10. PCA와 비지도학습

- 차원의 저주
- PCA(Principal Component Analysis)
- 비지도 학습

차원의 저주

차원의 저주

- **차원의 저주(Curse of dimensionality)**
  - 고차원 공간에 있는 데이터를 분석할 때 발생하는 여러가지 현상
  - 수학적 공간 차원이 늘어나면 문제의 계산법이 지수적으로 커지는 문제
    - 위키피디아
      - "The curse of dimensionality refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces (often with hundreds or thousands of dimensions) that do not occur in low-dimensional settings such as the three-dimensional physical space of everyday experience."
- **(예)**
  - 데이터가 차원은 높은데 개수가 적고, 이 데이터를 학습하여 분류하는 모델이 있다고 해보자.
  - 이런 경우에 이 모델은 주어진 데이터에 과대적합한 모델이 된다.

- **문제점**
  - 예측을 위해 훨씬 많은 작업을 해야 하고 과적합이 되어 저차원 일때보다 예측이 불안정해짐
  - 입력 변수의 수가 너무 많으면 잡음(noise)이 발생하여 분류 모형의 정확도 감소함
  - 입력 변수 간에 상관관계가 있는 경우 모형이 불안정해짐
- **해결방법**
  - 차원을 축소시키거나, 데이터를 많이 획득하는 방법
  - 효과
    - 모형의 복잡도를 낮춰 예측 모델의 성능을 개선할 수 있음
    - 모형의 정확도를 높일 수 있고, 모델 학습 속도가 향상되며, 데이터 시각화가 쉬워 짐

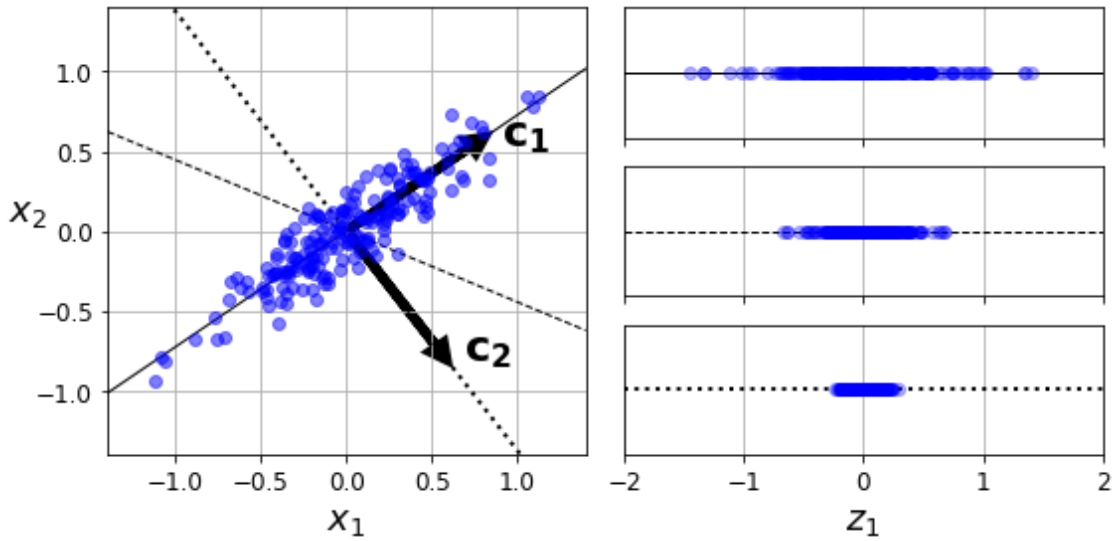
차원축소방법

- 특징선택(feature selection)
  - 변수들 중에 중요한 변수만 몇 개 고르고 나머지는 버리는 방법
  - 변수 간에 중첩이 있는지, 어떤 변수가 중요한 변수인지, 어떤 변수가 종속변수에 영향을 크게 주는 변수인지를 분석할 필요가 있음
    - **변수 간의 중첩을 확인하는 방법으로 상관관계(correlation)을 주로 사용**
- 특징추출(feature extraction)
  - 변수들을 조합하여 데이터를 잘 표현할 수 있는 중요성분을 가진 새로운 변수를 추출하는 방법
  - 주성분분석(PCA, Principal Component Analysis)
- 선형판별분석(LDA: Linear Discrimination Analysis)
  - 학습과정 중 클래스를 가장 잘 구분하는 축(axis)을 학습

PCA(Principal Component Analysis)

PCA(Principal Component Analysis)

- 데이터의 분산(variance)를 최대한 보존하면서 서로 직교하는 principal axis(주성분)를 찾아, 고차원 공간의 표본들을 저 차원 공간으로 변환하는 기법
  - 데이터를 잘 표현하는 초평면(hyperplane)을 정의한 후, 데이터를 이 초평면에 투영(projection)
    - 분산이 최대로 보존되는 축을 선택(아래 그림에서는 c_1)



주성분을 어떻게 찾을까?

- 특이값 분해(Singular Value Decomposition, SVD) 사용
  - 입력 데이터들의 공분산 행렬에 대한 고유값 분해(eigenvalue decomposition)
    - 고유벡터 : 주 성분 벡터. 데이터의 분포에서 분산이 큰 방향
    - 고유값 : 분산의 크기

- $A = U\Sigma V^T$
  - A: $m \times n$ matrix(주어진 $m \times n$ 행렬)
  - U: $m \times m$ orthogonal matrix(직교행렬)
  - Σ: $m \times n$ diagonal matrix(대각행렬)
  - V: $n \times n$ orthogonal matrix(직교행렬)

고유값 분해와 특이값 분해

- **고유값 분해**
  - $\mathbf{A = Q\Lambda Q^{-1}}$
    - $\mathbf{Q}$: $\mathbf{A}$의 고유벡터를 열에 배치한 행렬
    - $\mathbf{\Lambda}$: 고유값을 대각선에 배치한 대각행렬

- **특이값 분해(*n*m*행렬인 경우)**
  - 고유값 분해는 정방행렬만 분해 가능
  - 정방행렬이 아닌 경우에도 분해가 가능
  - $\mathbf{A = U\Sigma V^{-T}}$
    - $\mathbf{U}$: 특이행렬($\mathbf{AA^T}$의 고유벡터를 열에 배치한 n*n행렬)
    - $\mathbf{\Sigma}$: $\mathbf{AA^T}$의 고유값의 제곱근을 대각선에 배치한 대각행렬(n*m행렬)
    - $\mathbf{V^{-T}}$: 특이행렬($\mathbf{A^TA}$의 고유벡터를 열에 배치한 m*m행렬)

고유값분해(예1)

$\mathbf{A} = \mathbf{Q}\Lambda\mathbf{Q}^{-1}$

- $\mathbf{A} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$
  - $\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = 3\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ $\quad \lambda = 3, \mathbf{v} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$
  - $\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}\begin{pmatrix} 1 \\ -1 \end{pmatrix} = 1\begin{pmatrix} 1 \\ -1 \end{pmatrix}$ $\quad \lambda = 1, \mathbf{v} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$
  - $\mathbf{Q} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ $\quad \mathbf{Q}^{-1} = \frac{1}{(1*-1)-(1*1)}\begin{pmatrix} -1 & -1 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{pmatrix}$
  - $\Lambda = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}$

- $\mathbf{A} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{pmatrix}$

고유값분해(예2)

$\mathbf{A} = \mathbf{Q}\Lambda\mathbf{Q}^{-1}$

- $\mathbf{A} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 4 \\ 0 & 4 & 9 \end{pmatrix}$
  - $\begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 4 \\ 0 & 4 & 9 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix} = 11 \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix}$ $\quad \lambda = 11, \mathbf{v} = \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix}$
  - $\begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 4 \\ 0 & 4 & 9 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = 2 \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ $\quad \lambda = 2, \ \mathbf{v} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$
  - $\begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 4 \\ 0 & 4 & 9 \end{pmatrix} \begin{pmatrix} 0 \\ 2 \\ -1 \end{pmatrix} = 1 \begin{pmatrix} 0 \\ 2 \\ -1 \end{pmatrix}$ $\quad \lambda = 1, \ \mathbf{v} = \begin{pmatrix} 0 \\ 2 \\ -1 \end{pmatrix}$
  - $\mathbf{Q} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 2 \\ 2 & 0 & -1 \end{pmatrix}$ $\quad \mathbf{Q}^{-1} = \begin{pmatrix} 0 & 0.2 & 0.4 \\ 1 & 0 & 0 \\ 0 & 0.4 & -0.2 \end{pmatrix}$ $\quad \Lambda = \begin{pmatrix} 11 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
- $\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 2 \\ 2 & 0 & -1 \end{pmatrix} \begin{pmatrix} 11 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0.2 & 0.4 \\ 1 & 0 & 0 \\ 0 & 0.4 & -0.2 \end{pmatrix}$

특이값분해(예)

$\mathbf{A} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{-\mathrm{T}}$

- $\mathbf{A} = \begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix}$
- $\mathbf{A}\mathbf{A}^{\mathrm{T}} = \begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} -1 & 0 \\ 1 & -1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}$
  - $\mathbf{A}\mathbf{A}^{\mathrm{T}}$의 고유값을 λ_1과 λ_2라고 하면,
    - 고유값들의 합: 대각요소들의 합, 고유값들의 곱: 행렬식
    - $\lambda_1 + \lambda_1 = 4$, $\lambda_1\lambda_1 = 3$ $\quad\quad \det(\mathbf{A}\mathbf{A}^{\mathrm{T}}) = ad - bc = 4 - 1 = 3$
    - 따라서 $\lambda_1 = 3 \quad \lambda_2 = 1$, 특이값 $\sqrt{\lambda_1} = \sqrt{3}$, $\sqrt{\lambda_2} = 1$
  - $\mathbf{A}\mathbf{A}^{\mathrm{T}}$의 고유벡터는 $\lambda_1 = 3 \quad \lambda_2 = 1$ 일 때($\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$를 만족해야...)
    - $\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \end{pmatrix} = 3 \begin{pmatrix} -1 \\ 1 \end{pmatrix}$, $\mathbf{v} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$ $\quad$ $\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 1 \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, $\mathbf{v} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$

$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^{-\mathrm{T}}$

- $\Sigma = \begin{pmatrix} \sqrt{3} & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$
  - 특이값($\sqrt{3}$, 1)을 대각요소로 배치
- $\mathbf{U} = \begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix}$
  - 고유벡터가 $\mathbf{v} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$, $\mathbf{v} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ 이므로 $\mathbf{AA}^{\mathrm{T}}$의 고유벡터를 열에 배치
- $\mathbf{V}^{-\mathrm{T}}$는 $\mathbf{A}^{\mathrm{T}}\mathbf{A}$의 고유벡터를 열에 배치한 것이므로
  - $\mathbf{A}^{\mathrm{T}}\mathbf{A} = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix}$ 에 대한 고유벡터를 구하면 $\begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$
  - $\mathbf{V} = \begin{pmatrix} 1 & -1 & 1 \\ -2 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$, 남은 열은 정규직교벡터를 채움 $\mathbf{V}^{\mathrm{T}} = \begin{pmatrix} 1 & -2 & 1 \\ -1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

- 따라서 행렬 $\mathbf{A}$ 는 다음과 같이 분해됨
  - $\mathbf{A} = \begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} = \mathbf{U\Sigma V^{-T}}$

$$\begin{pmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} \sqrt{3} & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & -2 & 1 \\ -1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

sklearn.decomposition.**PCA**

```
class PCA(n_components=None,
          svd_solver='auto')
```

- **n_components: 유지할 주성분의 수(또는 비율)**
  - PCA(n_compoents=2)
  - PCA(n_compoents=0.95)
- **svd_solver:{‘auto’, ‘full’, ‘arpack’, ‘randomized’}**
  - full: full SVD(특이분해)를 실행
  - arpack: n_components로 잘린 SVD를 실행
  - randomized: Halko 등의 방법으로 무작위 SVD를 실행

Iris dataset

	sepal length	sepal width	petal length	petal width	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3					
4					
...					
145					
146					
147					
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

```
from sklearn.datasets import load_iris
import pandas as pd

iris = load_iris()
df = pd.DataFrame(iris.data,
     columns=['sepal length','sepal width','petal length','petal width'])
df['target']=iris.target
df
```

정규화

```
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
import pandas as pd

iris = load_iris()
X = StandardScaler().fit_transform(iris.data)
y = iris.target
df = pd.DataFrame(X,
     columns=['sepal length','sepal width','petal length','petal width'])
df['target']=y
df
```

	sepal length	sepal width	petal length	petal width	target
0	-0.900681	1.019004	-1.340227	-1.315444	0
1	-1.143017	-0.131979	-1.340227	-1.315444	0
2	-1.385353	0.328414	-1.397064	-1.315444	0
3	-1.506521	0.098217	-1.283389	-1.315444	0
4	-1.021849	1.249201	-1.340227	-1.315444	0
...	...	...	...	...	...
145	1.038005	-0.131979	0.819596	1.448832	2
146	0.553333	-1.282963	0.705921	0.922303	2
147	0.795669	-0.131979	0.819596	1.053935	2
148	0.432165	0.788808	0.933271	1.448832	2
149	0.068662	-0.131979	0.762758	0.790671	2

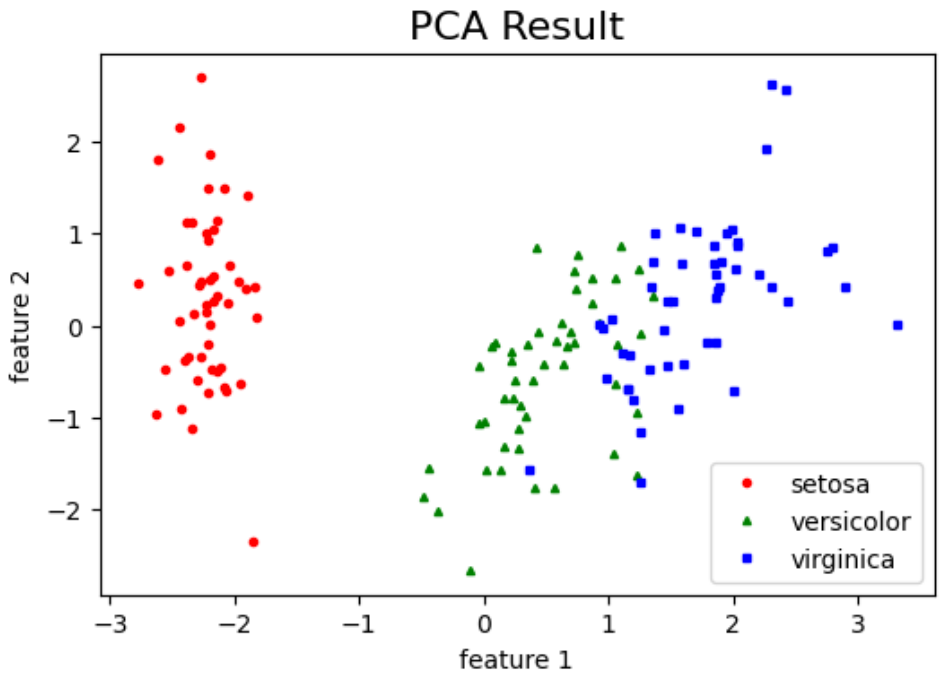
PCA

```
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import pandas as pd

iris = load_iris()
X = StandardScaler().fit_transform(iris.data)
y = iris.target

pca = PCA(n_components=2)
pca_X = pca.fit_transform(X)
df = pd.DataFrame(pca_X,
          columns=['feature 1', 'feature 2'])
df
```

	feature 1	feature 2
0	-2.264703	0.480027
1	-2.080961	-0.674134
2	-2.364229	-0.341908
3	-2.299384	-0.597395
4	-2.389842	0.646835
...	...	...
145	1.870503	0.386966
146	1.564580	-0.896687
147	1.521170	0.269069
148	1.372788	1.011254
149	0.960656	-0.024332

PCA결과 시각화

```
import matplotlib.pyplot as plt

plt.figure(dpi=100)
plt.title('PCA Result', fontsize=16)
plt.plot(pca_X[:,0][y==0], pca_X[:,1][y==0],
             "ro", markersize=3, label='setosa')
plt.plot(pca_X[:,0][y==1], pca_X[:,1][y==1],
             "g^", markersize=3, label='versicolor')
plt.plot(pca_X[:,0][y==2], pca_X[:,1][y==2],
             "bs", markersize=3, label='virginica')
plt.xlabel('feature 1')
plt.ylabel('feature 2')
plt.legend()
plt.show()
```



비지도 학습

비지도학습(Unsupervised Learning)

- 지도학습(supervised learning)
  - 부류 정보를 가진 샘플 (x,t)로 구성된 학습 집합 사용
    - x는 특징 벡터이고 t는 x가 속한 부류
- 비지도학습(Unsupervised Learning)
  - 지도 학습에서 사용했던 (x,t)중에 부류 정보 t가 없는 상황의 학습
  - 목적: 데이터들의 패턴과 구조 또는 타겟과의 관계를 파악
  - 주요 적용분야
    - 군집화(clustering)
    - 차원축소(dimentionality reduction) - PCA 등
    - 이상치 탐색(outlier detection)
    - 밀도추정(density estimation)

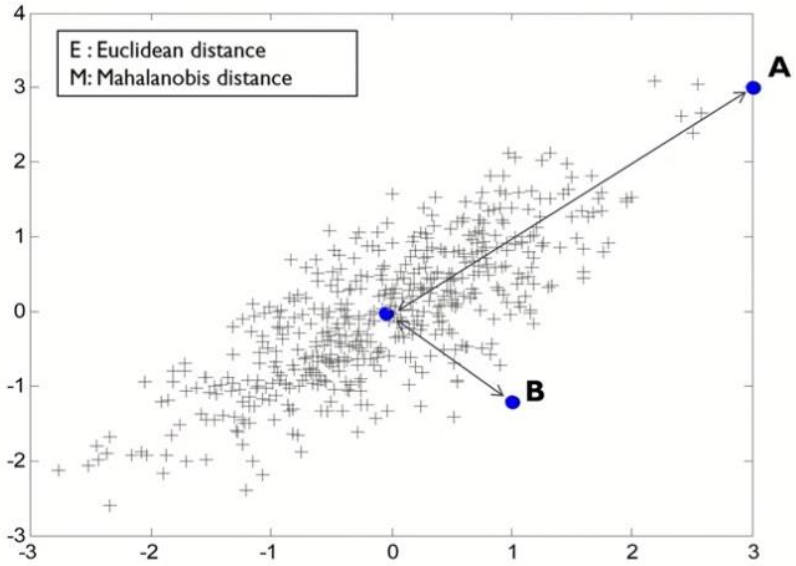
군집화(clustering)

- 유사한 속성들을 갖는 관측치들을 묶어 전체 데이터를 몇 개의 군집(cluster)으로 나누는 것
  - 군집(cluster)
    - 비슷한 특성을 가진 데이터들의 집합
  - 군집화의 원리
    - 군집내 응집도의 최대화 : 군집을 이루는 데이터들의 거리를 최소화
    - 군집간 분리도의 최대화 : 다른 군집 간의 거리를 최대화
- 군집화 수행 시 주요 고려사항
  - 어떤 거리 척도를 사용하여 유사도를 측정할 것인가?
  - 어떤 군집화 알고리즘을 사용할 것인가?
  - 어떻게 최적의 군집 개수를 결정할 것인가?
  - 어떻게 군집화 결과를 측정/평가할 것인가?

거리척도

유클리드 거리(Euclidian Distance)

$$d_E(x,y) = \left(\sum_{i=1}^{n}|x_i - y_i|^2\right)^{\frac{1}{2}} = \sqrt{\sum_{i=1}^{n}|x_i - y_i|^2}$$



맨하탄 거리(Manhattan Distance)

$$d_M(x,y) = \sum_{i=1}^{n}|x_i - y_i|$$

마할라노비스 거리(Mahalanobis Distance)

$$d_{Mahalanobis}(X,Y) = \sqrt{(X-Y)^T S^{-1}(X-Y)}$$

S^{-1}: covariance matrix

참고: 공분산과 상관계수

- **공분산(covariance)**
  - 각 확률변수의 분포를 나타내기 위한 것
  - $Cov(X,Y) = E\{(X-\mu)(Y-\nu)\}$
    - X의 편차와 Y의 편차를 곱한 것의 평균

$\therefore\ E(X) = \mu \quad E(Y) = \upsilon$

$$
\begin{aligned}
Cov(X,Y) &= E\big((X-\mu)(Y-\nu)\big) \\
&= E(XY - \mu Y - \nu X + \mu\nu) \\
&= E(XY) - \mu E(Y) - \nu E(X) + \mu\nu \\
&= E(XY) - \mu\nu
\end{aligned}
$$

- **상관계수(correlation)**
  - 확률변수의 절대크기에 영향을 받지 않도록 표준화한 것

$$\rho = \frac{Cov(X,Y)}{\sqrt{Var(X)Var(Y)}} \qquad -1 \le \rho \le 1$$

참고: 공분산 행렬(covariance matrix)

- **공분산들의 행렬**
  - 각 확률변수 사이의 공분산을 모두 구해서 행렬화

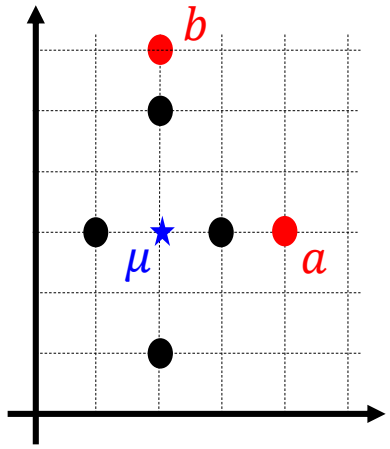
$$\begin{bmatrix} Cov(x_1,x_1) & Cov(x_1,x_2) & \cdots & Cov(x_1,x_m) \\ Cov(x_2,x_1) & Cov(x_2,x_2) & \cdots & Cov(x_2,x_m) \\ \vdots & & \ddots & \vdots \\ Cov(x_m,x_1) & Cov(x_3,x_2) & \cdots & Cov(x_m,x_m) \end{bmatrix}$$

$$Cov(x_i,x_i) = Var(x_i)$$

- **마할라노비스 거리 계산의 예** $d_M(X,Y) = \sqrt{(X-Y)^T S^{-1}(X-Y)}$
  - data: (2,1), (1,3), (2,5), (3,3)
  - $\mu = (2,\ 3)$
  - $S = \begin{bmatrix} 0.5 & 0 \\ 0 & 2 \end{bmatrix}$ $\quad S^{-1} = \begin{bmatrix} 2 & 0 \\ 0 & 0.5 \end{bmatrix}$

$$S = \begin{bmatrix} \frac{(2-2)^2+(1-2)^2+(2-2)^2+(3-2)^2}{4} & \frac{(2-6)+(3-6)+(10-6)+(9-6)}{4} \\ \frac{(2-6)+(3-6)+(10-6)+(9-6)}{4} & \frac{(1-3)^2+(3-3)^2+(5-3)^2+(3-3)^2}{4} \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 2 \end{bmatrix}$$

  - $a = (4,\ 3)$ $\quad b = (2,\ 6)$
  - $d(a,\mu) = \sqrt{(4-2 \quad 3-3)\begin{pmatrix} 2 & 0 \\ 0 & 0.5 \end{pmatrix}\begin{pmatrix} 4-2 \\ 3-3 \end{pmatrix}} = 2.8284$
  - $d(b,\mu) = \sqrt{(2-2 \quad 6-3)\begin{pmatrix} 2 & 0 \\ 0 & 0.5 \end{pmatrix}\begin{pmatrix} 2-2 \\ 6-3 \end{pmatrix}} = 2.1213$



군집화(clustering)

- 군집화의 원리
  - 군집내 응집도와 군집간 분리도를 최대화
- 군집화 수행 시 주요 고려사항
  - 거리 척도
    - Euclidian, Manhattan, Mahalanobis 등
  - 군집화 알고리즘
    - k-평균(k-means), 평균이동(mean-shift)
    - 가우스혼합모델(GMM:Gaussian Mixture Model)
    - DBSCAN 등
  - 최적의 군집 개수 결정
  - 군집화 결과 측정/평가

k-means 알고리즘

- k-means 알고리즘
  - 군집 중심점(centroid)라는 특정한 k개의 임의 지점을 선택해 해당 중심에 가장 가까운 포인트들을 선택하는 군집화기법
    - 중심점을 선택된 포인트들의 평균지점으로 이동
    - 이동된 중심점에서 다시 가까운 포인트를 선택
    - 다시 중심점을 평균지점으로 이동하는 프로세스를 반복적으로 수행

- 장단점
  - 이해하기 쉽고 간결함
  - 차원이 많을 수록 정확도가 낮고, 반복의 회수가 많을수록 느림
  - 초기에 몇 개의 군집을 선택할지 결정하기 어려움

Algorithm: k-means clustering

Input: a given data $\boldsymbol{X} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}\}$, the number of clusters $\boldsymbol{k}$, maximum number of iteration $\boldsymbol{I}$

Output: clustering results $\boldsymbol{r_{nk}}$ for all $\boldsymbol{n}$ and $\boldsymbol{k}$, centroid of clusters $\boldsymbol{C}$

Randomly initialize $\boldsymbol{C} = \{\boldsymbol{c_1}, \boldsymbol{c_2}, \ldots \boldsymbol{c_k}\}$

for $t = 1: I$ **do**

// Assignment step

for $n = 1: N$ **do**

$$r_{nk} = \begin{cases} 1, & if\ k = \underset{i}{\operatorname{argmin}} \parallel x_n - c_i \parallel^2 \\ 0, & otherwise \end{cases}$$

end

// Update step

for $k = 1: K$ **do**

$$c_k = \frac{1}{\sum_{n=1}^{N} r_{nk}} \sum_{n=1}^{N} r_{nk}\, x_n$$

end

end

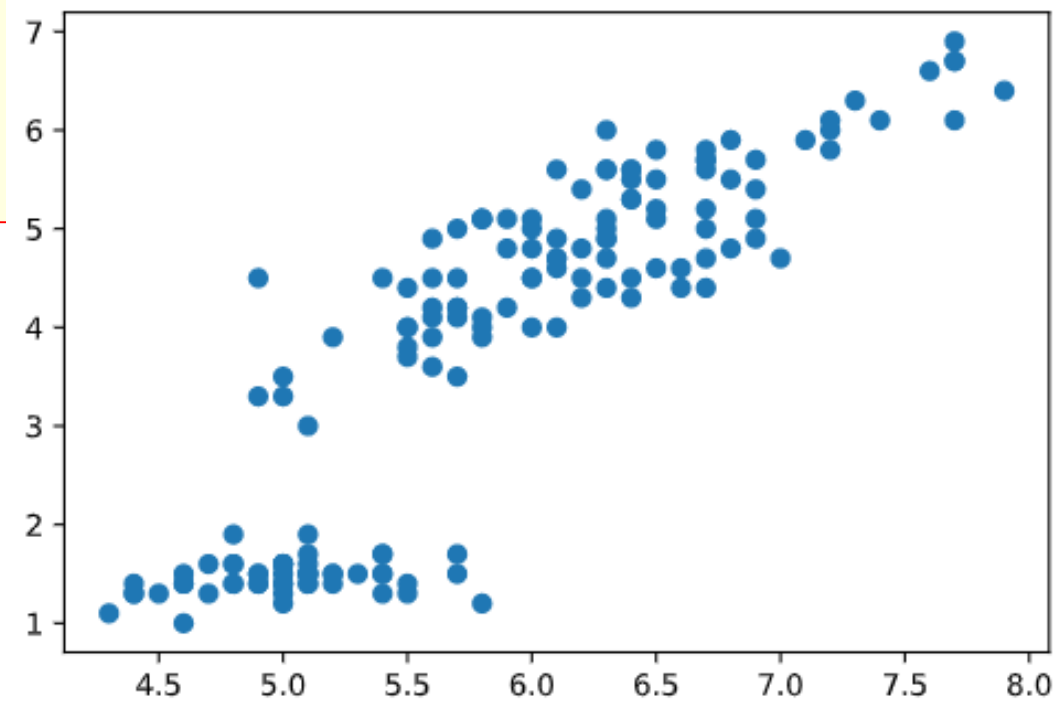
sklearn.cluster.**KMeans**

```
class KMeans(n_clusters=8, init='k-means++’,
             n_init=10, max_iter=300,
             random_state=None, algorithm='auto')
```

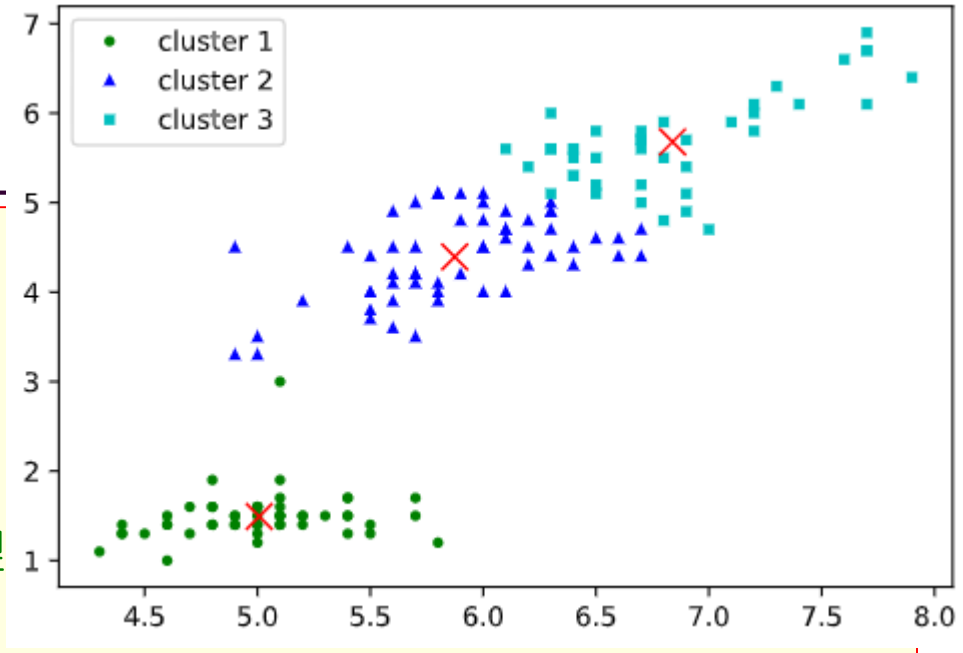
- **n_clusters: cluster의 개수**
- **init: {'k-means++', 'random’}**
  - k-means++ : 수렴속도를 높이기 위한 스마트한 방식
- **n_init: 다른 centroid로 시도할 회수**
- **algorithm: {“auto”, “full”, “elkan”}**
  - full: classical EM-style algorithm
    - EM(Expectation Maximization) 알고리즘

데이터 준비

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

iris = load_iris()
X = iris["data"][:, (0,2)] # 꽃받침 길이, 꽃잎 길이

plt.scatter(X[:,0], X[:,1])
plt.show()
```

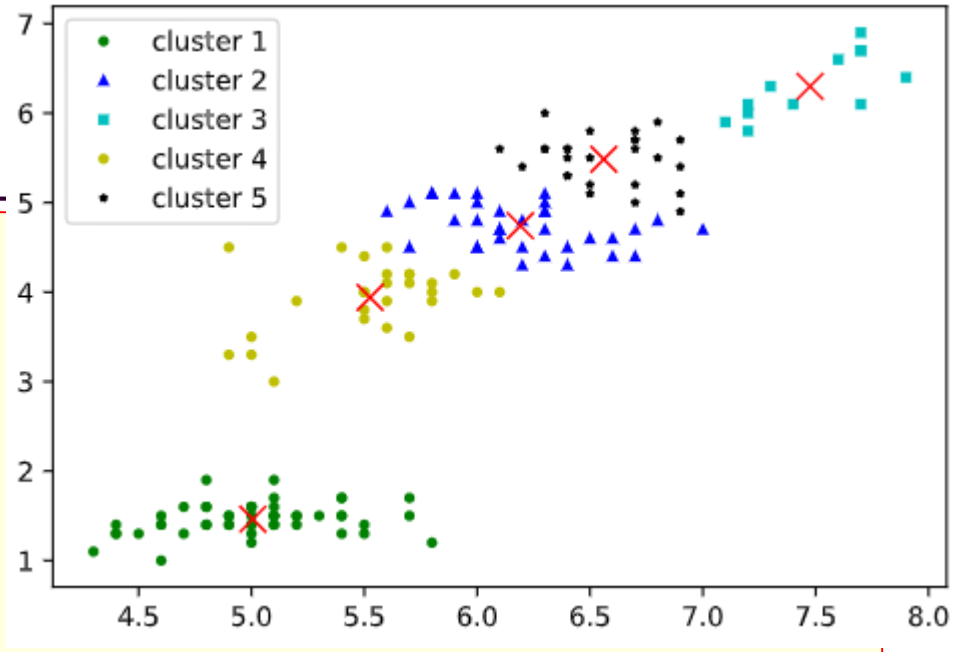


k=3인 경우




```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

iris = load_iris()
X = iris["data"][:, (0,2)] # 꽃받침 길이, 꽃잎
kmeans = KMeans(n_clusters=3)
y = kmeans.fit_predict(X)
C = kmeans.cluster_centers_

plt.plot(X[:,0][y==0], X[:,1][y==0], "go", label="cluster 1", markersize=3)
plt.plot(X[:,0][y==1], X[:,1][y==1], "b^", label="cluster 2", markersize=3)
plt.plot(X[:,0][y==2], X[:,1][y==2], "cs", label="cluster 3", markersize=3)
plt.plot(C[:,0], C[:,1], "rx", markersize=10)
plt.legend()
plt.show()
```

k=5인 경우




```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

iris = load_iris()
X = iris["data"][:, (0,2)] # 꽃받침 길이, 꽃잎
kmeans = KMeans(n_clusters=5)
y = kmeans.fit_predict(X)
C = kmeans.cluster_centers_

plt.plot(X[:,0][y==0], X[:,1][y==0], "go", label="cluster 1", markersize=3)
plt.plot(X[:,0][y==1], X[:,1][y==1], "b^", label="cluster 2", markersize=3)
plt.plot(X[:,0][y==2], X[:,1][y==2], "cs", label="cluster 3", markersize=3)
plt.plot(X[:,0][y==3], X[:,1][y==3], "yo", label="cluster 4", markersize=3)
plt.plot(X[:,0][y==4], X[:,1][y==4], "k*", label="cluster 5", markersize=3)
plt.plot(C[:,0], C[:,1], "rx", markersize=10)
plt.legend()
plt.show()
```

이너셔(inertia)

- **Centroid 초기화 방법**

```
class KMeans(n_clusters=8, init='k-means++’,
                 n_init=10, max_iter=300,
                 random_state=None, algorithm='auto')
```

  - Centroid 위치를 근사하게 알고 있는 경우
    - init : 초기화 리스트를 전달
  - 랜덤하게 여러 번 바꾸어 가면서 적용해 보고, 가장 좋은 솔루션을 선택
    - n_init : 초기화 회수 조절
    - 최적의 솔루션? : 이너셔(inertia)가 가장 낮은 것을 선택
      - 이너셔(inertia) : centroid와 sample들 사이의 거리의 제곱 합
      - intertia_ 변수로 확인 가능

실루엣 점수(silhouette score)

- 최적의 cluster 개수 찾기
  - Rule of Thumb
    - k : cluster의 개수
    - N : sample의 개수

$$k = \sqrt{\frac{N}{2}}$$

  - Silhouette Score(값이 1에 가까울수록 잘 나뉘어진 것)

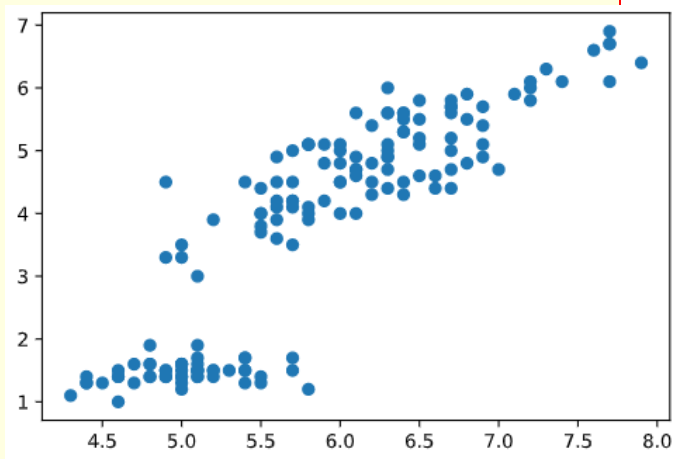
$$s = \frac{1}{N}\sum_{i=1}^{N}\frac{b_i - a_i}{\max(a_i, b_i)}, \qquad -1 \le s \le 1$$

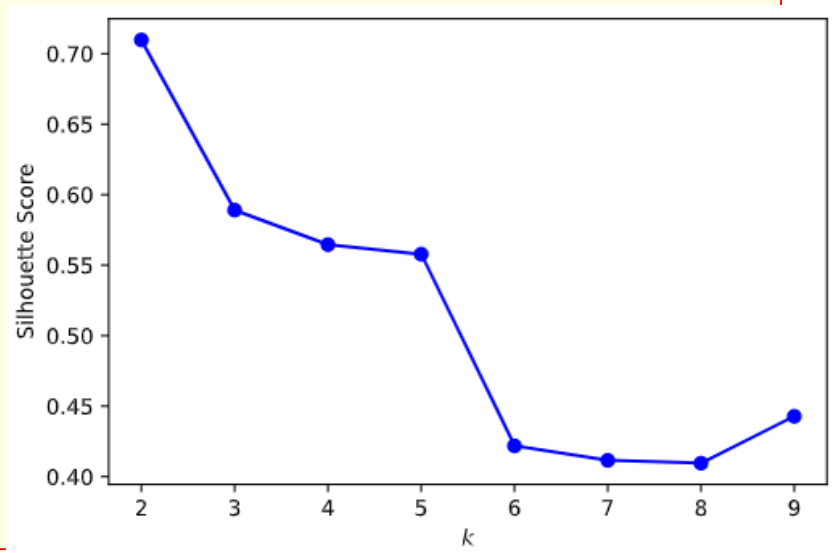
    - N : sample의 개수
    - a_i : i-번째 sample이 속한 cluster 내부의 평균거리
    - b_i : i-번째 sample과 가장 가까운 cluster sample들과의 평균거리

sklearn.metrics.**silhouette_score**

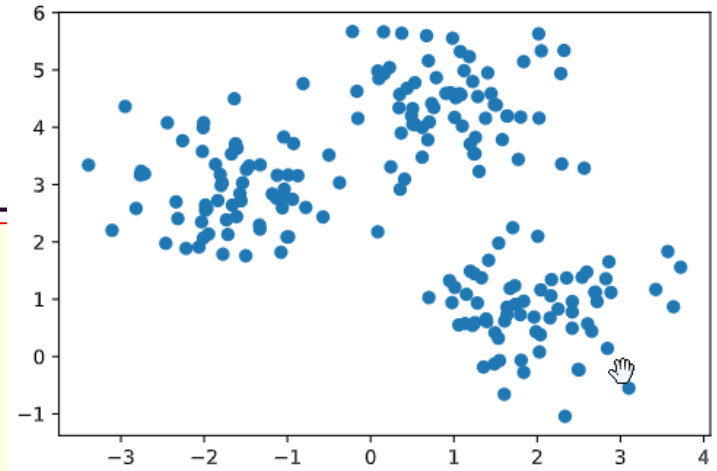
```
silhouette_score(X, labels, metric='euclidean')
```

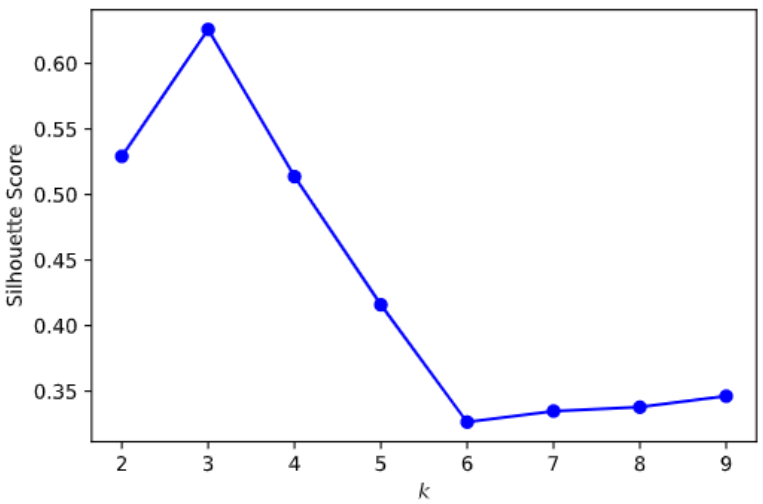
```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

iris = load_iris()
X = iris["data"][:, (0,2)] # 꽃받침 길이, 꽃잎 길이

kmeans_per_k = [KMeans(n_clusters=k).fit(X)
                for k in range(2, 10)]
s_scores = [silhouette_score(X, m.labels_)
                for m in kmeans_per_k]
plt.plot(range(2, 10), s_scores, "bo-")
plt.xlabel("$k$")
plt.ylabel("Silhouette Score")
plt.show()
```





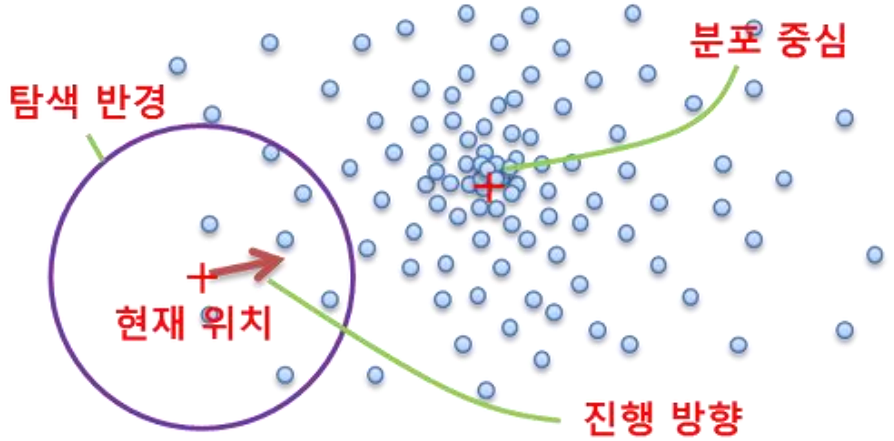
```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

X, y = make_blobs(n_samples=200, n_features=2, centers=3,
                  cluster_std=0.7, random_state=0)

kmeans_per_k = [KMeans(n_clusters=k).fit(X)
                for k in range(2, 10)]
s_scores = [silhouette_score(X, m.labels_)
                for m in kmeans_per_k]
plt.plot(range(2, 10), s_scores, "bo-")
plt.xlabel("$k$")
plt.ylabel("Silhouette Score")
plt.show()
```





Mean-Shift 알고리즘

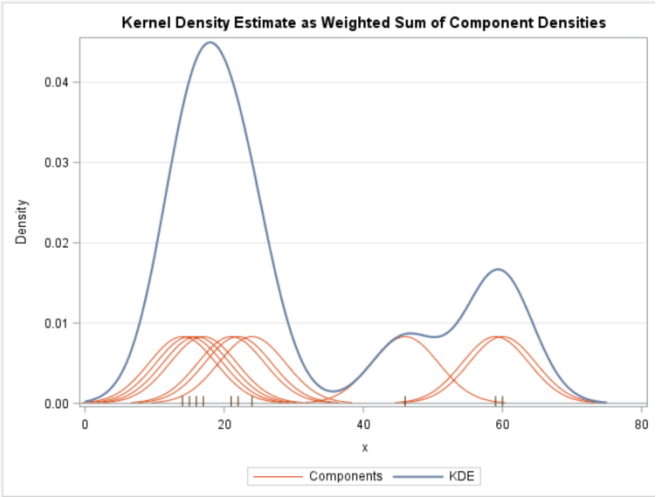
- 모든 sample 각각에 대해 주어진 bandwidth내에서 KDE(Kernel Density Estimation)를 이용하여 데이터 분포가 높은 곳으로 이동하면서 군집화를 수행하는 모델
  - Non-parametric모델 : 사전에 군집 개수를 지정하지 않으며 데이터 분포도에 기반해 자동으로 군집화 개수를 결정
- 장단점
  - 유연한 군집화 가능
  - 이상치의 영향력이 크지 않음
  - 미리 군집의 개수를 정할 필요가 없음
  - 알고리즘의 수행시간이 오래걸림
  - bandwidth의 크기에 따른 영향이 매우 큼



KDE(Kernel Density Estimation)

- 개별 데이터 포인트들에 커널함수(Gaussian 등)를 적용한 값들을 모두 합한 후 평균을 구하여 확률 밀도 함수를 추정하는 방식

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$



- $K()$: Kernel function
- h : bandwidth

- 탐색반경(bandwidth)
  - 너무 크면, 정확한 중심위치를 찾지 못하게 되며
  - 너무 작으면, local minimum에 빠지기 쉬움

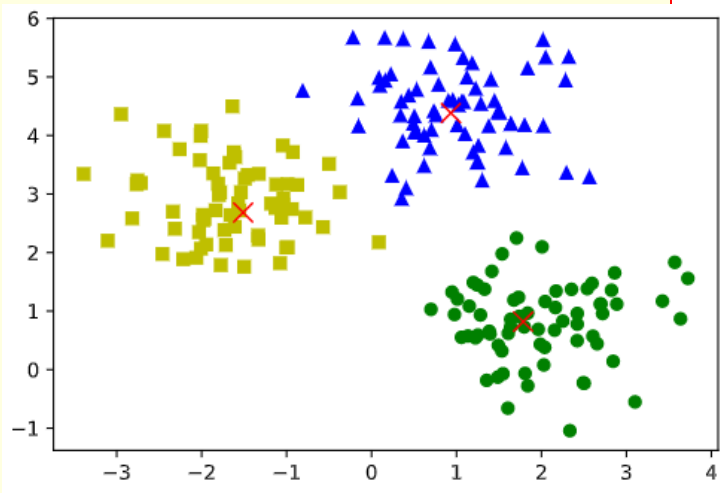
sklearn.cluster.**MeanShift**

```
class MeanShift(bandwidth=None,
                seeds=None,
                n_jobs=None,
                max_iter=300)
```

- **bandwidth: float value used in the RBF kernel**
- **seeds: used to initialize kernels**

```
cluster labels : [0 1 2]
[[ 1.78436594  0.82059933]
 [ 0.93390739  4.38280396]
 [-1.51114621  2.68467011]]
```

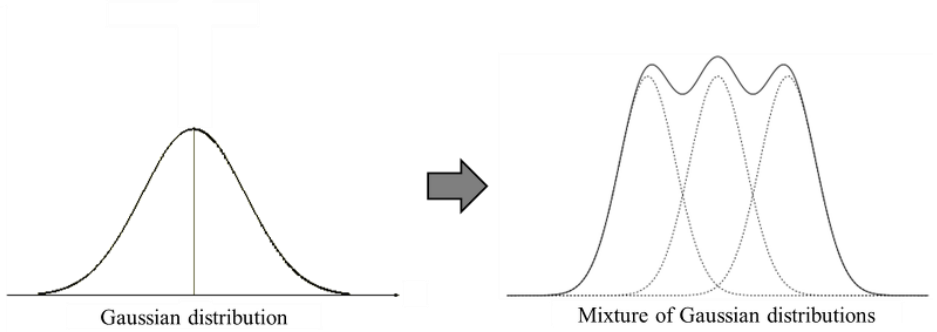
```python
from sklearn.datasets import make_blobs
from sklearn.cluster import MeanShift
import numpy as np
import matplotlib.pyplot as plt

X, y = make_blobs(n_samples=200, n_features=2, centers=3,
                  cluster_std=0.7, random_state=0)

mean_shift = MeanShift(bandwidth=1.0)
labels = mean_shift.fit_predict(X)

print('cluster labels :', np.unique(labels))
C = mean_shift.cluster_centers_
print(C)

plt.plot(X[:,0][labels==0], X[:,1][labels==0], "go")
plt.plot(X[:,0][labels==1], X[:,1][labels==1], "b^")
plt.plot(X[:,0][labels==2], X[:,1][labels==2], "ys")
plt.plot(C[:,0], C[:,1], "rx", markersize=10)
plt.show()
```



GMM(Gaussian Mixture Model)

- 샘플들은 파라메터가 알려지지 않은 여러 개의 혼합된 가우시안 분포에서 생성되었다고 가정하는 확률모델
  - 복잡한 확률분포를 K개의 가우시안(Gaussian) 분포를 혼합하여 표현

$$p(x) = \sum_{k=1}^{K} \pi_k \, N(x|\mu_k, \Sigma_k)$$



  - K: 확률분포의 개수
  - $N(x|\mu_k, \Sigma_k)$: 가우시안 확률밀도 함수
  - $\boldsymbol{\pi_k}$: k번째 가우시안 분포가 선택될 확률
  - $\boldsymbol{\mu_k}$: k번째 가우시안 분포의 평균
  - $\boldsymbol{\Sigma_k}$: k번째 가우시안 분포의 공분산

GMM을 이용한 분류

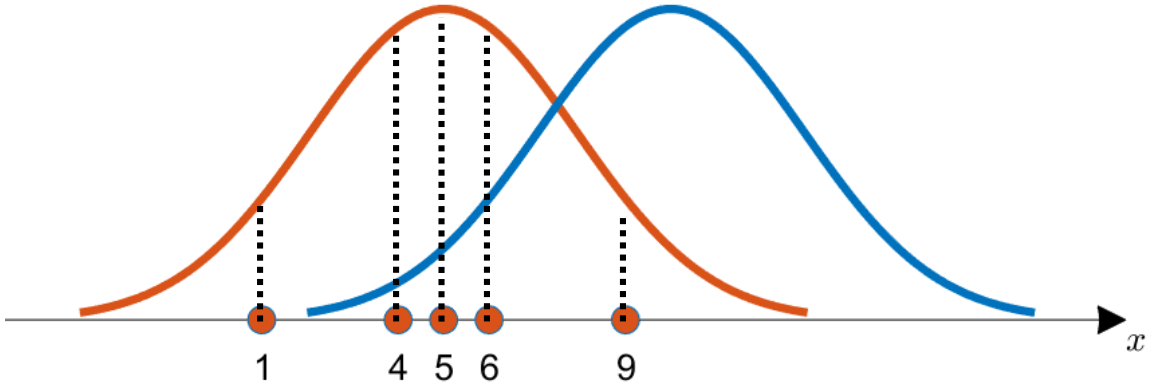
- 주어진 데이터 x_n에 대해, k개의 $\gamma(z_{nk})$를 계산하여 가장 값이 높은 가우시안 분포로 분류
- 신뢰도(Responsibility) $\gamma(z_{nk}) = P(z_{nk} = 1 \mid x_n), \quad z_{nk} \in \{0, 1\}$
  - z_{nk} 는 GMM의 k번째 분포가 선택되면 1, 아니면 0인 이진값

$$\gamma(z_{nk}) = \frac{P(z_{nk} = 1)P(x_n|z_{nk} = 1)}{\sum_{j=1}^{K} P(z_{nj} = 1)P(x_n|z_{nj} = 1)} = \frac{\pi_k \, N(x|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \, N(x|\mu_j, \Sigma_j)}$$

- 주어진 데이터 $X = \{x_1, x_2, \ldots, x_N\}$에 대해 EM알고리즘으로 GMM의 파라메터 $\boldsymbol{\pi}$, $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$를 추정
  - EM(Expactation Maximization) Algorithm
    - 최대우도법(Maximum Likelihood Estimation)을 이용하여 모수(파라메터)를 추정하는 과정

최대우도법(Maximum Likelihood Estimation)

- 파라메터 $\theta = \{\theta_1, \theta_2, \dots \theta_m\}$으로 구성된 **어떤 확률밀도함수 $P(x|\theta)$에서 관측된 표본 데이터집합을 $x = (x_1, x_2, \dots x_n)$이라 할 때, 이 표본들로 부터 최대 가능도(likelihood)를 갖는 분포를 찾아내고 그 파라메터 $\theta = \{\theta_1, \theta_2, \dots \theta_m\}$를 추정하는 기법**
- 가능도(likelihood)
  - 주어진 데이터가 해당 분포로부터 나왔을 가능성
  - 어떤 분포에 대한 각 데이터 샘플들의 높이(likelihood 기여도)를 계산해서 모두 곱한 것



sklearn.mixture.**GaussianMixture**

```
class GaussianMixture(n_components=1, tol=0.001,
                      max_iter=100, n_init=1,
                      random_state=None)
```

- **n_components: 군집(mixture component)의 개수**
- **tol: 수렴을 위한 임계치**
- **max_iter: EM Iteration의 수**
- **n_init: 시도할 초기화의 개수**

```
[[10.  2.]
 [ 1.  2.]]
[[[1.00000000e-06 1.20958672e-29]
  [1.20958672e-29 2.66666767e+00]]

 [[1.00000000e-06 1.25724707e-30]
  [1.25724707e-30 2.66666767e+00]]]
```

```
import numpy as np
from sklearn.mixture import GaussianMixture
X = np.array([[1, 2], [1, 4], [1, 0], [10, 2], [10, 4], [10, 0]])
gm = GaussianMixture(n_components=2, random_state=0).fit(X)
print(gm.means_)
print(gm.covariances_)
```

```
from sklearn.datasets import make_blobs
from sklearn.mixture import GaussianMixture
import numpy as np
import matplotlib.pyplot as plt

X, y = make_blobs(n_samples=200, n_features=2, centers=3,
                  cluster_std=0.7, random_state=0)

gmm = GaussianMixture(n_components=3)
labels = gmm.fit_predict(X)

print('cluster labels :', np.unique(labels))
C = gmm.means_
print(C)

plt.plot(X[:,0][labels==0], X[:,1][labels==0], "go")
plt.plot(X[:,0][labels==1], X[:,1][labels==1], "b^")
plt.plot(X[:,0][labels==2], X[:,1][labels==2], "ys")
plt.plot(C[:,0], C[:,1], "rx", markersize=10)
plt.show()
```

```
cluster labels : [0 1 2]
[[ 0.95995969  4.43266216]
 [ 1.98952065  0.80683533]
 [-1.65934006  2.89402567]]
```