머신러닝

[HW02 – EM ALGORITHM USING KMEANS FOR GMM]

담당교수 : 박철수 교수님

학　　번 : 2021202058

성　　명 : 송채영

[Introduction]

이 과제의 제목은 EM ALGORITHM USING KMEANS FOR이며, 과제의 목적은 Expectation Maximization (EM) 알고리즘을 사용해 Iris dataset에 대한 clustering을 수행하는 코드를 구현하는 것이다. Initialization, Multivariate gaussian distribution 계산, expectation, maximization, 모델 fitting, plotting 과정을 구현한다. 또한 결과를 plot하여 clustering 결과를 시각화해 K Means clustering과의 성능을 비교해본다.

[Assignment]

다음은 소스코드에 관한 설명이다.

```
C:\anaconda3\envs\scy\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning: When grouping with a lengt
h-1 list-like, you will need to pass a length-1 tuple to get_group in a future version of pandas. Pass `(na
me,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
C:\anaconda3\envs\scy\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning: When grouping with a lengt
h-1 list-like, you will need to pass a length-1 tuple to get_group in a future version of pandas. Pass `(na
me,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
C:\anaconda3\envs\scy\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is de
precated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\anaconda3\envs\scy\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning: When grouping with a lengt
h-1 list-like, you will need to pass a length-1 tuple to get_group in a future version of pandas. Pass `(na
me,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
```

```python
import copy
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.cluster import KMeans
import warnings
warnings.filterwarnings("ignore") # To use eliminate seaborn warnings
```

실행 시 seaborn으로 인한 warning이 생겨 이를 결과화면에서 없애기 위해 warnings를 import하여 무시해주었다.

```python
class EM:
    def __init__(self, n_clusters, iteration):

        """
        Expectation-Maximization (EM) clustering algorithm

        Parameter
        - n_clusters : Number of clusters to partition the data
        - iteration : Number of iterations for EM alrgorithm

        Attributes
        - n_clusters : Number of clusters
        - iteration : Number of iterations
        - mean : Mean vectors for each cluster
        - sigma : Covariance matrix for each cluster
        - pi : Prior probability that each cluster

        """

        self.n_clusters = n_clusters # Initialize number of clusters attribute
        self.iteration = iteration # Initialize number of iterations attribute
        self.mean = np.zeros((3, 4)) # Initialize mean vectors
        self.sigma = np.zeros((3, 4, 4)) # Initialize covariance matrices
        self.pi = np.zeros((3)) # Initialize prior probablility
```

class EM은 EM 알고리즘을 구현한 class로 n_cluster, cluster의 개수와 iteration, 반복횟수를 인자로 받으며 클래스의 멤버 변수로 저장되어 있다. 파라미터는 mean, sigma, pi이며 각각 cluster의 평균, 공분산, 사전 확률에 해당한다. EM class는 init 메서드, initialization 메서드, multivariate gaussian distribution 메서드, expectation 메서드, maximization 메서드, fit 메서드의 멤버 함수를 가지고 있다.

```python
def initialization(self, data):

    """
    Initialize cluster parameters(mean, sigma, and pi)

    Parameter
    - data : Input data matrix
    """

    # Randomly select data points as initial mean vectors for each cluster
    self.mean = data[np.random.choice(data.shape[0], size = self.n_clusters, replace=False)]
    # Initialize covariance matrices for each cluster as identity matrices
    self.sigma = np.array([np.eye(4)] * self.n_clusters)
    # Initialize prior probabilities for each cluster uniformly
    self.pi = np.ones(self.n_clusters) / self.n_clusters
```

initialization 메서드는 EM 알고리즘의 초기화를 수행하며, cluster의 mean, sigam, pi를 초기화한다. 먼저 평균은 주어진 데이터에서 무작위로 클러스터의 평균을 선택하였다. 이때 np.random.choice를 사용해 데이터에서 중복되지 않는 무작위 인덱스를 선택하고 선택된 인덱스를 사용하여 초기 클러스터의 평균을 선택하도록 구현하였다. 분산의 경우 np.eye를 사용해 각 cluster의 공분산 matrix를 단위 행렬로 초기화하였다. pi의 경우 각 cluster의 prior probabilitie를 균일하게 초기화해 주었다. 여기선 $\frac{1}{3}$에 해당한다.

```python
def multivariate_gaussian_distribution(self, data, mean, sigma):

    """
    Calculate multivariate Gaussian probability density function
    Computes the probability density function (PDF) value for a given data point with respect
    to a multivariate Gaussian distribution defined by the mean vector and covariance matrix

    Parameter
    - data : Input data point
    - mean : Mean vector of the distribution
    - sigma : Covariance matrix of the distribution
    """

    # Calculate the difference between the data point and the mean vector
    diff = data - mean
    # Calculate the determinant of the covariance matrix
    det_sigma = np.linalg.det(sigma)
    # Calculate the inverse of the covariance matrix
    inv_sigma = np.linalg.inv(sigma)
    # Calculate the constant term in the Gaussian distribution formula
    constant = 1 / ((2 * np.pi) ** (4 / 2) * det_sigma ** 0.5)
    # Calculate the exponent term in the Gaussian distribution formul
    exponent = -0.5 * np.dot(diff, np.dot(inv_sigma, diff.T))
    # Calculate the probability density function value
    return constant * np.exp(exponent)
```

다음으로 multivariate_gaussian_distribution 메서드는 다변수 가우시안 확률 밀도 함수를 계산하며, 각 데이터 포인트가 주어진 cluster에 속할 확률을 계산하고 이를 EM 알고리즘의 Expectation에서 사용한다. 위 코드에서 diff 변수는 데이터 포인트와 평균 벡터 간의 차이를 계산하고, det_sigma는 공분산 행렬의 행렬식을, inv_sigma는 공분산 행렬의 역행렬을 계산한다. 제안서의 수식을 기반으로 코드로 구현하였으며 이에 해당하는 수식은 다음과 같다.

$$g_{(\mu,\Sigma)}(x) = \frac{1}{\sqrt{2\pi}^{d}\sqrt{\det(\Sigma)}} e^{-\frac{1}{2}(x-\mu)^{T}\Sigma^{-1}(x-\mu)}$$

```python
def expectation(self, data):

    """
    Expectation step of the EM algorithm
    Computes the posterior probabilities of each data point
    belonging to each cluster in the EM algorithm

    Parameter
    - data : Input data matrix
    """

    # Initialize an array to store posterior probabilities
    posterior = np.zeros((data.shape[0], self.n_clusters))

    for n in range(data.shape[0]): # Loop through each data point
        total_likelihood = 0
        for k in range(self.n_clusters): # Loop through each cluster
            # Get parameters for the current cluster
            pi = self.pi[k]
            mean = self.mean[k]
            sigma = self.sigma[k]

            # Calculate the likelihood of the data point belonging to the current cluster
            likelihood = self.multivariate_gaussian_distribution(data[n], mean, sigma)
            # Calculate the unnormalized posterior probability
            temp = pi * likelihood
            total_likelihood += temp
            posterior[n][k] = temp
        posterior[n] /= total_likelihood  # Normalize the posterior probabilities for the current data point
    return posterior
```

다음으로 expectation 메서드는 EM 알고리즘의 Expectations step에 해당하는 부분으로 주어진 데이터 포인트가 각 cluster에 속할 확률을 계산한다. 각 데이터 포인터에 대한 Posterior Probability는 다음과 같은 과정으로 구하였다. total_likelihood 변수에 현재 데이터 포인트의 모든 cluster에 대한 모든 likelihood의 합을 저장하였고 각 cluster에 대해 현재 cluster의 pi, mean, sigma를 가져오고 위에서 구현한 다변수 가우시안 확률 밀도 함수를 사용하여 현재 데이터 포인트가 현재 cluster에 속할 확률을 계산하였다. 계산된 확률과 cluster의 prior probability를 곱하여 normalization된 posterior probability를 계산하고 이를 total likelihood에 더해 posterior 배열에 저장하였다. 이를 수식으로 나타내면 다음과 같다. $P(z_{nk} = 1|x_n) = \frac{\pi_k \cdot N(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \cdot N(x_n|\mu_j|\Sigma_j)}$

```python
def maximization(self, data, posterior):

    """
    Maximization step of the EM algorithm
    Updates cluster parameters based on data and posterior probabilities

    Parameter
    - data : Input data matrix
    - posterior : Posterior probabilities of data point belonging to each cluster
    """

    for k in range(self.n_clusters): # Loop through each cluster

        # Update mean for the current cluster
        mean_sum = np.zeros_like(self.mean[k])
        for i in range(data.shape[0]):
            mean_sum += posterior[i, k] * data[i] # Accumulate the weighted sum of data points
        # Update mean by dividing the sum by the total posterior probability for the cluster
        self.mean[k] = mean_sum / np.sum(posterior[:, k])

        # Update covariance matrix for the current cluster
        sigma_sum = np.zeros_like(self.sigma[k])
        for i in range(data.shape[0]):
            diff = data[i] - self.mean[k] # Compute the difference between the data point and the mean
            sigma_sum += posterior[i, k] * np.outer(diff, diff) # Accumulate the weighted sum of outer products
        # Update covariance matrix by dividing the sum by the total posterior probability for the cluster
        self.sigma[k] = sigma_sum / np.sum(posterior[:, k])

        # Update the prior probability for the current cluster
        self.pi[k] = np.mean(posterior[:, k])
```

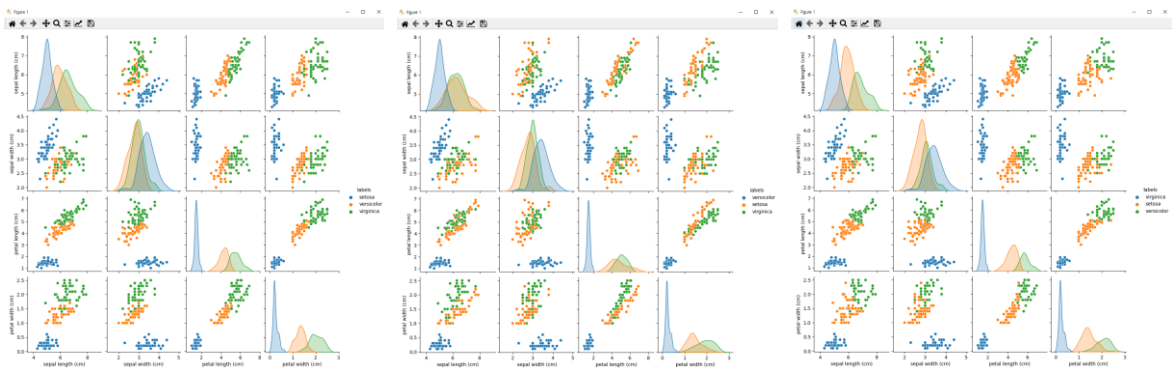
다음으로 maximization 메서드는 EM 알고리즘의 Maximization step에 해당하는 부분으로 cluster의 평균, 공분산 행렬, 사전 확률이 업데이트된다. 평균, 공분산, 사전 확률로 나누어서 구현하였으며 각각은 다음과 같다. 우선 평균은 현재 cluster에 속하는 각 데이터 포인트의 가중 평균을 계산하여 cluster의 새로운 평균을 구하기 위해 데이터 포인트와 해당 포인트의 posterior probability를 곱하여 가중치를 주고 이를 모두 더하고 총 posterior probability로 나누었다. 이어서 공분산 행렬은 각 데이터 포인트와 해당 포인트의 후방 확률을 곱하여 외적 계산을 하여 cluster의 새로운 공분산 행렬을 계산한다. 이후 그것들을 모두 더하여 총 posterior probability로 나누었다. 마지막으로 사전 확률은 해당 cluster에 속하는 데이터 포인트의 평균 posterior probability로 구하였다. 이를 수식으로 나타내면 다음과 같다. 차례대로, 평균, 공분산 행렬, 사전 확률을 update하는 식에 해당한다. $\mu_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) \cdot x_n}{\sum_{n=1}^{N} \gamma(z_{nk})}$, $\sum_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk}) \cdot (x_n - \mu_k)((x_n - \mu_k)^T}{\sum_{n=1}^{N} \gamma(z_{nk})}$, $\pi_k = \frac{1}{N} \sum_{n=1}^{N} \gamma(z_{nk})$

```
def fit(self, data):

    """
    Fit the EM model to the input data
    Fits the EM model to the given input data by initializing cluster parameters,
    performing iterations of the expectation-maximization algorithm, and returning
    the predicted cluster assignments for each data point

    Parameter
    - data : Input data matrix
    """

    self.initialization(data = data) # Initialize cluster parameters

    for i in range(self.iteration): # Perform iterations of the expectation-maximization algorithm
        posterior = self.expectation(data = data)
        self.maximization(data = data, posterior = posterior)

    # Format predictions by selecting the cluster with the highest posterior probabilit
    prediction = np.argmax(posterior, axis=1)
    return prediction
```

다음으로 fit 메서드는 EM 모델을 입력 데이터에 맞추는 부분이다. EM 알고리즘을 반복해서 실행하며 cluster 파라미터를 초기화, update를 하고 각 데이터 포인트에 대한 예측된 cluster를 반환한다. EM 알고리즘은 Expectation step과 Maximization step을 반복하며 수행하며 위에서 구현한 함수를 호출하여 구현하였다. EM 알고리즘을 실행한 후 각 데이터 포인트에 대한 cluster 할당을 결정하기 위해 posterior probability가 가장 높은 cluster를 선택해 해당 데이터 포인트를 반환하도록 하였다.

```
def plotting(data, labels):

    """
    Plot pairwise relationships in a dataset
    Visualizes pairwise relationships between features
    in the input data matrix using a seaborn pairplo

    Parameter
    - data : Input data matrix
    - labels : Labels for each data point
    """
    # Create a pairplot using seaborn and assigning colors based on the provided labels
    sns.pairplot(pd.DataFrame(data=data, columns=iris['feature_names']).assign(labels=labels), hue='labels')
    plt.show()
```

plotting 메서드는 입력 데이터 행렬과 해당 데이터 포인트에 대한 label을 사용하여 데이터 set을 쌍 관계를 시각화하는 부분이다. 조건에 맞게 sns.pairplot을 사용하였다.



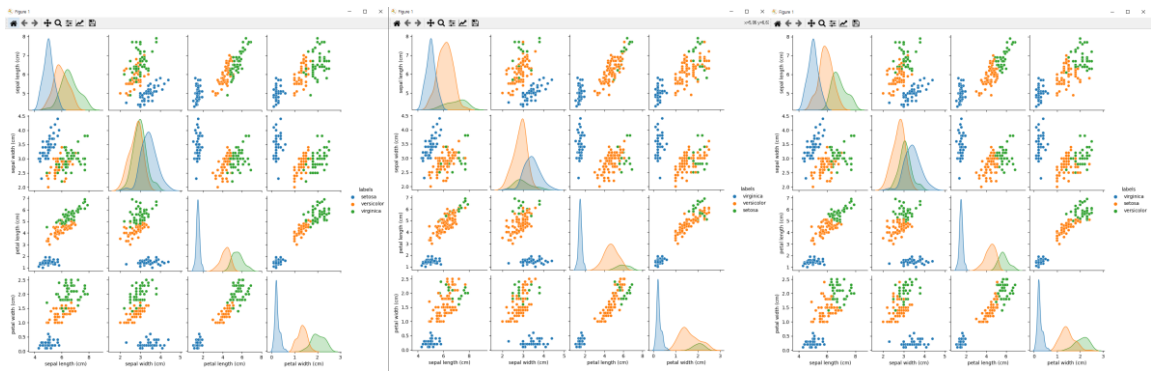
왼쪽부터 original, EM, KMeans에 해당한다. setosa, versicolor, virignica 이 3가지 종의 데이터 셋이 있고 꽃 각각은 sepal_length, sepal_width, petal_length, petal_width, 4가지 변수를 비교한다. pairplot 그래프를 사용하여 데이터의 일치도를 확인할 수 있었다.

```
if __name__ == '__main__':
    # Loading and labeling data
    iris = datasets.load_iris()
    original_data = pd.DataFrame(data=np.c_[iris['data'], iris['target']], columns=iris['feature_names'] + ['labels'])
    original_data['labels'] = original_data['labels'].map({0: 'setosa', 1: 'versicolor', 2: 'virginica'})
    plotting(original_data.drop(columns=['labels']), original_data['labels']) # Visualize the original data

    data = iris['data']

    # Unsupervised learning(clustering) using EM algorithm
    EM_model = EM(n_clusters=3, iteration=iteration)
    EM_pred = EM_model.fit(data)
    EM_pd = pd.DataFrame(data=np.c_[data, EM_pred], columns=iris['feature_names'] + ['labels'])
    EM_pd['labels'] = EM_pd['labels'].map({0:'setosa', 1:'versicolor', 2:'virginica'})
    plotting(EM_pd.drop(columns=['labels']), EM_pd['labels']) # Visualize clustering results using EM algorithm

    # Print EM algorithm results
    print(f'pi :            {EM_model.pi}')
    print(f'count / total : {np.bincount(EM_pred) / 150}')

    # Unsupervised learning(clustering) using KMeans algorithm
    KM_model = KMeans(n_clusters=3, init='random', random_state=seed_num, max_iter=iteration).fit(data)
    KM_pred = KM_model.predict(data)
    KM_pd = pd.DataFrame(data=np.c_[data, KM_pred], columns=iris['feature_names'] + ['labels'])
    KM_pd['labels'] = KM_pd['labels'].map({0:'setosa', 1:'versicolor', 2:'virginica'})
    plotting(KM_pd.drop(columns=['labels']), KM_pd['labels'] )# Visualize clustering results using KMeans algorithm

    # Evaluate and print clustering performance
    for idx in range(2):
        EM_point = np.argmax(np.bincount(EM_pred[idx*50:(idx+1)*50]))
        KM_point = np.argmax(np.bincount(KM_pred[idx*50:(idx+1)*50]))
        EM_pred = np.where(EM_pred == idx, 3, EM_pred)
        EM_pred = np.where(EM_pred == EM_point, idx, EM_pred)
        EM_pred = np.where(EM_pred == 3, EM_point, EM_pred)
        KM_pred = np.where(KM_pred == idx, 3, KM_pred)
        KM_pred = np.where(KM_pred == KM_point, idx, KM_pred)
        KM_pred = np.where(KM_pred == 3, KM_point, KM_pred)

    EM_hit = np.sum(iris['target']==EM_pred)
    KM_hit = np.sum(iris['target']==KM_pred)
    print(f'EM Accuracy: {round(EM_hit / 150,2)}    Hit: {EM_hit} / 150')
    print(f'KM Accuracy: {round(KM_hit / 150,2)}    Hit: {KM_hit} / 150')
```

main 부분이다. 데이터 로드 및 labeling, EM 알고리즘을 사용한 clustering, KMeans 알고리즘을 사용한 clustering, clustering 성능 평가 부분이 구현되어 있다.

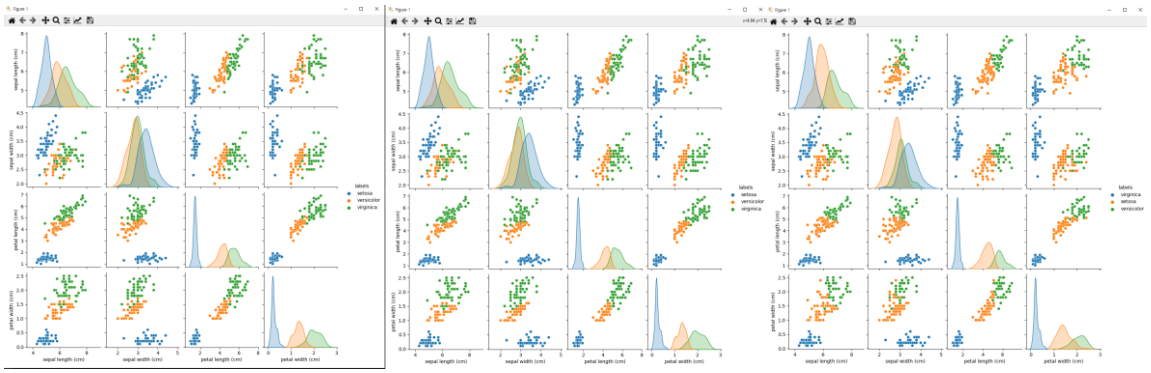
[Result]

①



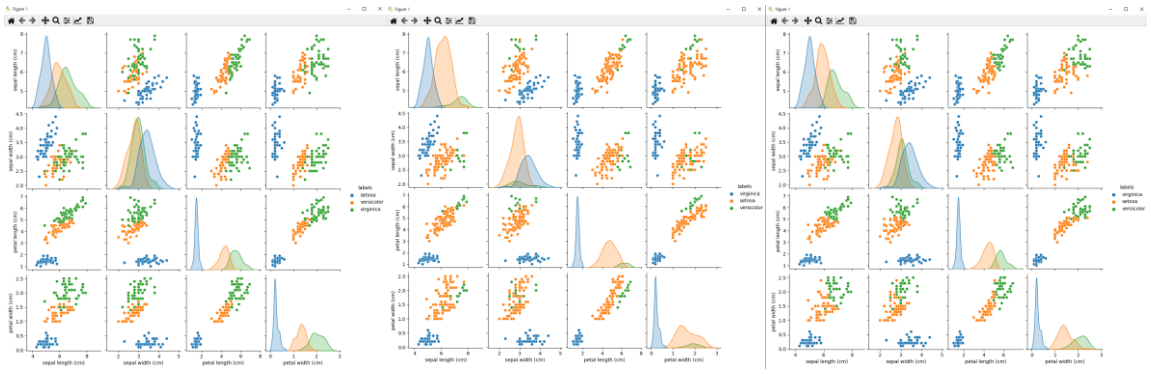
```
PS C:\Users\송채영\Desktop\code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/code/CYS.py
pi :            [0.12149265 0.5451741  0.33333326]
count / total : [0.10666667 0.56       0.33333333]
EM Accuracy: 0.77    Hit: 116 / 150
KM Accuracy: 0.89    Hit: 134 / 150
PS C:\Users\송채영\Desktop\code>
```

②



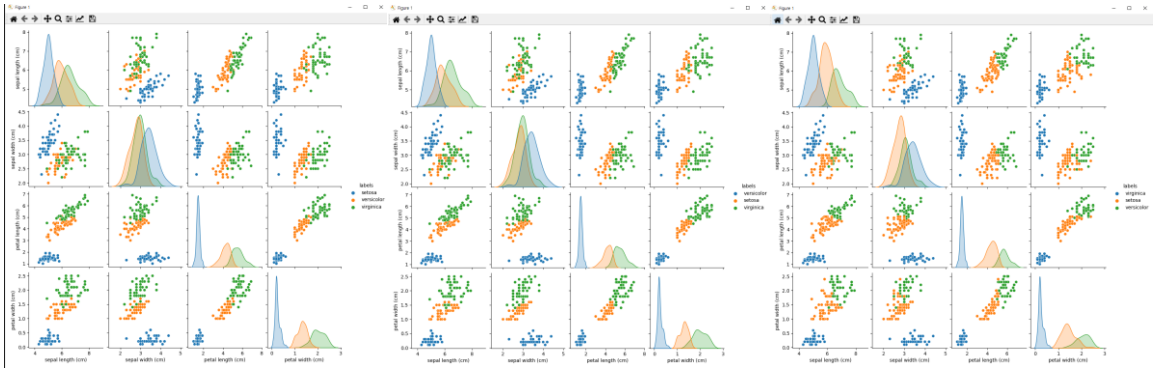
```
PS C:\Users\송채영\Desktop\code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/code/CYS.py
pi :            [0.33333333 0.29919319 0.36747348]
count / total : [0.33333333 0.3        0.36666667]
EM Accuracy: 0.97    Hit: 145 / 150
KM Accuracy: 0.89    Hit: 134 / 150
PS C:\Users\송채영\Desktop\code>
```

③



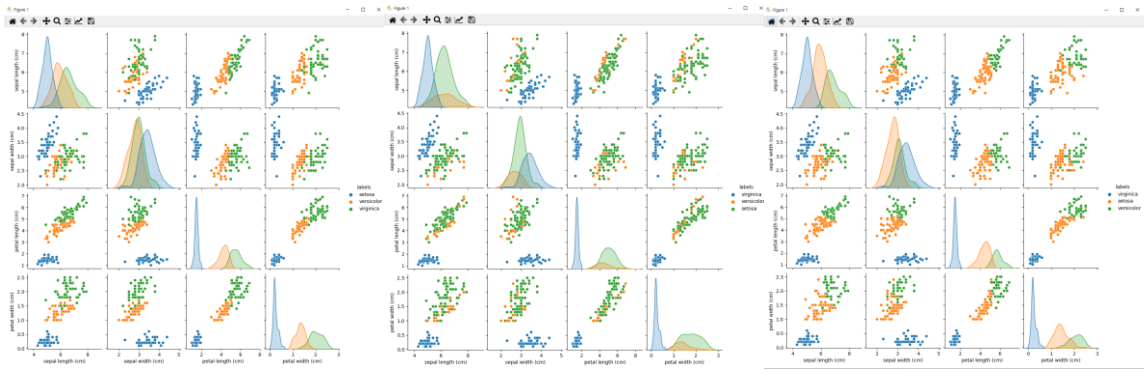
```
PS C:\Users\송채영\Desktop\code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/code/CYS.py
pi :            [0.59452813 0.07213855 0.33333332]
count / total : [0.59333333 0.07333333 0.33333333]
EM Accuracy: 0.74    Hit: 111 / 150
KM Accuracy: 0.89    Hit: 134 / 150
PS C:\Users\송채영\Desktop\code>
```

④



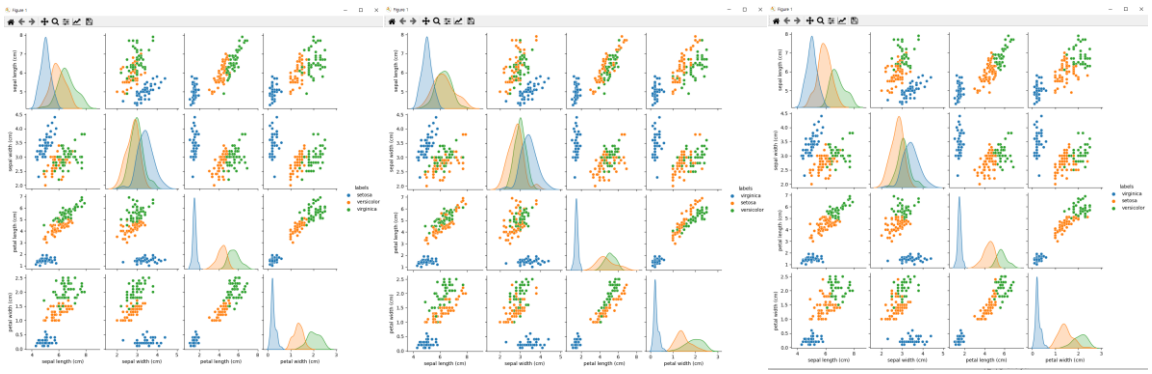
```
PS C:\Users\송채영\Desktop\code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/code/CYS.py
pi :            [0.29919319 0.33333333 0.36747348]
count / total : [0.3        0.33333333 0.36666667]
EM Accuracy: 0.97    Hit: 145 / 150
KM Accuracy: 0.89    Hit: 134 / 150
PS C:\Users\송채영\Desktop\code>
```

⑤



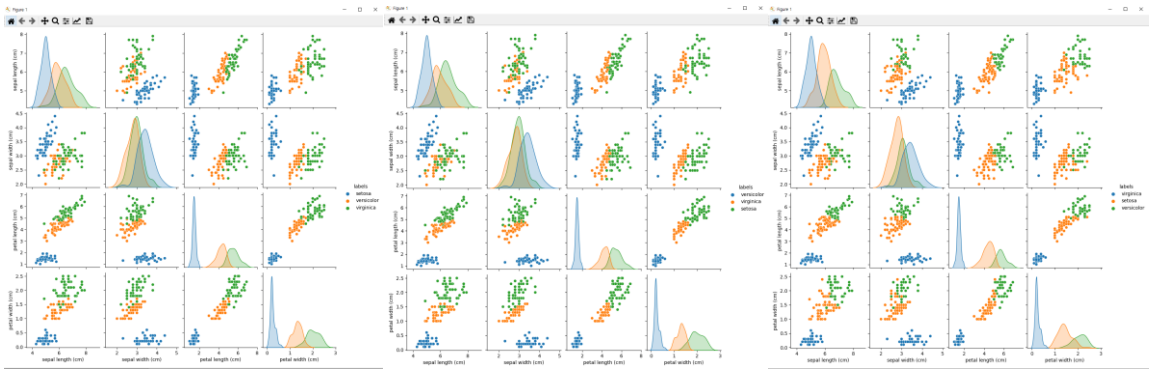
```
PS C:\Users\송채영\Desktop\code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/code/CYS.py
pi :            [0.52709603 0.13957738 0.33332659]
count / total : [0.50666667 0.16       0.33333333]
EM Accuracy: 0.57    Hit: 86 / 150
KM Accuracy: 0.89    Hit: 134 / 150
PS C:\Users\송채영\Desktop\code>
```

⑥



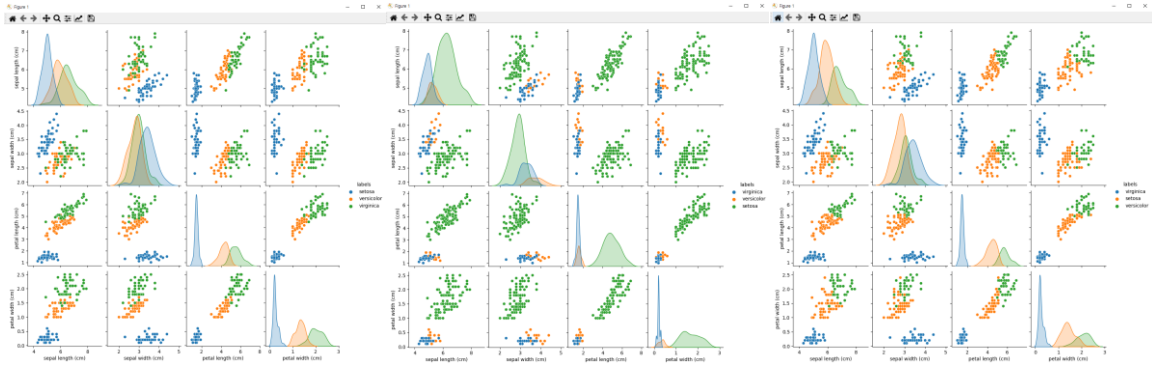
```
PS C:\Users\송채영\Desktop\code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/code/CYS.py
pi :            [0.33003332 0.33677147 0.33319521]
count / total : [0.36       0.30666667 0.33333333]
EM Accuracy: 0.81    Hit: 122 / 150
KM Accuracy: 0.89    Hit: 134 / 150
PS C:\Users\송채영\Desktop\code>
```

⑦



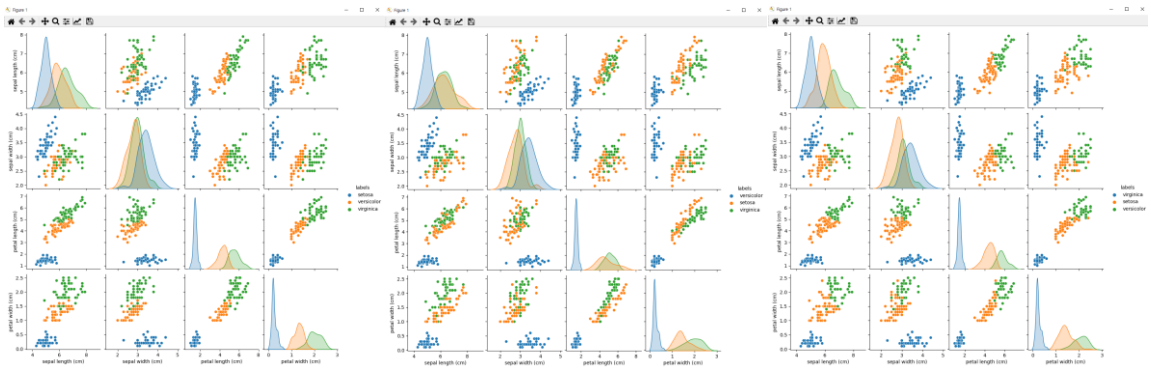
```
PS C:\Users\송채영\Desktop\code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/code/CYS.py
pi :            [0.36747348 0.33333333 0.29919319]
count / total : [0.36666667 0.33333333 0.3       ]
EM Accuracy: 0.97    Hit: 145 / 150
KM Accuracy: 0.89    Hit: 134 / 150
PS C:\Users\송채영\Desktop\code>
```

⑧



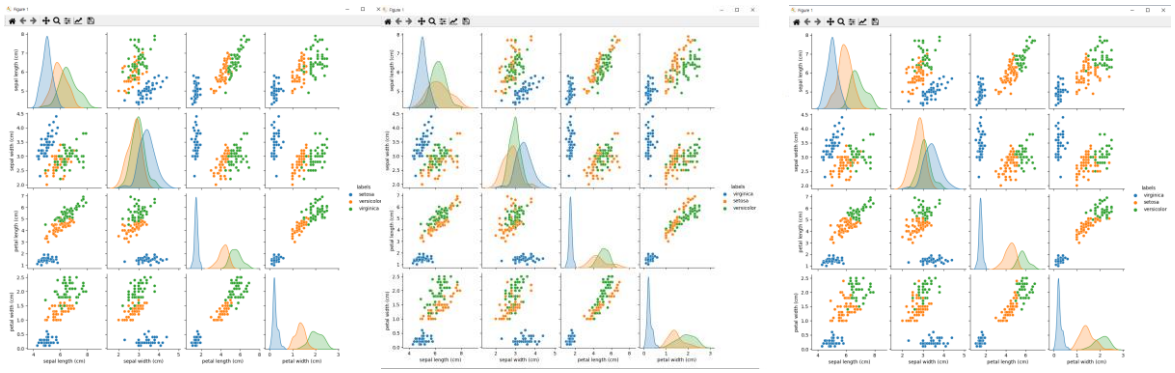
```
PS C:\Users\송채영\Desktop\code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/code/CYS.py
pi :            [0.66667318 0.10632192 0.22700491]
count / total : [0.66666667 0.1        0.23333333]
EM Accuracy: 0.57    Hit: 85 / 150
KM Accuracy: 0.89    Hit: 134 / 150
PS C:\Users\송채영\Desktop\code>
```

⑨



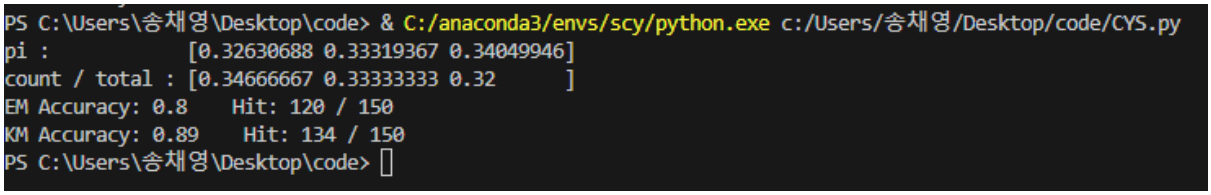
```
PS C:\Users\송채영\Desktop\code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/code/CYS.py
pi :            [0.32774908 0.33319413 0.33905679]
count / total : [0.35333333 0.33333333 0.31333333]
EM Accuracy: 0.81    Hit: 121 / 150
KM Accuracy: 0.89    Hit: 134 / 150
PS C:\Users\송채영\Desktop\code>
```

⑩



```
PS C:\Users\송채영\Desktop\code> & C:/anaconda3/envs/scy/python.exe c:/Users/송채영/Desktop/code/CYS.py
pi :            [0.31250407 0.35430246 0.33319347]
count / total : [0.31333333 0.35333333 0.33333333]
EM Accuracy: 0.82    Hit: 123 / 150
KM Accuracy: 0.89    Hit: 134 / 150
PS C:\Users\송채영\Desktop\code>
```

original, EM, KMeans의 순서이며 총 10번 출력해서 확인해보았다. EM의 경우 77, 97, 74, 97, 57, 81, 97, 57, 81, 82, 80 %가 나와 평균 88%정도이며, KMeans는 평균 89%이다. KMeans의 정확도와 비슷한 것을 확인할 수 있고 iris 데이터세트 자체가 cluster가 뚜렷하고 노이즈가 최소화 되어 꽤 높은 정확도가 나온 것 같다. 또한 EM 알고리즘의 초기화 단계에서 무작위로 초기화 한 점이 성능이 높게 나올 수 있게 작용한 것 같다. 하지만 일부 실행시 50%대의 accuracy가 뽑힌 것을 볼 수 있는데 random하게 했기 때문에 clustering의 초기값이 잘못 들어갈 경우 이러한 결과가 나올 것이라 예상했다.



위 10번의 실행결과에서도 볼 수 있듯이 pi값과 예측된 라벨이 나오는 확률이 비슷한 것을 볼 수 있다. EM 알고리즘은 latent variable의 pdf를 이용해 expectation 단계에서 기대값을 유도하고 Maximization 단계에서 Bayes 정리를 적용해 파라미터를 업데이트 하며 posterior를 최대화하는 파라미터를 찾는다. 즉 latent variable과 pdf의 Kullback-Leibler distance를 고려해 두 분포가 서로 가까울수록 값이 작아진다. 따라서 EM 알고리즘이 최적의 파라미터를 찾기 위해서는 latent variable의 pdf와 posterior 간의 KL-distance를 최소화해야 하고, 이를 위해 priori와 posteriori가 같아진다. (머신러닝 ch 5) 이로 인해 두 확률이 비슷하게 나온 것임을 알 수 있다.

[Reference]

- 머신러닝 과제 제안서
- 머신러닝 ch 5 pdf
- https://seaborn.pydata.org/generated/seaborn.pairplot.html - **seaborn**
- https://rfriend.tistory.com/380 - **linear algebra**
- https://velog.io/@jhdai_ly/%EB%84%98%ED%8C%8C%EC%9D%B4NumpyN%EC%B0%A8%EC%9B%90-%EB%B0%B0%EC%97%B4-%EC%83%9D%EC%84%B12-zeros-ones-full-eye-arange-linspace-logspace – **zeros, ones, eye**
- https://abluesnake.tistory.com/114 - **random choice**