 2, 0, 0, 1, 1, 0, 0, 2, 0, 2, 1, 2,
       1, 1, 0, 2, 2, 1, 0, 2, 0, 0, 0, 0, 1, 2, 2, 2, 1, 1, 0, 0, 1, 2,
       2, 2, 0, 1, 0, 1, 2, 1, 1, 2, 2, 0, 1, 2, 0, 1, 0, 0, 0, 0, 1, 0,
       1, 2, 1, 1, 1, 2, 2, 1, 2, 1, 1, 0, 0, 1, 2, 2, 1, 1, 2, 2, 2, 0,
       0, 2, 2, 1, 2, 1, 2, 1, 0, 0, 2, 2, 2, 2, 0, 2, 2, 1, 0, 1, 1, 1,
       0, 1, 2, 0, 1, 0, 1, 1, 0, 0, 1, 2, 1, 1, 2, 2, 0, 2, 0, 0, 0, 0,
       2, 0, 0, 0, 1, 0, 2, 0, 1, 1, 2, 2, 0, 0, 0, 0, 2, 2])
```

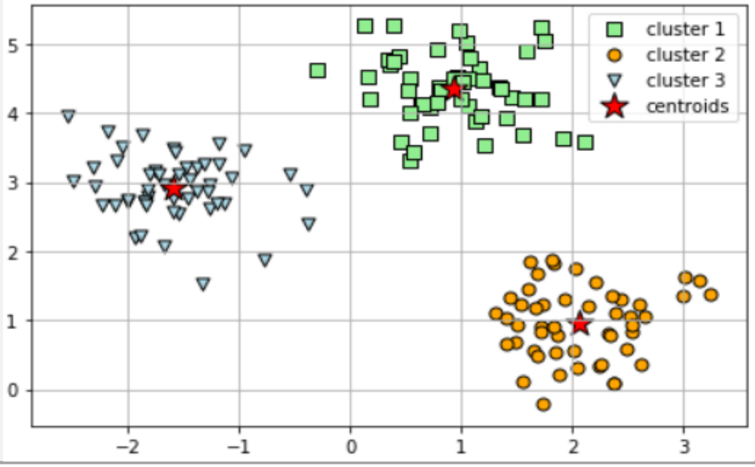
```
# cluster centers
print('<Cluster centers>\n', km.cluster_centers_)
```

```
<Cluster centers>
 [[ 0.9329651   4.35420712]
 [ 2.06521743  0.96137409]
 [-1.5947298   2.92236966]]
```

K-means clustering using scikit-learn

- Train the K-means model and Plot

```
plt.scatter(X[y_km == 0, 0],
            X[y_km == 0, 1],
            s=50, c='lightgreen',
            marker='s', edgecolor='black',
            label='cluster 1')
plt.scatter(X[y_km == 1, 0],
            X[y_km == 1, 1],
            s=50, c='orange',
            marker='o', edgecolor='black',
            label='cluster 2')
plt.scatter(X[y_km == 2, 0],
            X[y_km == 2, 1],
            s=50, c='lightblue',
            marker='v', edgecolor='black',
            label='cluster 3')
plt.scatter(km.cluster_centers_[:, 0],
            km.cluster_centers_[:, 1],
            s=250, marker='*',
            c='red', edgecolor='black',
            label='centroids')

plt.legend(scatterpoints=1)
plt.grid()
plt.tight_layout()
plt.show()
```

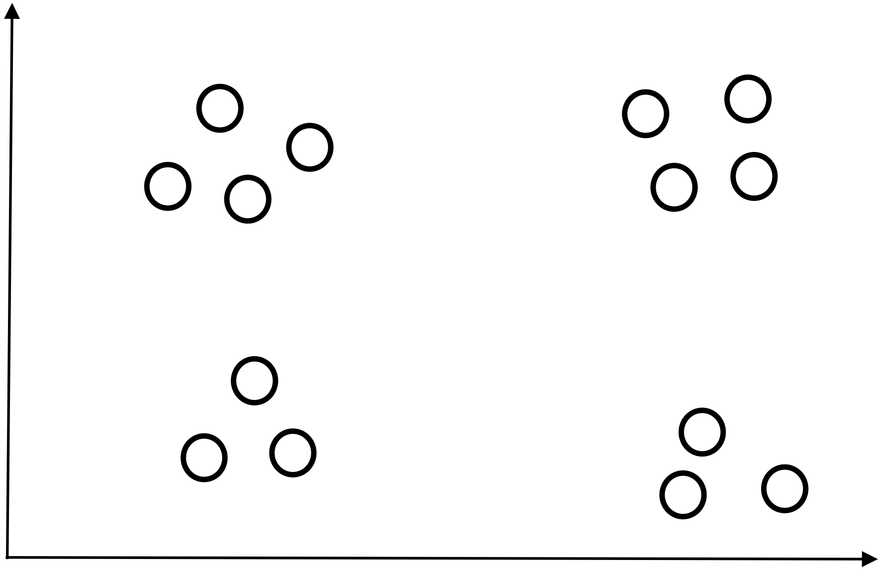


K-means++

- Concept
  - Classic K-means
    - It can sometimes result in bad clustering or slow convergence if the initial centroids are chosen poorly
  - Algorithm of K-means++
    1. Initialize an empty set M to store the k centroids being selected
    2. Randomly choose the first centroid $\mu^{(j)}$ from the input samples and assign it to M
    3. For each sample $x^{(i)}$ that is not in M, find the minimum squared distance $d(x^{(i)}, M)^2$ to any of the centroids in M
    4. To randomly select the next centroid $\mu^{(p)}$, use a weighted probability distribution equal to $\frac{d(x^{(p)}, M)^2}{\sum_i d(x^{(i)}, M)^2}$
    5. Repeat steps 2 and 3 until k centroids are chosen
    6. Proceed with the classic K-means algorithm

K-means++

- Algorithm of K-means++

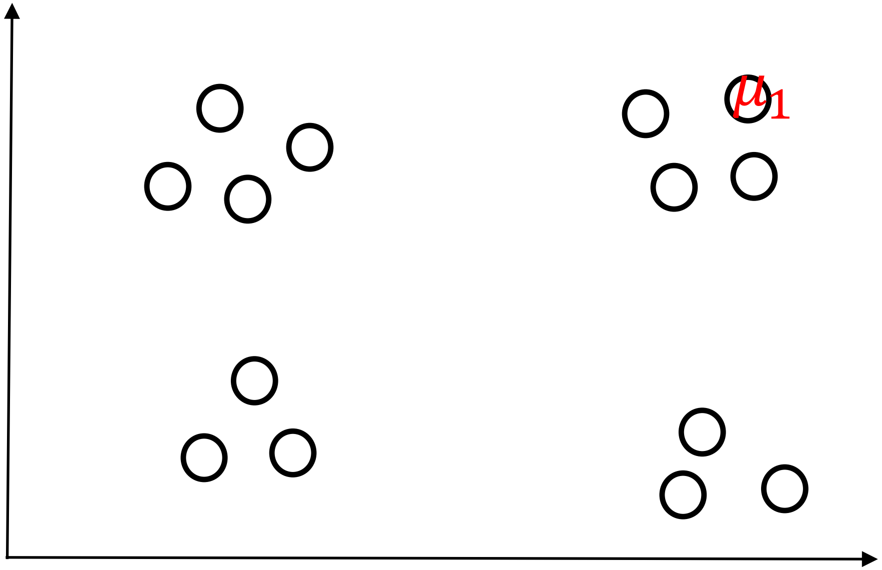


$M = \{\}$

dongguk UNIVERSITY

K-means++

- Algorithm of K-means++



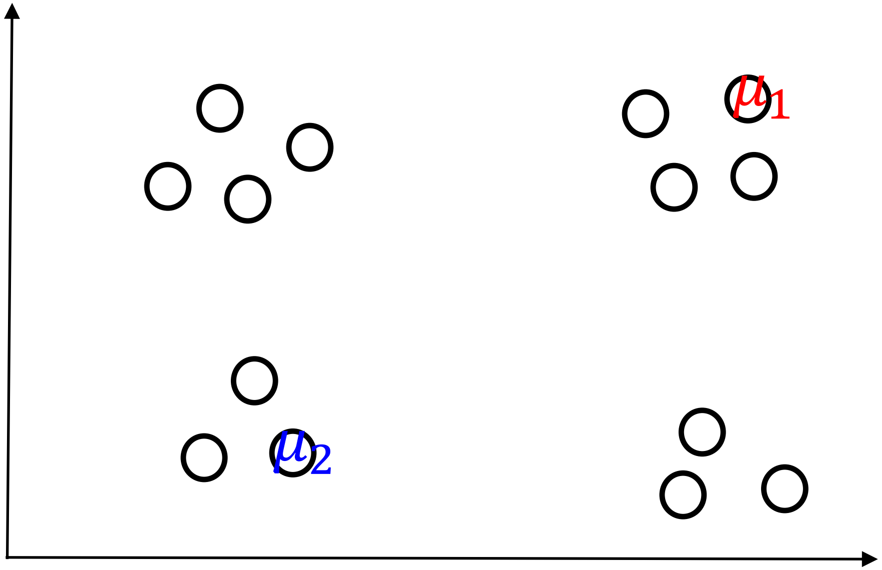
$$M = \{\mu_1, \}$$

K-means++

- Algorithm of K-means++

select the next centroid using weight probability

$$\frac{d(x^{(p)}, M)^2}{\sum_i d(x^{(i)}, M)^2}$$



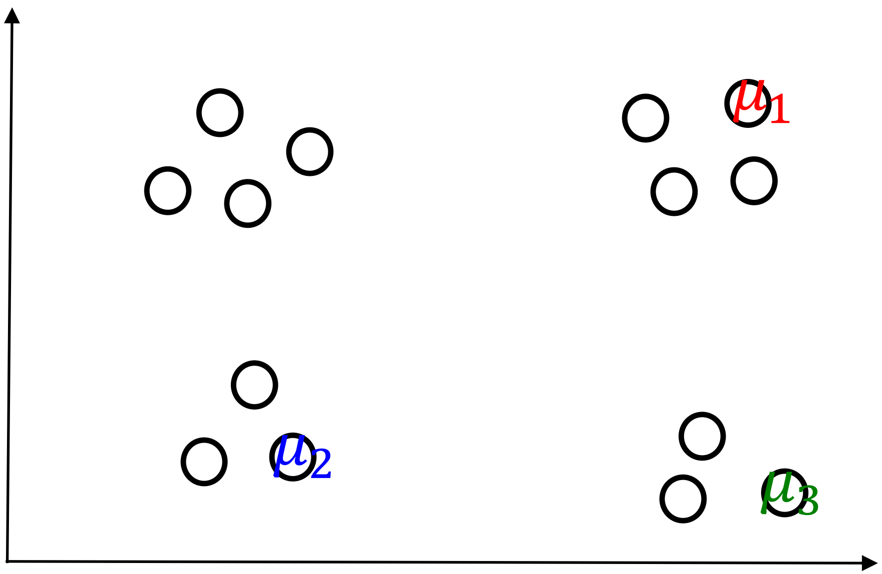
$$M = \{\mu_1, \mu_2, \}$$

K-means++

- Algorithm of K-means++

select the next centroid using weight probability

$$\frac{d(x^{(p)}, M)^2}{\sum_i d(x^{(i)}, M)^2}$$



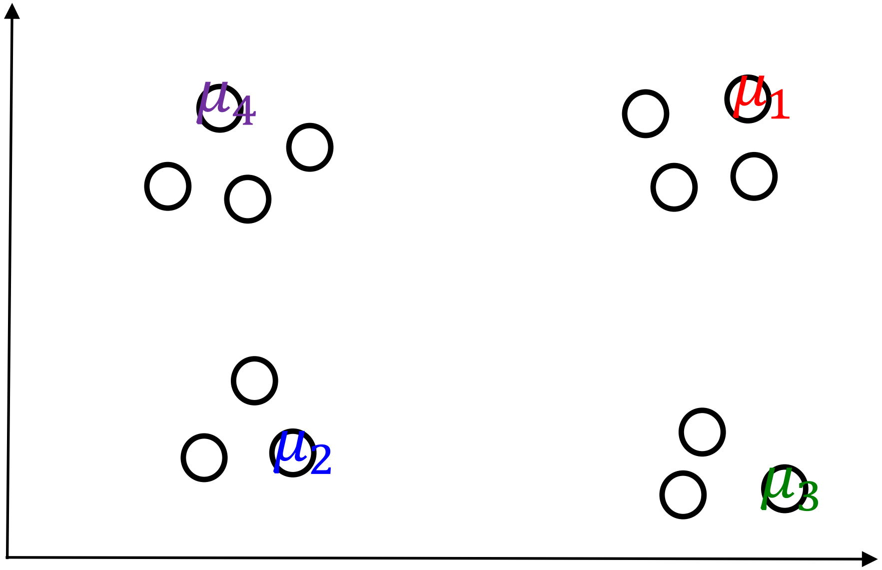
$$M = \{\mu_1, \mu_2, \mu_3\}$$

K-means++

- Algorithm of K-means++

select the next centroid using weight probability

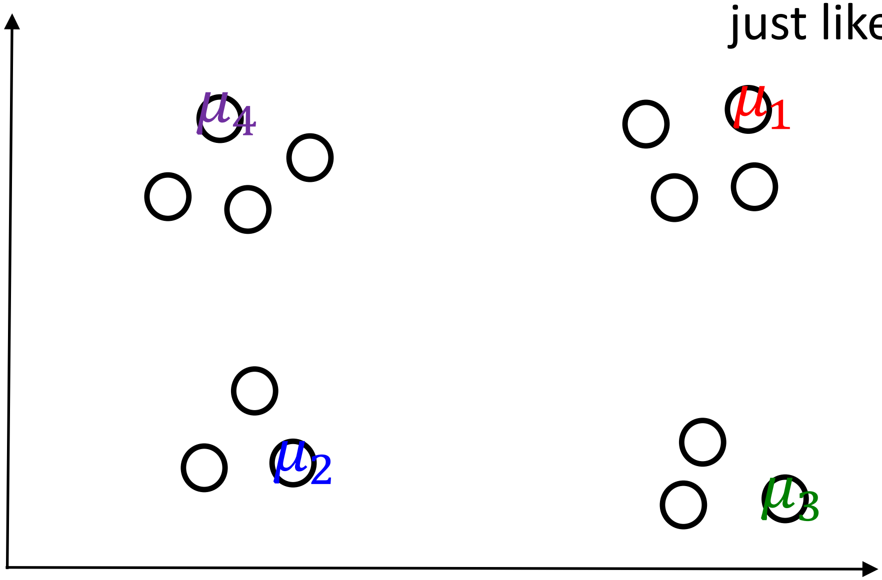
$$\frac{d(x^{(p)}, M)^2}{\sum_i d(x^{(i)}, M)^2}$$



$$M = \{\mu_1, \mu_2, \mu_3, \mu_4\}$$

K-means++

- Algorithm of K-means++



And then, apply vanilla K-means!
You can use K-means in scikit-learn just like this

```
from sklearn.cluster import Kmeans

km = KMeans(n_clusters=3,
            init='k-means++',
            n_init=10,
            …
```

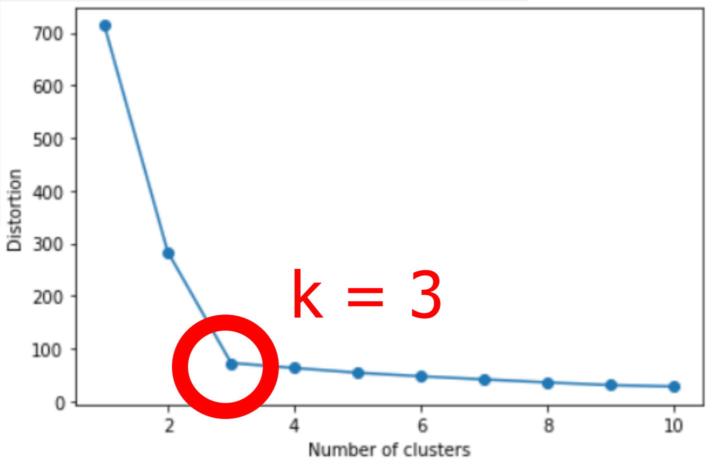
$$M = \{\mu_1, \mu_2, \mu_3, \mu_4\}$$

Distortion

- Using the elbow method to find the optimal number of clusters

```python
print('Distortion: %.2f' % km.inertia_)

# plotting distortions for k = 1 to 11
distortions = []
for i in range(1, 11):
      km = KMeans(n_clusters=i,
                  init='k-means++',
                  max_iter=300,
                  random_state=0)
      km.fit(X)
      distortions.append(km.inertia_)
plt.plot(range(1, 11), distortions, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.tight_layout()
plt.show()
```

```
Distortion: 72.48
```



Silhouette Analysis

- Silhouette Analysis
  - One of intrinsic metric to evaluate the quality of a clustering
    - Can be used as a graphical tool to plot a measure of how tightly grouped the samples in the clusters are
  - Algorithm
    1. Calculate **the cluster cohesion $\boldsymbol{a}^{(i)}$** as the average distance between a sample $x^{(i)}$ and all other points in the same cluster

    2. Calculate **the cluster separation $\boldsymbol{b}^{(i)}$** from the next closest cluster as the average distance between the sample $x^{(i)}$ and all samples in the nearest cluster

    3. Calculate **the silhouette $\boldsymbol{s}^{(i)}$** as the difference between cluster cohesion and separation divided by the greater of the two, as shown here:

$$s^{(i)} = \frac{b^{(i)} - a^{(i)}}{max\{b^{(i)}, a^{(i)}\}}$$

Silhouette Analysis

- Silhouette Analysis
  - Average Silhouette Width
    - Average of $s^{(i)}$
    - How to use the value

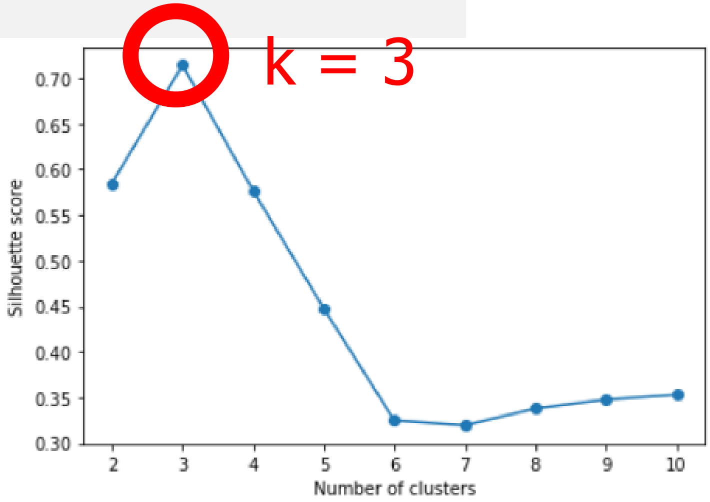
Range of SC	Interpretation
0.71-1.0	A strong structure has been found
0.51-0.70	A reasonable structure has been found
0.26-0.50	The structure is weak and could be artificial
< 0.25	No substantial structure has been found

https://www.stat.berkeley.edu/~spector/s133/Clus.html

Silhouette Analysis using scikit-learn

- Measuring the quality of clustering via silhouette plots

```
from sklearn.metrics import silhouette_score

# plotting silhouette scores for k = 2 to 11
silhouette_scores = []
for i in range(2, 11):
     km = KMeans(n_clusters=i,
                 init='k-means++',
                 max_iter=300,
                 random_state=0)
     y_km = km.fit_predict(X)
     silhouette_scores.append(silhouette_score(X, y_km, metric='euclidean'))

plt.plot(range(2, 11), silhouette_scores, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette score')
plt.tight_layout()
plt.show()
```



Example

- Make simple 3D data and plotting

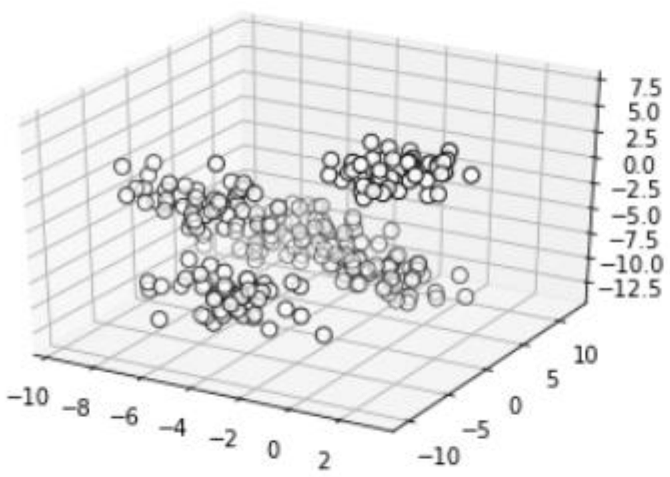
```
from sklearn.datasets import make_blobs

# make simple 3D data
X, y = make_blobs(n_samples=250,
                  n_features=3,
                  centers=5,
                  cluster_std=1.5,
                  shuffle=True,
                  random_state=1)
X
```

```
array([[ -3.53629743e+00,  -6.02983465e+00,  -9.53019216e+00],
       [  1.81801027e+00,  -9.91484955e-01,   3.15589155e+00],
       [ -4.32462107e+00,  -7.40257340e+00,  -7.58369028e+00],
       [  1.85916632e+00,  -9.24290830e-01,   1.72209806e+00],
       [ -2.78149052e+00,   3.53529403e+00,  -1.01638435e+01],
       [ -4.67636968e+00,  -2.97755645e+00,   4.29766489e-01],
       [ -8.02808319e+00,  -1.20932028e+00,  -2.57803932e+00],
       [ -9.97923558e-01,  -2.52376196e+00,   2.57342840e+00],
       [ -4.84489985e+00,  -3.74014658e+00,  -1.87979408e-01],
       [ -1.38773851e+00,   5.25300717e+00,  -1.08474778e+01],…
```

Example

- Make simple 3D data and plotting

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# plotting simple 3D data
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], s=50, c='white', edgecolor='black')
plt.show()
```



dongguk UNIVERSITY

Example

- Train the K-means model and Plot

```python
from sklearn.cluster import KMeans

# KMeans with k = 3
km = KMeans(n_clusters=3,
           init='random',
           max_iter=300,
           random_state=0)

# cluster assignment
y_km = km.fit_predict(X)
y_km
```

```
array([2, 0, 2, 0, 1, 2, 2, 0, 2, 1, 1, 2, 1, 1, 1, 1, 2, 2, 1, 1, 1, 2, 2,
       2, 2, 1, 2, 0, 2, 1, 1, 0, 0, 1, 2, 0, 1, 0, 1, 2, 2, 2, 2, 2, 0, 0,
       2, 2, 0, 2, 0, 2, 1, 2, 2, 1, 0, 2, 2, 1, 1, 0, 1, 1, 0, 0, 2, 2, 1,
       2, 2, 1, 0, 1, 1, 2, 0, 0, 0, 2, 1, 2, 1, 1, 2, 1, 0, 2, 1, 2, 2, 0,
       1, 0, 2, 1, 2, 2, 2, 1, 2, 0, 1, 0, 1, 0, 2, 1, 2, 1, 0, 1, 1, 1, 2,
       1, 1, 1, 0, 2, 1, 1, 1, 2, 1, 2, 2, 1, 2, 1, 0, 1, 2, 2, 0, 2, 2, 2,
       1, 2, 1, 1, 0, 2, 1, 2, 0, 0, 1, 0, 2, 2, 0, 2, 2, 2, 0, 1, 1, 1, 1,
       1, 0, 2, 2, 2, 2, 1, 1, 2, 1, 1, 0, 0, 1, 2, 1, 1, 2, 2, 0, 0, 2, 1,
       2, 0, 1, 1, 2, 1, 2, 0, 1, 1, 2, 2, 0, 1, 2, 0, 2, 1, 0, 2, 1, 1, 2,
       1, 2, 1, 1, 0, 1, 1, 2, 2, 2, 1, 1, 1, 1, 0, 1, 2, 1, 1, 1, 1, 1, 1,
       0, 2, 2, 2, 0, 2, 2, 1, 2, 1, 2, 2, 0, 2, 2, 1, 1, 2, 1, 1])
```

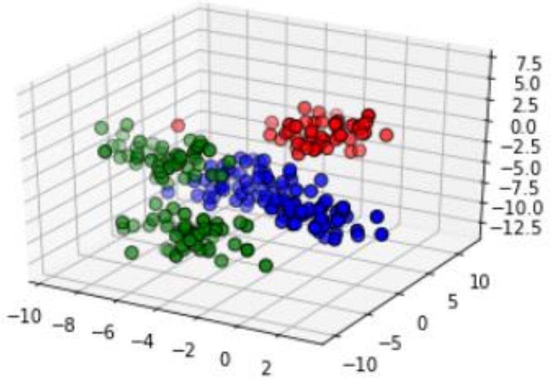
Example

- Train the K-means model and Plot

```
# cluster centers
print('<Cluster centers>\n', km.cluster_centers_)
```

```
<Cluster centers>
 [[ 2.06521743  0.96137409]
 [-0.33088235  3.63828839]]
```

```
# plotting clusters
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[y_km == 0, 0], X[y_km == 0, 1], X[y_km == 0, 2],
           s=50, c='red', edgecolor='black')
ax.scatter(X[y_km == 1, 0], X[y_km == 1, 1], X[y_km == 1, 2],
           s=50, c='blue', edgecolor='black')
ax.scatter(X[y_km == 2, 0], X[y_km == 2, 1], X[y_km == 2, 2],
           s=50, c='green', edgecolor='black')
plt.show()
```

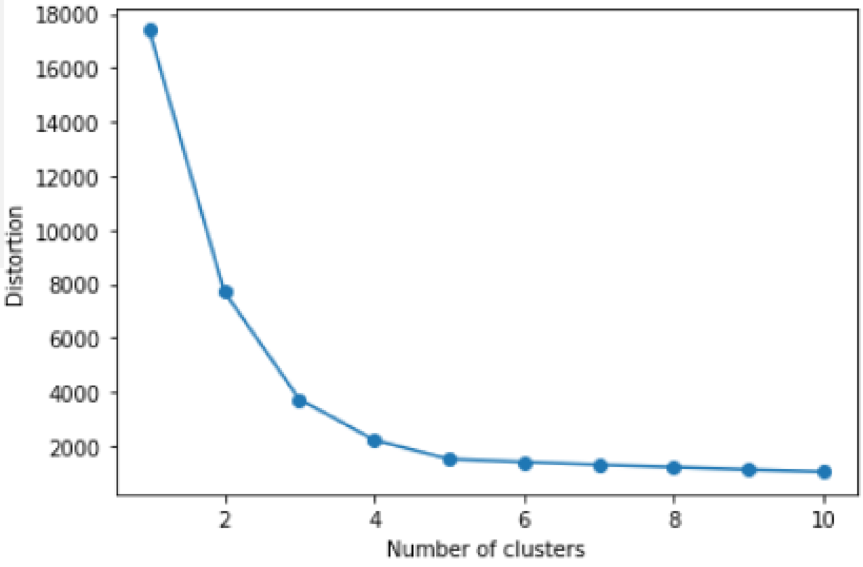


dongguk UNIVERSITY

Example

- Using the elbow method to find the optimal number of clusters

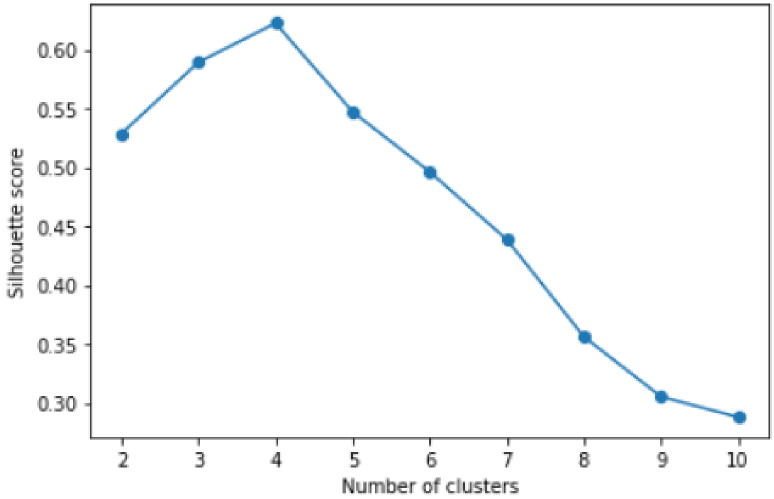
```
# plotting distortions for k = 1 to 11
distortions = []
for i in range(1, 11):
     km = KMeans(n_clusters=i,
                 init='k-means++',
                 max_iter=300,
                 random_state=0)
     km.fit(X)
     distortions.append(km.inertia_)
plt.plot(range(1, 11), distortions, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.tight_layout()
plt.show()
```



Example

- Measuring the quality of clustering via silhouette plots

```
# plotting silhouette scores for k = 1 to 11
silhouette_scores = []
for i in range(2, 11):
     km = KMeans(n_clusters=i,
                 init='k-means++',
                 max_iter=300,
                 random_state=0)
     y_km = km.fit_predict(X)
     silhouette_scores.append(silhouette_score(X, y_km, metric='euclidean'))

plt.plot(range(2, 11), silhouette_scores, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette score')
plt.tight_layout()
plt.show()
```

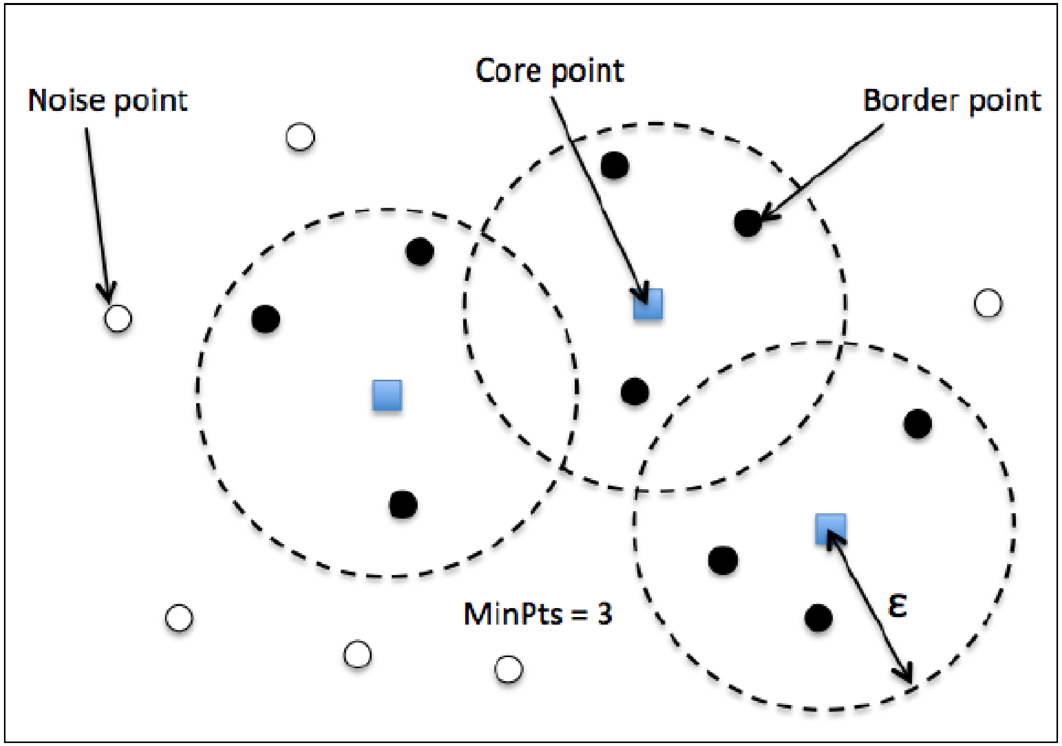


DBSCAN

- Density-Based Spatial Clustering of Applications with Noise
  - DBSCAN groups data in dense region
  - Density parameters
    - ***Radius*** ε : Distance to determine the neighborhood
    - ***MinPts*** : Minimum number of points in neighborhood
- In DBSCAN, a special label is assigned to each sample (point)
  - Core point : It contains *MinPts* of neighboring points within *radius* ε
  - Border point : It has fewer neighbors than MinPts within ε, but lies within the ε radius of a core point
  - Noise points(outliers) : All other points that are neither core nor border points are considered as noise points

DBSCAN

- Density-Based Spatial Clustering of Applications with Noise
  - MinPts = 4



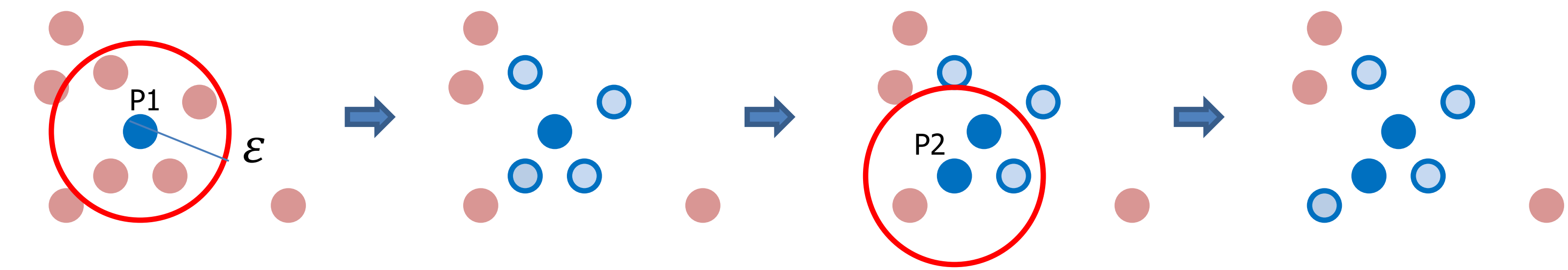
DBSCAN

- DBSCAN algorithm
  - A *cluster* - a maximal set of density-connected points
  - Discovers clusters of arbitrary shape in databases with noise

1. Arbitrary select a point ***p***
2. Retrieve all ε-neighborhood of ***p***
3. If ***p*** is a core object, a cluster is formed
4. From each core object ***p***, iteratively collects density-reachable objects (may merge clusters)
5. Continue the process until no new points can be added

DBSCAN

- DBSCAN algorithm example
  - MinPts = 4 (example)

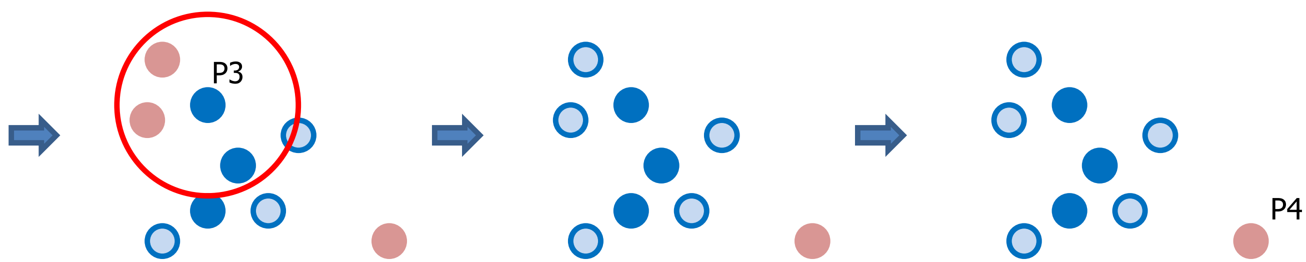


P1 is a core point

Density-reachable points form a cluster

P2 is a core point

Cluster extends



P3 is a core point

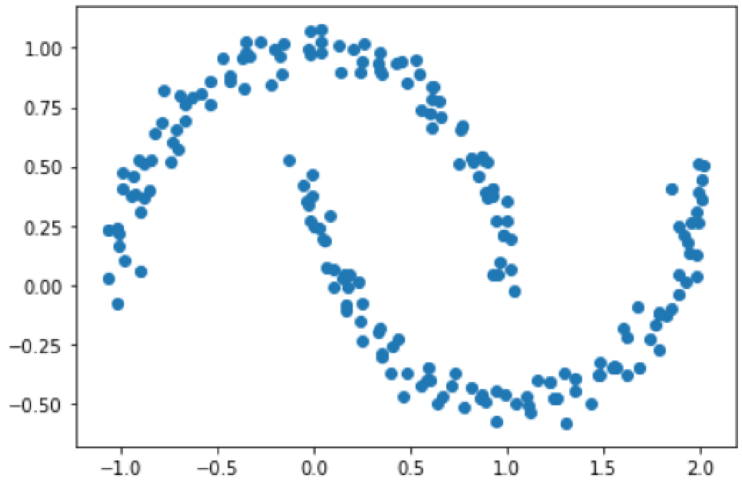
Cluster extends

No more core points.
P4 is an outlier

DBSCAN clustering using scikit-learn

- Make simple data and plotting

```
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt

X, y = make_moons(n_samples=200, noise=0.05, random_state=0)
plt.scatter(X[:, 0], X[:, 1])
plt.tight_layout()

plt.show()
```



DBSCAN clustering using scikit-learn

- Problem of K-means clustering

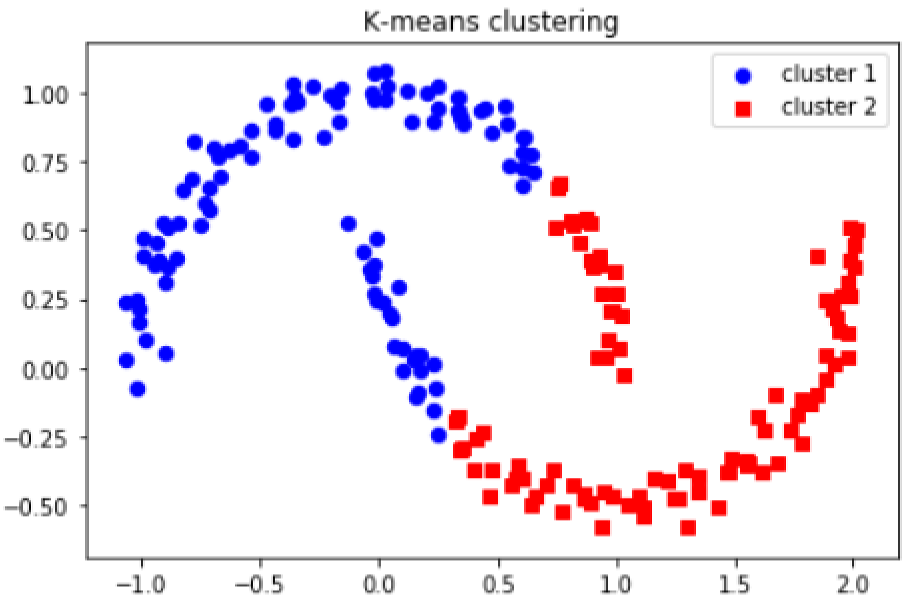
```python
from sklearn.cluster import KMeans
# KMeans with k = 2
km = KMeans(n_clusters=2, random_state=0)
y_km = km.fit_predict(X)
y_km
```

```
array([1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0,
       1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0,
       0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1,
       1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1])
```

DBSCAN clustering using scikit-learn

- Problem of K-means clustering

```
# plotting clusters
f, (ax1)= plt.subplots(1, 1, figsize=(6, 4))
ax1.scatter(X[y_km == 0, 0], X[y_km == 0, 1],
           c='blue', marker='o', s=40, label='cluster 1')
ax1.scatter(X[y_km == 1, 0], X[y_km == 1, 1],
           c='red', marker='s', s=40, label='cluster 2')
ax1.set_title('K-means clustering')
plt.legend()
plt.tight_layout()

plt.show()
```



DBSCAN clustering using scikit-learn

- Density based clustering

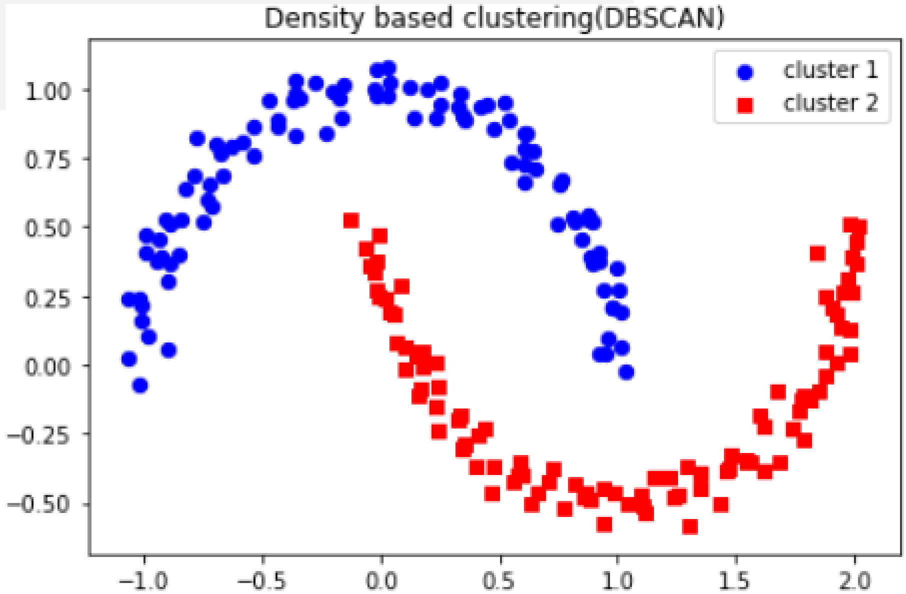
```
from sklearn.cluster import DBSCAN
# DBSCAN with eps = 0.2, min_samples = 5
db = DBSCAN(eps=0.2, min_samples=5, metric='euclidean')
y_db = db.fit_predict(X)
y_db
```

```
array([0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0,
       0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0,
       0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0,
       1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1], dtype=int64)
```

DBSCAN clustering using scikit-learn

- Density based clustering

```
# plotting clusters
plt.scatter(X[y_db == 0, 0], X[y_db == 0, 1],
           c='blue', marker='o', s=40,
           label='cluster 1')
plt.scatter(X[y_db == 1, 0], X[y_db == 1, 1],
           c='red', marker='s', s=40,
           label='cluster 2')
plt.legend()
plt.title('Density based clustering(DBSCAN)')
plt.tight_layout()
plt.show()
```



DBSCAN clustering using scikit-learn

- Outlier Detection

```
# make outliers
X[0] = [1, 1]
X[1] = [-0.5, -0.5]
```

```
# DBSCAN with eps = 0.2, min_samples = 5
db = DBSCAN(eps=0.2, min_samples=5, metric='euclidean')
y_db = db.fit_predict(X)
y_db
```

```
array([-1, -1,  0,  1,  0,  0,  1,  0,  1,  0,  1,  0,  0,  0,  1,  1,  1,
        0,  1,  1,  0,  0,  1,  0,  1,  0,  0,  0,  0,  1,  1,  1,  0,  0,
        1,  0,  0,  1,  1,  0,  0,  1,  1,  0,  0,  1,  1,  1,  0,  0,  1,
        0,  0,  1,  0,  1,  1,  0,  1,  1,  0,  1,  0,  1,  0,  1,  1,  0,
        1,  1,  0,  1,  0,  0,  0,  1,  0,  1,  1,  0,  0,  1,  0,  0,  0,
        1,  1,  1,  0,  0,  1,  1,  0,  1,  0,  0,  0,  0,  1,  0,  0,  0,
        1,  1,  1,  0,  1,  1,  0,  1,  1,  1,  1,  1,  1,  0,  1,  0,  0,
        1,  1,  1,  0,  1,  0,  1,  1,  0,  0,  0,  1,  1,  1,  0,  0,  0,
        0,  1,  0,  1,  0,  0,  1,  1,  1,  1,  0,  0,  1,  0,  0,  0,  1,
        1,  0,  1,  0,  0,  1,  1,  0,  0,  1,  0,  0,  0,  1,  0,  0,  0,
        1,  1,  1,  1,  0,  0,  0,  1,  1,  1,  0,  1,  0,  0,  0,  1,  1,
        0,  1,  1,  1,  1,  1,  1,  0,  1,  0,  0,  1,  0], dtype=int64)
```

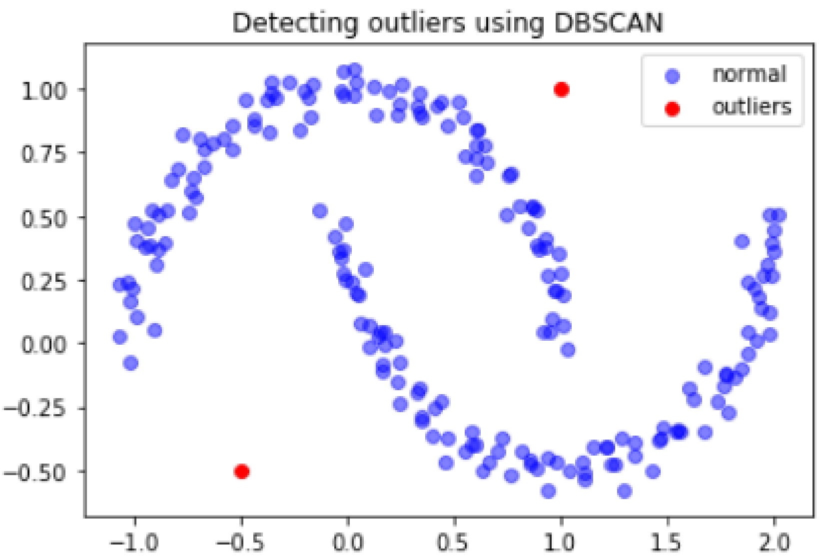
DBSCAN clustering using scikit-learn

- Outlier Detection

```
# find data that is not clustered (-1) - outlier
outlier_idxs = (y_db == -1)
X[outlier_idxs]
```

```
array([[ 1. ,  1. ],
       [-0.5, -0.5]])
```

```
# plotting outliers
plt.scatter(X[~outlier_idxs, 0],
            X[~outlier_idxs, 1],
            alpha=0.5, color='b', label='normal')
plt.scatter(X[outlier_idxs, 0],
            X[outlier_idxs, 1],
            color='r', label='outliers')
plt.legend()
plt.title("Detecting outliers using DBSCAN")
plt.show()
```



Submit

- To make sure if you have completed this practice,
  Submit your practice file(Week11_givencode.ipynb) to e-class.
- **Deadline : tomorrow 11:59pm**
- Modify your ipynb file name as “Week11_StudentNum_Name.ipynb”
  Ex) **Week11_2020123456_홍길동.ipynb**
- You can upload this file without taking the quiz, but homework will be provided like a quiz every three weeks, so it is recommended to take the quiz as well.

Quiz1 : K-means Clustering

- Figure out the data structure of unlabeled data
- Motor Trend Dataset
  - The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).
- Goal
  - We would like to recommend the products that meet the needs of customers by classifying the cars in the list into several clusters according to their features.
  - After clustering, Explain each cluster based on its features

https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/mtcars.html

Quiz2 : DBSCAN

- Detecting outliers using DBSCAN
- Iris Dataset
- Goal
  - Detect & Remove outliers from the iris dataset