머신러닝

HW02

EM ALGORITHM USING KMEANS FOR GMM

2019202050 이강현

Introduction

꽃 사진 데이터를 기반으로 GMM을 이용한 EM 알고리즘을 구현해본다.

E-step, M-step을 차례로 구현해보면서 label이 없는 데이터에 대해서 latent variable을 선정하고 likelihood를 최대로 하는 파라미터를 단계적으로 반복해 구해내는 EM 알고리즘의 원리를 코드에 적용하여 이해할 수 있다.

추가적으로 이를 partial clustering 방법중 하나인 k-means와 오리지널 데이터와 서로 비교해보면서 차이를 이해한다.

Source Code

먼저 코드상 EM class에 대한 설명이다.

```
class EM:
    # expectation-maximization algorithm, EM algorithm
    # The EM class is a class that implements an EM algorithm u

    # Within the fit function, the remaining functions should b
    # Other functions can be added, but all tasks must be imple
    # You should annotate each function with a description of t

    def __init__(self, n_clusters, iteration):
        # """
        # Parameters
        # ----------
        # n_clusters (int): Num of clusters (num of GMM)
        # iteration (int): Num of iteration
        #     Termination conditions if the model does not conv
        # mean (ndarray): Num of clusters x Num of features
        #     The mean vector that each cluster has.
        # sigma (ndarray): Num of clusters x Num of features x
        #     The covariance matrix that each cluster has.
        # pi (ndarray): Num of labels (num of clusters)
        #     z(x), Prior probability that each cluster has.
        # return None.
        # -------
        # None.

        # """
        self.n_clusters = n_clusters
        self.iteration = iteration
        self.mean = np.zeros((3,4))
        self.sigma = np.zeros((3,4,4))
        self.pi = np.zeros((3))
```

클러스터 개수와 EM 알고리즘의 반복횟수를 인자로 전달받고 클래스의 멤버변수로 가지고 있는다. 추가적으로 mean,sigma,pi 변수가 클래스에 정의되어 있으며 이들은 다변량 가우시안 확률분포의 핵심 파라미터이며 E-step, M-step을 반복하면서 지속적으로 재조정된다.

다음은 class에 정의된 멤버함수들의 대한 설명이다.

```
def initialization(self, data):
    # 1.initialization, 10 points
    # Initial values for mean, sigma, and pi should be assigned.
    # It have a significant impact on performance.
    # your comment here

    # your code here

    # Select 3 unique row indices from 0 to 149 without replacement
    selected_indices = np.random.choice(150, 3, replace=False)

    # Use these indices to select rows from the dataset
    self.mean = data[selected_indices, :]

    # Use the overall covariance of the data as the initial value
    self.sigma[0] = np.cov(data, rowvar=False)
    self.sigma[1] = np.cov(data, rowvar=False)
    self.sigma[2] = np.cov(data, rowvar=False)
    # Set to have the same ratio as the initial value
    self.pi[0] = 1/3
    self.pi[1] = 1/3
    self.pi[2] = 1/3
    return
```

먼저 initialization부분이다. 150개의 데이터 중 random 하게 데이터 3개를 뽑아 초기 cluster의 mean를 결정하고 sigma는 전체 데이터의 공분산 행렬, pi는 1을 cluster개수로 나눈 확률을 동일하게 초기화하였다. 수식적으로 mean을 랜덤하게 선택하게 되면 sigma,pi 또한 영향을 받고 랜덤하게 시작되는 효과를 얻기에 위와 같은 초기화를 진행하였다.

```
def multivariate_gaussian_distribution(self, x, mean, sigma):
    # """ 2.multivariate_gaussian_distribution, 10 points
    # Use the linear algebraic functions of Numpy. π of this function is not self.pi

    # your comment here
    # """

    # Implementing multivariate gaussian distribution formula
    Gaussian_distribution = (1. / (np.sqrt((2 * np.pi)**4 * np.linalg.det(sigma))) * np.exp(-np.dot((x - mean), np.dot(np.linalg.inv(sigma), (x - mean).T)) / 2))
    return Gaussian_distribution
```

multivariate_gaussian_distribution을 그대로 코드화 한 부분이다.

```python
def expectation(self, data):
    # """ 3.expectation step, 20 points
    # The multivariate_gaussian_distribution(MVN) function must be used.

    # your comment here
    # """
    # your code here
    posterior = np.zeros((150,3)) # num of data x num of clusters

    for i in range(data.shape[0]):
        denominator = 0. # denominator initializing
        for j in range(3):
            posterior[i][j] = self.pi[j] * self.multivariate_gaussian_distribution(x=data[i],mean=self.mean[j],sigma=self.sigma[j]) #Li
            denominator += self.pi[j] * self.multivariate_gaussian_distribution(x=data[i],mean=self.mean[j],sigma=self.sigma[j]) # magi

        posterior[i] = posterior[i]/denominator # posterior

    return posterior
```

E-step 부분이다.

먼저 데이터의 개수 x cluster 개수만큼의 posterior 행렬을 만들어주고 2중 for문을 이용해 수식을 구현하였다. 여기서 현재단계의 파라미터(mean,sigma,pi)와 데이터를 이용해 posterior를 계산한다. 이때 latent variable은 pi로 데이터의 라벨이라는 정보가 빠져있고 또한 라벨 각각의 수를 알지 못하는 상황이므로 분모 부분에는 pi를 기준으로 marginalization을 진행한 것을 확인할 수 있다.

```python
def maximization(self, data, posterior):
    # """ 4.maximization step, 20 points
    # Hint. np.outer

    # your comment here
    # """
    # your code here
    denominator = np.sum(posterior, axis=0) # Sum of posteriors for all data

    # By differentiating A, B, and C, the parameter that maximizes the likelihood can be readjus
    for i in range(3): # Repeat as many clusters
        mean = 0.
        sigma = 0.
        for j in range(data.shape[0]): # mean value readjustment
            mean += data[j]*posterior[j][i]
        self.mean[i] = mean / denominator[i]

        for j in range(data.shape[0]): # Sigma value readjustment
            sigma += posterior[j][i] * np.outer((data[j]-self.mean[i]),(data[j]-self.mean[i]))
        self.sigma[i] = sigma / denominator[i]

        self.pi[i] = denominator[i] / data.shape[0] # pi value readjustment

    return
```

M-step부분이다.

E-step에서 구해낸 posterior를 이용하여 파라미터를 재조정하는 부분이다. GMM을 이용해서 모델링한 경우 likelihood를 최대로 만드는 파라미터를 구하기 위해 각각 파라미터를 기준으로 미분했

을때 모두 posterior를 이용하여 재조정이 가능하다. 따라서 mean,sigma,pi를 업데이트 하는 부분이 이 단계에서 이루어진다. 모든 cluster 각각의 파라미터를 재조정한다.

```
def fit(self, data):
    # """ 5.fit clustering, 20 points
    # Functions initialization, expectation, and maximization should be used by default.
    # Termination Condition. Iteration is finished or posterior is the same as before. (Beware of shallow copy)
    # Prediction for return should be formatted. Refer to iris['target'] format.

    # your comment here
    # """
    # your code here
    self.initialization(data=data) # initializing parameter
    for i in range(self.iteration): # Repeat as many times as iteration
        posterior = self.expectation(data=data) # E-step
        self.maximization(data=data,posterior=posterior) # M-step

    prediction = np.argmax(posterior,axis=1) # Predict label using posterior
    return prediction
```

초기화, E-step, M-step, prediction이 모두 이루어지는 부분인 fit 함수이다.

인자로 전달된 꽃 데이터를 초기화 함수 인자로 전달해주어 초기화를 진행하고 E-step, M-step을 iteration 변수의 크기만큼 반복한다. 각 단계가 반복되면서 클러스터링이 완료되고 이후 얻어낸 posterior 값을 비교하여 데이터가 어떤 label을 가지는지를 예측하여 반환한다.

```
def plotting(data, labels):
    # """ 6.plotting, 20 points with report
    # Default = seaborn pairplot

    # your comment here
    # """
    # your code here
    sns.pairplot(pd.DataFrame(data=data, columns=iris['feature_names']).assign(labels=labels), hue='labels') # data plot
    plt.show()
    return
```

오리지널 데이터, EM, K-mean각각의 결과를 보여주기 위한 플롯 함수이다. seaborn 라이브러리의 pairplot을 이용했다.

다음은 main함수 부분이다. 데이터를 불러오고 모델들을 피팅하여 서로 비교하는 코드들이 있다.

```
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']) # Pass data and labels separately

    # Only data is used W/O labels because EM and Kmeans are unsupervised learning
    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']) # Pass data and labels separately
```

오리지널 데이터를 플롯하는 부분과 EM 알고리즘으로 구현된 모델을 구축하고 플롯하는 부분이다.

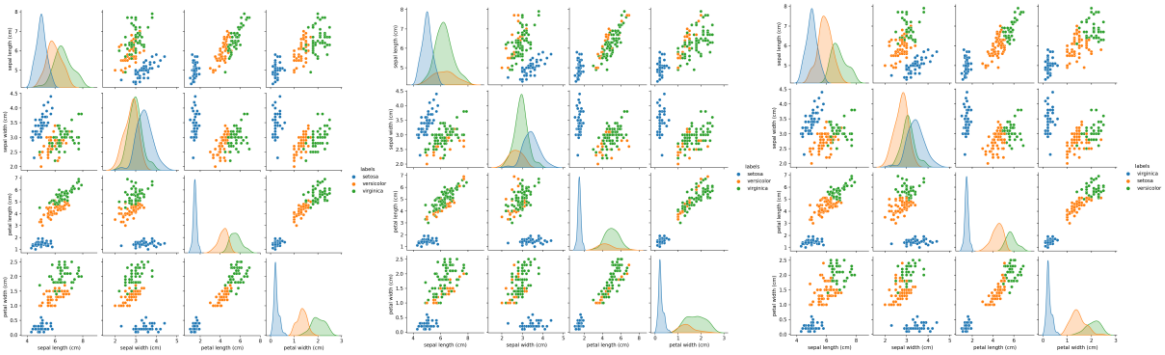
```
# Unsupervised learning(clustering) using KMeans algorithm
KM_model = KMeans(n_clusters=3, init='random', random_state=seed_num, max_iter=iteration).fit(data)
KM_pred = KM_model.predict(data)
KM_pd = pd.DataFrame(data= np.c_[data, KM_pred], columns= iris['feature_names'] + ['labels'])
KM_pd['labels']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

plotting(KM_pd.drop(columns=['labels']), KM_pd['labels']) # Pass data and labels separately
# No need to explain.
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')
```

라이브러리로 제공된 KMeans를 이용해 데이터를 예측하고 플롯하는 부분과 최종적으로 EM과 KM의 정확도를 비교하는 부분이다.

result

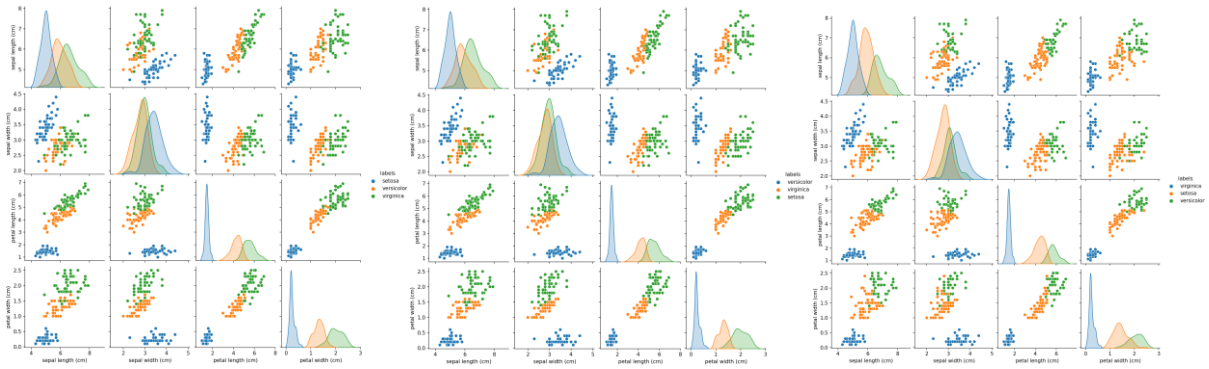
왼쪽부터 오리지널, EM, KM 순서이다.

<1회>



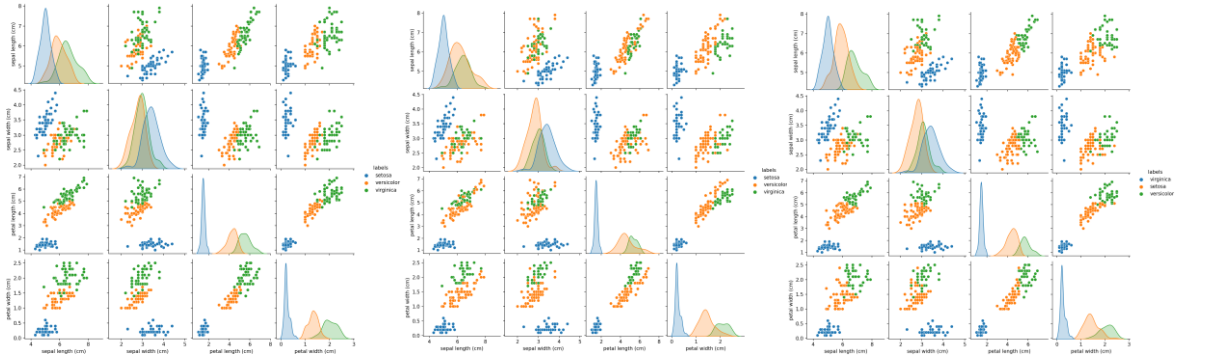
```
PS C:\Users\dlrkd> & C:/Users/dlrkd/anaconda3/envs/ML3_11/python.exe c:/Users/dlrkd/On
eDrive/4-1/머신러닝/3차과제/KHL.py
pi :            [0.33332659 0.13957738 0.52709603]
count / total : [0.33333333 0.16       0.50666667]
EM Accuracy: 0.57    Hit: 86 / 150
KM Accuracy: 0.89    Hit: 134 / 150
```

<2회>



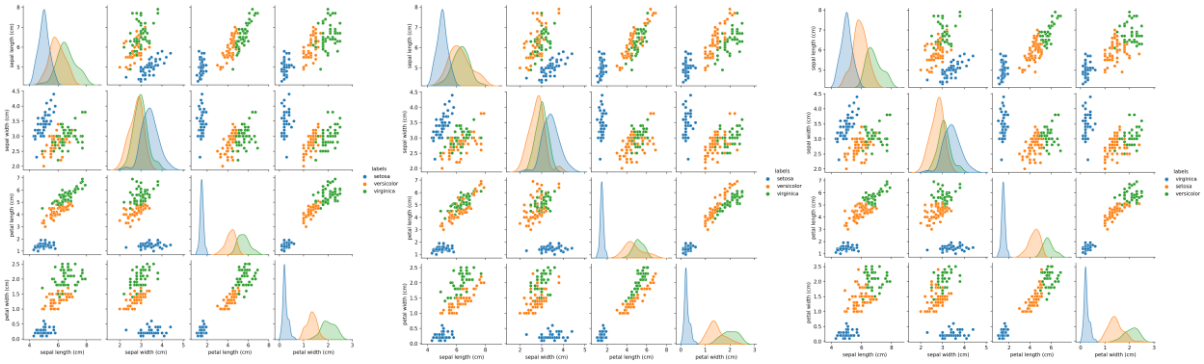
```
PS C:\Users\dlrkd> & C:/Users/dlrkd/anaconda3/envs/ML3_11/python.exe c:/Users/dlrkd/On
eDrive/4-1/머신러닝/3차과제/KHL.py
pi :            [0.36747348 0.33333333 0.29919319]
count / total : [0.36666667 0.33333333 0.3       ]
EM Accuracy: 0.97    Hit: 145 / 150
KM Accuracy: 0.89    Hit: 134 / 150
PS C:\Users\dlrkd>
```

<3회>



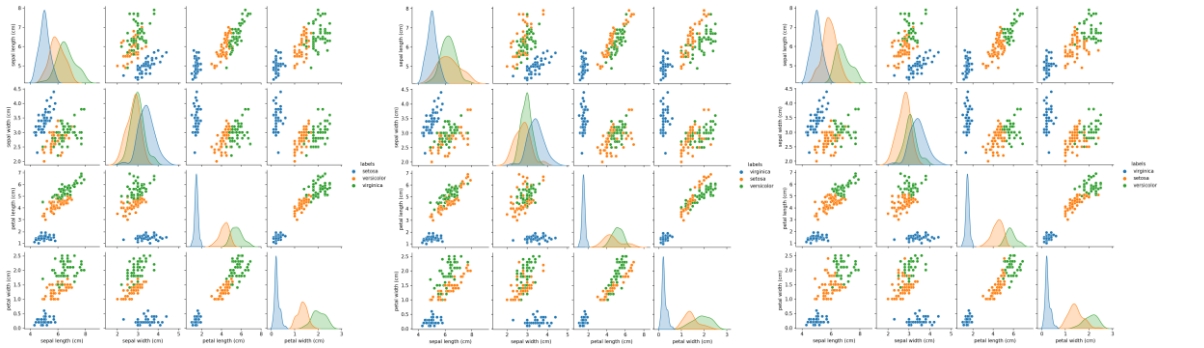
```
PS C:\Users\dlrkd> & C:/Users/dlrkd/anaconda3/envs/ML3_11/python.exe c:/Users/dlrkd/On
eDrive/4-1/머신러닝/3차과제/KHL.py
pi :            [0.33328802 0.43736938 0.22934259]
count / total : [0.33333333 0.43333333 0.23333333]
EM Accuracy: 0.89    Hit: 133 / 150
KM Accuracy: 0.89    Hit: 134 / 150
```

<4회>



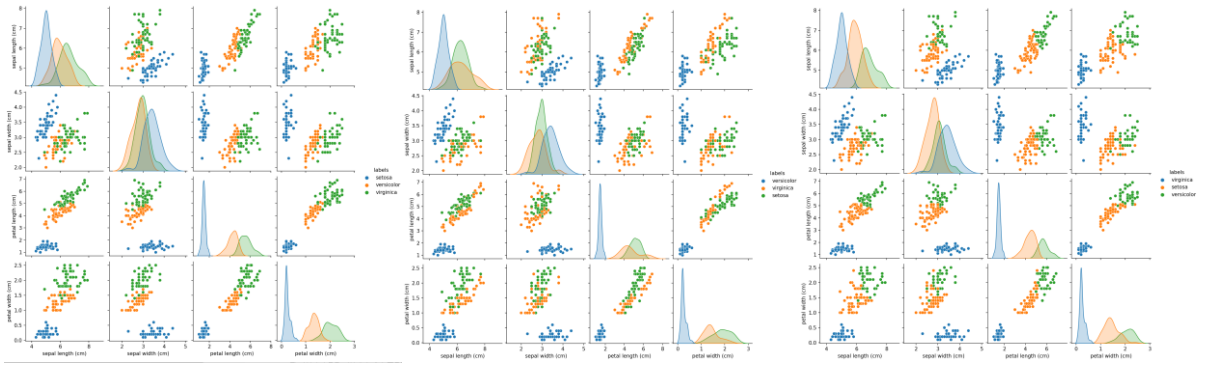
```
PS C:\Users\dlrkd> & C:/Users/dlrkd/anaconda3/envs/ML3_11/python.exe c:/Users/dlrkd/On
eDrive/4-1/머신러닝/3차과제/KHL.py
pi :            [0.3332095  0.34659864 0.32019186]
count / total : [0.33333333 0.38       0.28666667]
EM Accuracy: 0.83    Hit: 125 / 150
KM Accuracy: 0.89    Hit: 134 / 150
```

<5회>



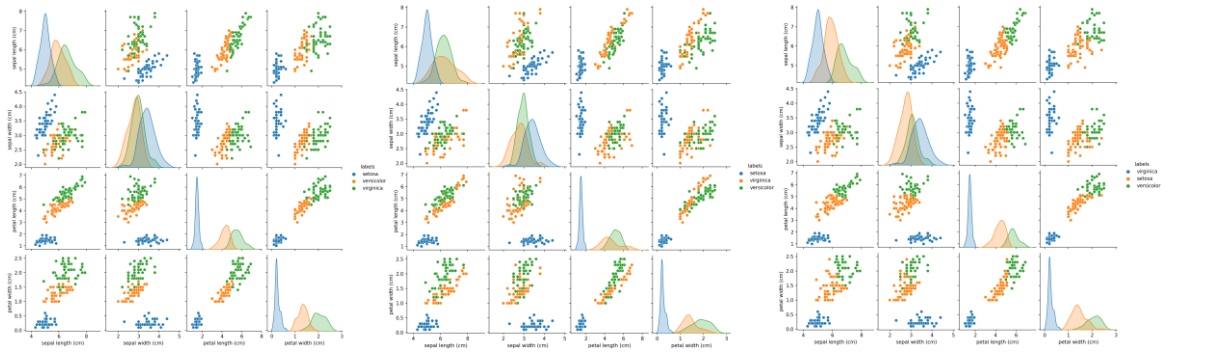
```
PS C:\Users\dlrkd> & C:/Users/dlrkd/anaconda3/envs/ML3_11/python.exe c:/Users/dlrkd/On
eDrive/4-1/머신러닝/3차과제/KHL.py
pi :            [0.31251683 0.35428967 0.3331935 ]
count / total : [0.31333333 0.35333333 0.33333333]
EM Accuracy: 0.82    Hit: 123 / 150
KM Accuracy: 0.89    Hit: 134 / 150
```

<6회>



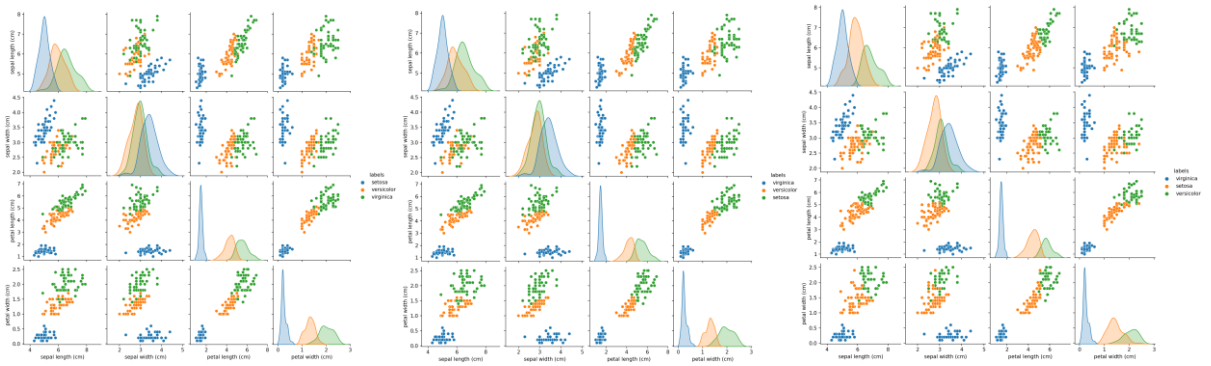
```
PS C:\Users\dlrkd> & C:/Users/dlrkd/anaconda3/envs/ML3_11/python.exe c:/Users/dlrkd/On
eDrive/4-1/머신러닝/3차과제/KHL.py
pi :            [0.3543053  0.33319347 0.31250123]
count / total : [0.35333333 0.33333333 0.31333333]
EM Accuracy: 0.82    Hit: 123 / 150
KM Accuracy: 0.89    Hit: 134 / 150
```

<7회>



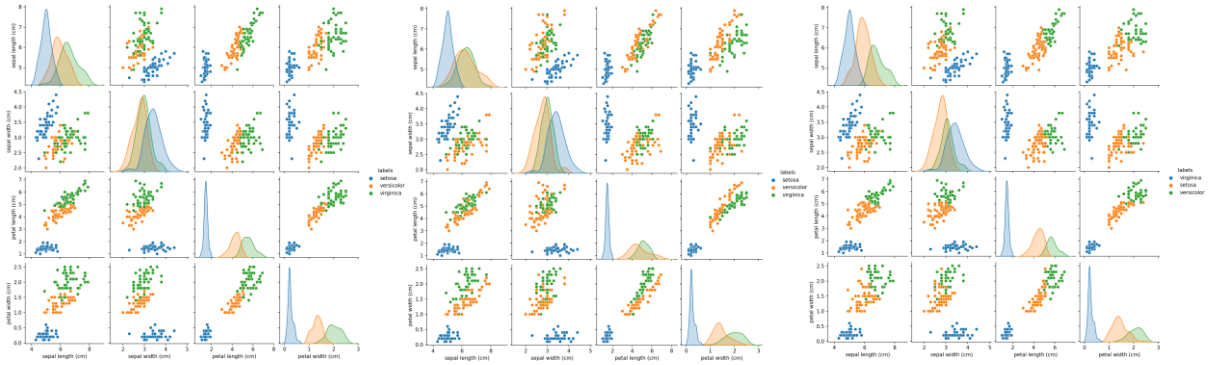
```
PS C:\Users\dlrkd> & C:/Users/dlrkd/anaconda3/envs/ML3_11/python.exe c:/Users/dlrkd/On
eDrive/4-1/머신러닝/3차과제/KHL.py
pi :            [0.33319342 0.35432724 0.31247934]
count / total : [0.33333333 0.35333333 0.31333333]
EM Accuracy: 0.82    Hit: 123 / 150
KM Accuracy: 0.89    Hit: 134 / 150
```

<8회>



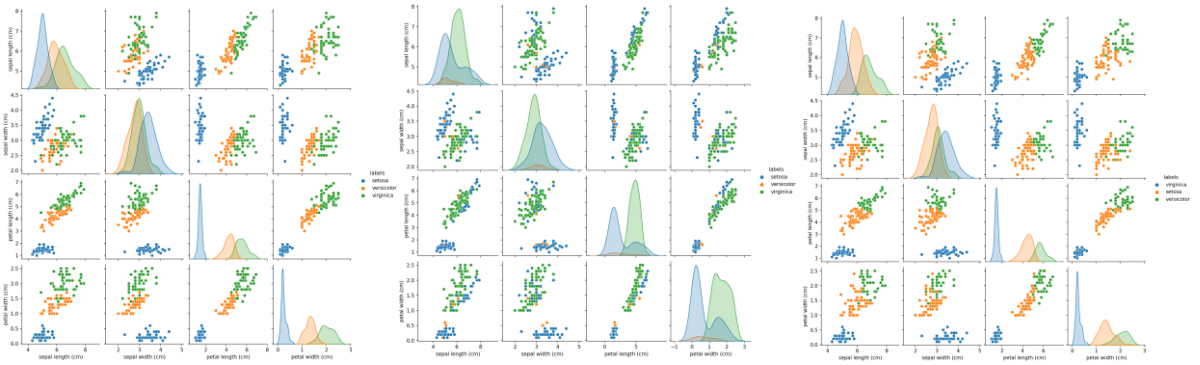
```
PS C:\Users\dlrkd> & C:/Users/dlrkd/anaconda3/envs/ML3_11/python.exe c:/Users/dlrkd/On
eDrive/4-1/머신러닝/3차과제/KHL.py
pi :            [0.36747348 0.29919319 0.33333333]
count / total : [0.36666667 0.3        0.33333333]
EM Accuracy: 0.97    Hit: 145 / 150
KM Accuracy: 0.89    Hit: 134 / 150
```

<9회>



```
PS C:\Users\dlrkd> & C:/Users/dlrkd/anaconda3/envs/ML3_11/python.exe c:/Users/dlrkd/On
eDrive/4-1/머신러닝/3차과제/KHL.py
pi :            [0.33319902 0.33550414 0.33129684]
count / total : [0.33333333 0.36666667 0.3        ]
EM Accuracy: 0.82    Hit: 123 / 150
KM Accuracy: 0.89    Hit: 134 / 150
```

<10회>



```
PS C:\Users\dlrkd> & C:/Users/dlrkd/anaconda3/envs/ML3_11/python.exe c:/Users/dlrkd/On
eDrive/4-1/머신러닝/3차과제/KHL.py
pi :              [0.4543919  0.03809232 0.50751578]
count / total : [0.44666667 0.04       0.51333333]
EM Accuracy: 0.57    Hit: 85 / 150
KM Accuracy: 0.89    Hit: 134 / 150
```

3개의 class 모두 petal length와 petal width변수에서는 비교적 명확한 분류가 이루어졌지만 sepal length와 sepal width변수로는 명확한 분류가 이루어지지 않는 것을 확인할 수 있었다. 핵심적인 해석은 EM 알고리즘을 이용하여 라벨이 존재하지 않고 데이터가 부족한 경우에 존재하는 데이터의 특징 변수들을 이용하여 예측 모델을 구현할 수 있고 존재하는 특징 변수가 얼마나 데이터와 상관성을 지니는지를 플롯을 통해 확인하여 추후 분류 성능의 개선을 위해서 참고하기에 좋은 예시가 될 수 있을 것으로 예상된다.

모델 / 실행 횟수	EM	KM
1	57	89
2	97	89
3	89	89
4	83	89
5	82	89
6	82	89
7	82	89
8	89	89
9	97	89
10	82	89
11	57	89

EM 평균: 89.7% KM 평균: 89%

결과적으로는 KM과 EM이 비슷한 정확도를 보이는 것을 확인했다. EM의 플롯을 확인해 보았을 때 꽃 데이터는 petal에 대한 특징으로 분류하는 것이 더 군집이 겹치지 않고 분류됨을 확인할 수 있었다. 따라서 랜덤하게 초기 군집을 설정하지 않고 특정 변수를 기반으로 초기 clustering을 결정한다면 EM의 성능을 좀 더 올릴 수 있을 것 같다는 생각이다.

```
# Why are these two elements almost the same? Write down the reason in your report. Additional 10 points
print(f'pi :            {EM_model.pi}')
print(f'count / total : {np.bincount(EM_pred) / 150}')
```

추가적으로 pi값과 EM 알고리즘을 이용하여 예측된 라벨이 나올 확률이 같아지는 이유에 대한 설명이다.

EM 알고리즘에서는 latent variable의 pdf를 이용한 수식으로 expectation equation을 유도해 내었다. 이후 bayes thorem을 적용하여 maximization해야하는 값을 정의하고 정의된 값은 latent variable의 pdf와 posterior의 관계성과 연관되어 있었다. 결과적으로 latent variable의 pdf와 posterior의 KL-distance와 관련된 수식이었고 이 값이 0이 되어야 maximazation하는 목적에 부합했기 때문에 priori와 posteriori는 같아야 한다는 결론을 도출했다. 따라서 latent variable에 해당하는 pdf인 pi와 예측값을 결정짓는 posterior가 비슷해지는 결과가 나타난다.

reference

머신러닝 수업 pdf와 과제 제안서

https://m.blog.naver.com/saftyzon/222387168800 pairplot 관련 자료

https://dhpark1212.tistory.com/entry/%EB%8B%A4%EB%B3%80%EB%9F%89-%EA%B0%80%EC%9A%B0%EC%8B%9C%EC%95%88-%EB%B6%84%ED%8F%ACMultivariate-Gaussian-Distribution 다변량 가우시안 분포 관련 자료

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

파이썬 문법 관련 자료