Machine Learning

EM ALGORITHM USING KMEANS FOR GMM

담당교수 : 박철수

제출일 : 2022. 05. 16.

학과 : 컴퓨터정보공학부

학번 : 2017202088

이름 : 신해담

1. 과제 목적

- 임의의 label이 없는 dataset을 가지고 training해서 K개의 cluster로 분류하는 EM algorithm을 구현한다. Data는 4개의 feature를 가지므로 각 cluster는 multivariable normal distribution을 따른다고 가정하며, GMM을 사용하여 학습한다.

2. 코드 설명

- **initialization(data, z)**

  입력 받은 data를 임의의 cluster로 배정하고 mean, sigma, pi를 계산하는 함수이다. NxK 크기의 z에 data의 cluster를 배정하고 shuffle해서 무작위로 초기화된 동일한 크기의 cluster set을 얻는다. 그리고 각 cluster의 mean, sigma, pi를 계산한다.

  mean은 각 cluster의 data의 feature별 평균을 구해서 얻을 수 있다.

  sigma는 각 cluster의 data의 공분산으로 data별 편차를 내적해서 data 수로 나눈다..

  pi는 data 수 / N으로 계산할 수 있다.

```
r, _ = np.shape(data) # data 수

# z[n]을 n_clusters에 균등하게 배정
for n in range(r):
    z[n, n%self.n_clusters] = 1

# shuffle해서 초기 cluster 랜덤
np.random.shuffle(z)

# cluster별 mean, sigma, pi 계산
for k in range(self.n_clusters):
    self.mean[k] = np.mean(data[np.where(z[:,k] == 1)], 0)

    mean_centered = data[np.where(z[:,k] == 1)] - self.mean[k]
    self.sigma[k] = mean_centered.T.dot(mean_centered) / (np.sum(z[:,k])-1)
    #self.sigma[k] = (np.cov(data[np.where(z[:,k] == 1)].T))

    self.pi[k] = np.sum(z[:,k])/r
```

- **multivariate_gaussian_distribution(data, k, z)**

  입력 받은 data의 cluster k에 대한 mvn을 계산한다.

  Data의 feature가 4이므로 data의 편차는 1x4 행렬이며, 4x4 sigma, 4x1 편차의 전치와 내적해서 하나의 scalar 값을 계산할 수 있다.

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

```
# 분자부분
d_m = data-self.mean[k] # 1×4, 데이터값-평균값
inv = np.linalg.inv(self.sigma[k]) # 4×4, inverse of covariance matrix
exp = np.exp((-0.5) * (d_m.dot(inv).dot(d_m.T))) # 1×1 = 1×4 . 4×4 . 4×1
# 분모부분
det = np.linalg.det(self.sigma[k])
denom = np.power((2 * np.pi), np.sum(z[:,k])/2) * np.sqrt(det)

# mvn 계산
g =  exp / denom
```

- **expectation(data, z)**

  각 data의 posterior를 계산해서 cluster를 재배정하는 함수이다.

```
r, _ = np.shape(data) # N
y = np.zeros((r, self.n_clusters)) # N x K

# data의 cluster별 posterior 계산
for n in range(r):
    for k in range(self.n_clusters):
        g = self.multivariate_gaussian_distribution(data[n], k, z)
        y[n,k] = self.pi[k] * g

        # z 초기화
        z[n,k] = 0

    y[n] = y[n]/sum(y[n])

    # posterior가 최대가 되는 cluster에 재배정
    z[n, np.argmax(y[n])] = 1

return y
```

- **maximization(data, y, z)**

  E-step 에서 재배정한 cluster 의 mean, sigma, pi 를 다시 계산하는 함수이다.

```
r, _ = np.shape(data) # N

# covariance matrix 초기화
self.sigma = np.zeros((3,4,4))

# 각 cluster별로 mu, sigma, pi 재계산
for k in range(self.n_clusters):
    # 분모
    denom = np.sum(y[np.where(z[:,k] == 1)][:,k])

    # 분자
    num = np.sum([y[n,k]*data[n] for n in np.where(z[:,k] == 1)[0]], 0)

    # mean 계산
    self.mean[k] = num/denom

    # sigma 계산
    for n in np.where(z[:,k] == 1)[0]:
        # transpose 함수 사용 가능하도록 array to matrix
        d = np.array(data[n]-self.mean[k])[np.newaxis]
        self.sigma[k] += y[n,k] * np.transpose(d).dot(d)
    self.sigma[k] /= denom

    # 기댓값으로 pi 계산
    self.pi[k] = np.sum(y[np.where(z[:,k] == 1)][:,k])/r
```

- **fit(data)**

  입력받은 data로 초기화 및 E-step, M-step을 반복함으로써 data를 clustering하는 함수이다. Cluster 예측이 이전과 같으면 반복을 종료한다.

```
r,_ = np.shape(data) # N

# NxK matrix
# data의 cluster면 1, 아니면 0
z = np.zeros((r, self.n_clusters)).astype(int) # 0으로 초기화

# EM 초기화
self.initialization(data, z)

# z 이전값
z_prev = copy.copy(z)

# 최대 iteration만큼 반복
for _ in range(self.iteration):
    # e step
    y = self.expectation(data, z)

    # z가 이전과 같으면 반복 탈출
    if (z==z_prev).all():
        break
    else: # z_prev 갱신
        z_prev = copy.copy(z)

    # m step
    self.maximization(data, y, z)

# data의 cluster로 구성된 np array(150)
prediction = np.array([np.argmax(z[i]) for i in range(r)])

return prediction
```

- **plotting(data)**

  입력받은 data로 산점도 행렬을 출력하는 함수이다.

```
# 산점도 행렬을 각 변수별 커널밀도추정곡선을 볼 수 있도록,
# labels별로 색을 다르게해서, 출력색을 bright로 출력한다.
sns.pairplot(data, diag_kind='kde', hue="labels", palette='bright')
plt.show(block=True)
return
```

- **EM class**

  EM class는 class를 초기화하는 __init__ 함수, EM step 전 초기값을 설정하는 initialization 함수, E-step 역할의 expectation함수, M-step 역할의 maximization 함수, E-step에서 posterior를 구하기 위해 likelihood 계산에 사용하는 multivariable_gaussian_distribution 함수, 초기화 및 E-step, M-step을 반복해서 학습하는 fit 함수로 구성된다.

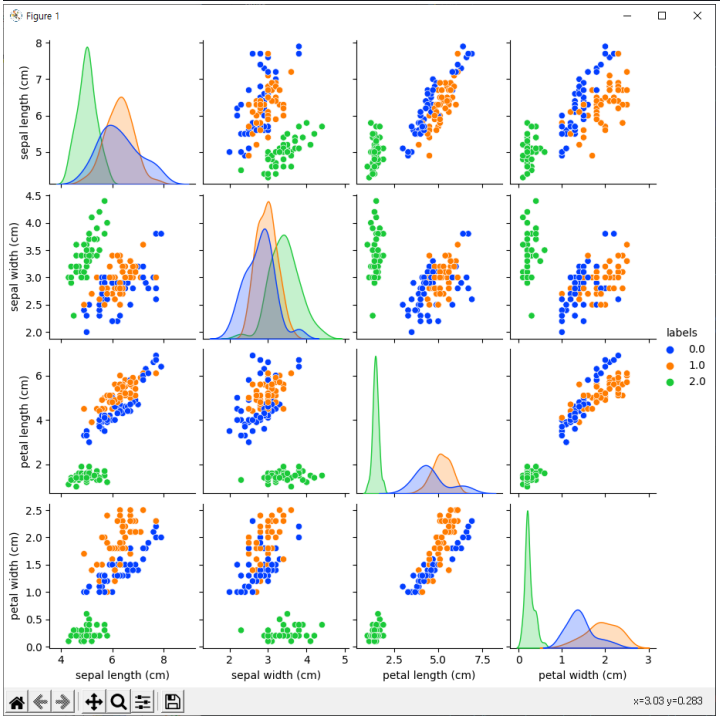
  main에서 EM class에 원하는 결과 cluster 수와 최대 반복 iteration을 입력해서 초기화한 다음, clustering할 data로 fit 함수를 호출함으로써 cluster 예측 결과를 얻을 수 있다.

```
# Unsupervised learning(clustering) using EM algorithm
EM_model = EM(n_clusters=3, iteration=iteration)
EM_pred = EM_model.fit(data) # EM으로 학습
EM_pd = pd.DataFrame(data= np.c_[data, EM_pred], columns= iris['feature_names'] + ['labels'])
plotting(EM_pd)
```

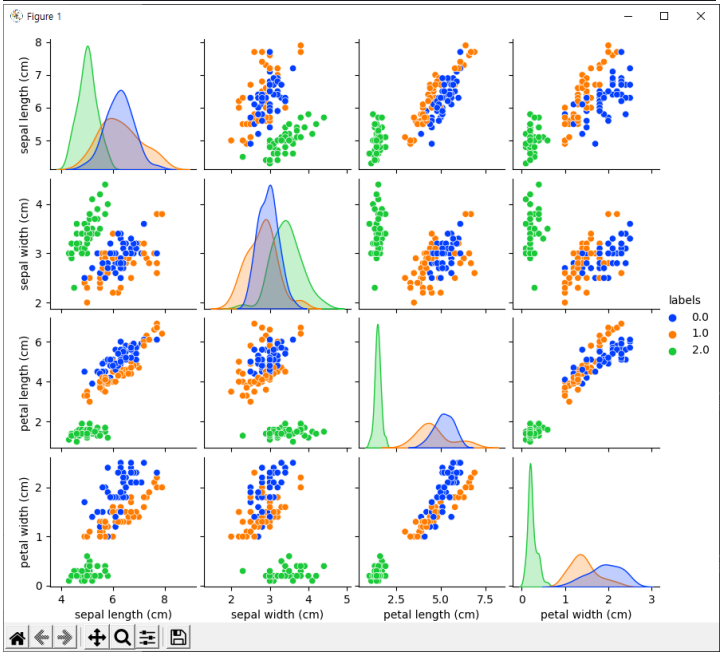
3. 실행 결과

- **10회 실행 결과 및 Plot**

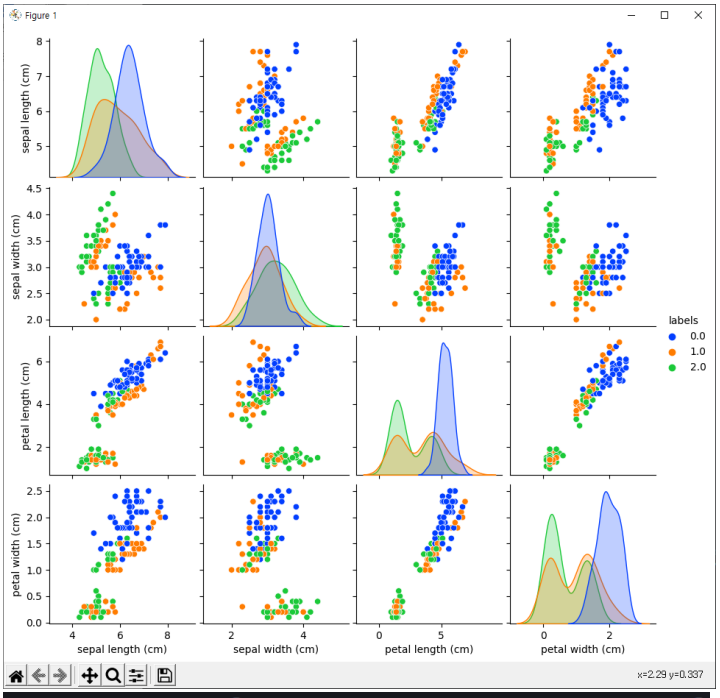
```
pi :            [0.30680737 0.31420973 0.33322413]
count / total : [0.33333333 0.33333333 0.33333333]
EM Accuracy: 0.85     Hit: 128 / 150
KM Accuracy: 0.89     Hit: 134 / 150
```



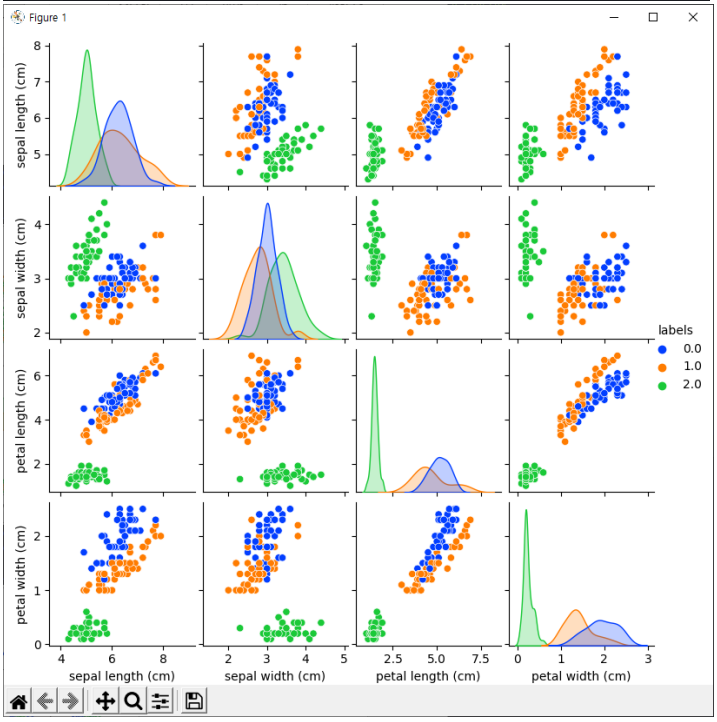

```
pi :            [0.31440915 0.30209587 0.33329856]
count / total : [0.33333333 0.33333333 0.33333333]
EM Accuracy: 0.85     Hit: 128 / 150
KM Accuracy: 0.89     Hit: 134 / 150
```



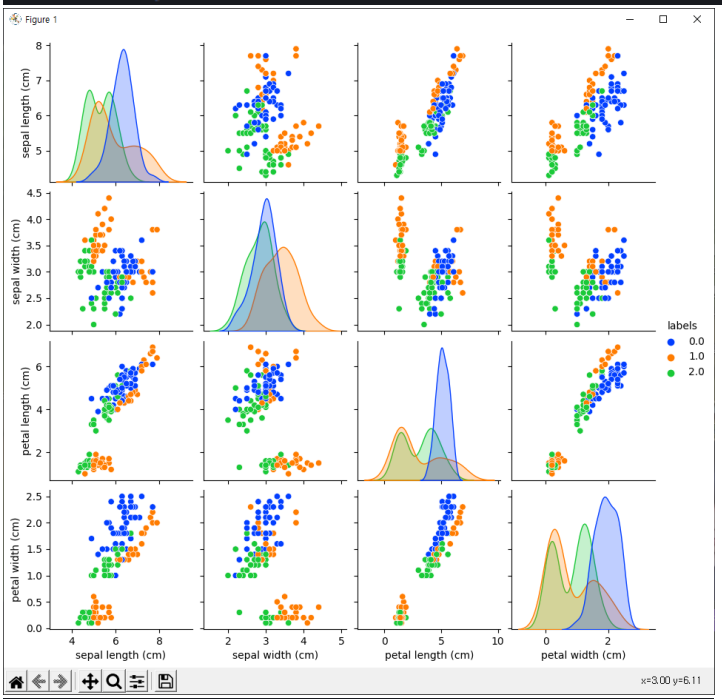
```
pi :              [0.31047837 0.31554164 0.30754648]
count / total : [0.34       0.33333333 0.32666667]
EM Accuracy: 0.64     Hit: 96 / 150
KM Accuracy: 0.89     Hit: 134 / 150
```



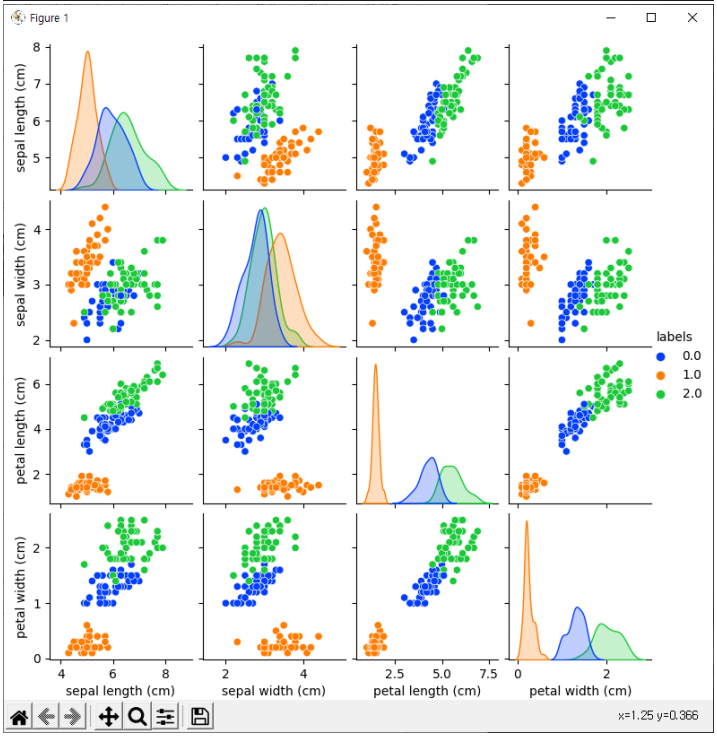

```
pi :              [0.31449272 0.30817183 0.33318581]
count / total : [0.33333333 0.33333333 0.33333333]
EM Accuracy: 0.83     Hit: 124 / 150
KM Accuracy: 0.89     Hit: 134 / 150
```



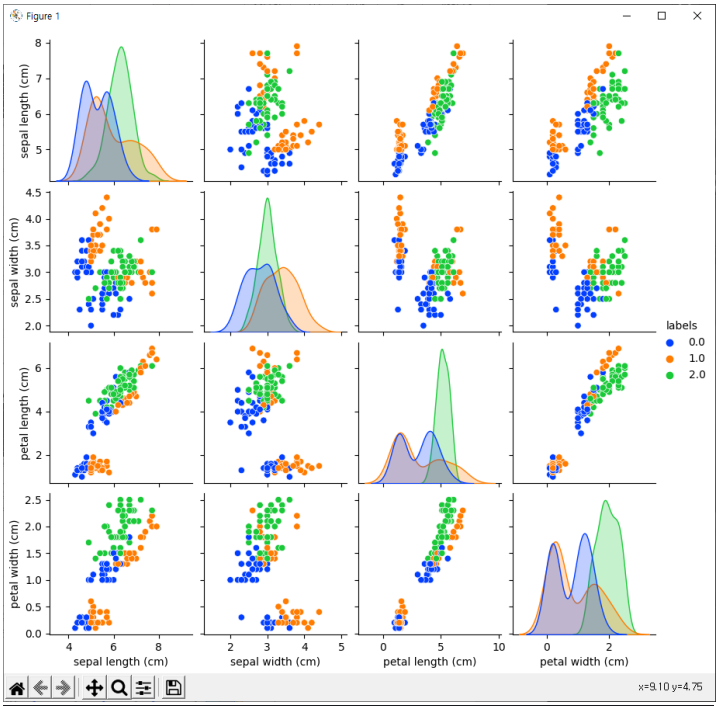
```
pi :              [0.32109509 0.30088532 0.29352335]
count / total :  [0.33333333 0.33333333 0.33333333]
EM Accuracy: 0.61     Hit: 91 / 150
KM Accuracy: 0.89     Hit: 134 / 150
```



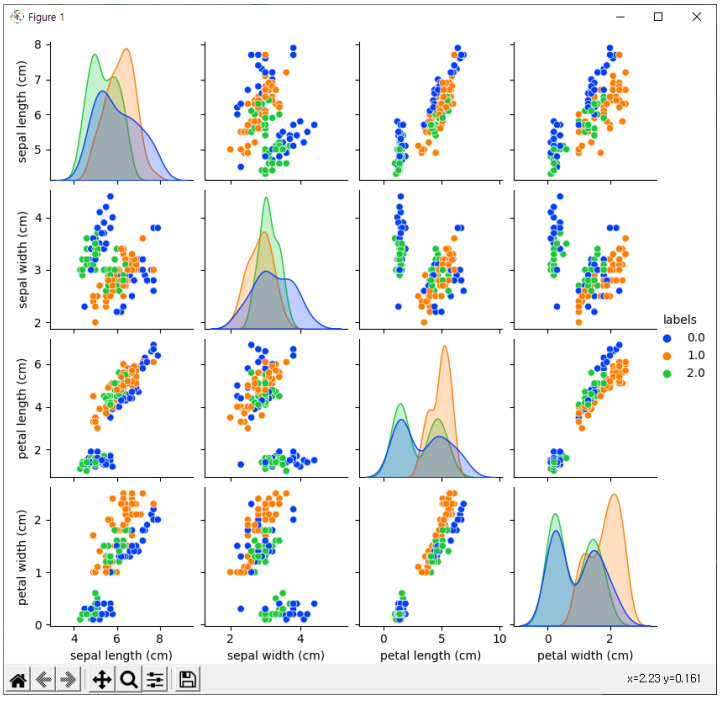

```
pi :              [0.32443725 0.33333333 0.32817005]
count / total :  [0.32666667 0.33333333 0.34        ]
EM Accuracy: 0.98     Hit: 147 / 150
KM Accuracy: 0.89     Hit: 134 / 150
```



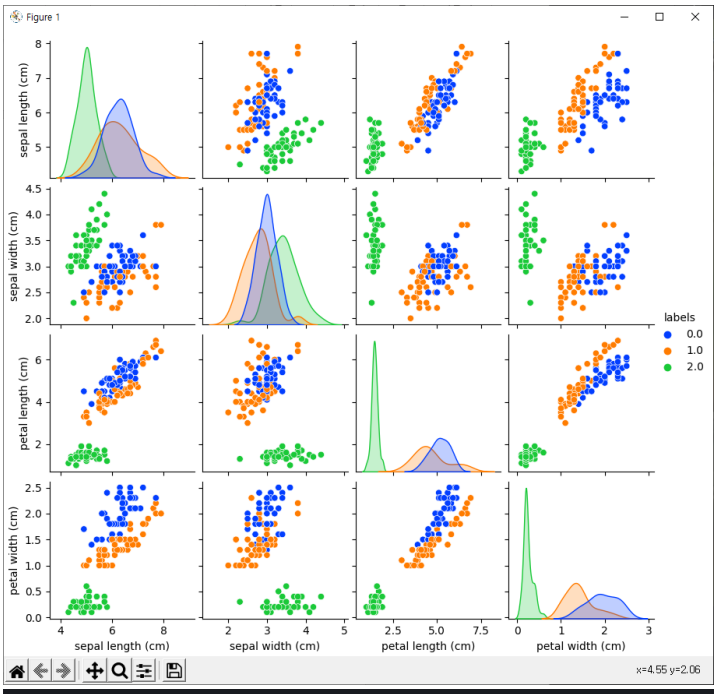
```
pi :              [0.29699197 0.30323241 0.3194094 ]
count / total :  [0.33333333 0.33333333 0.33333333]
EM Accuracy: 0.62     Hit: 93 / 150
KM Accuracy: 0.89     Hit: 134 / 150
```



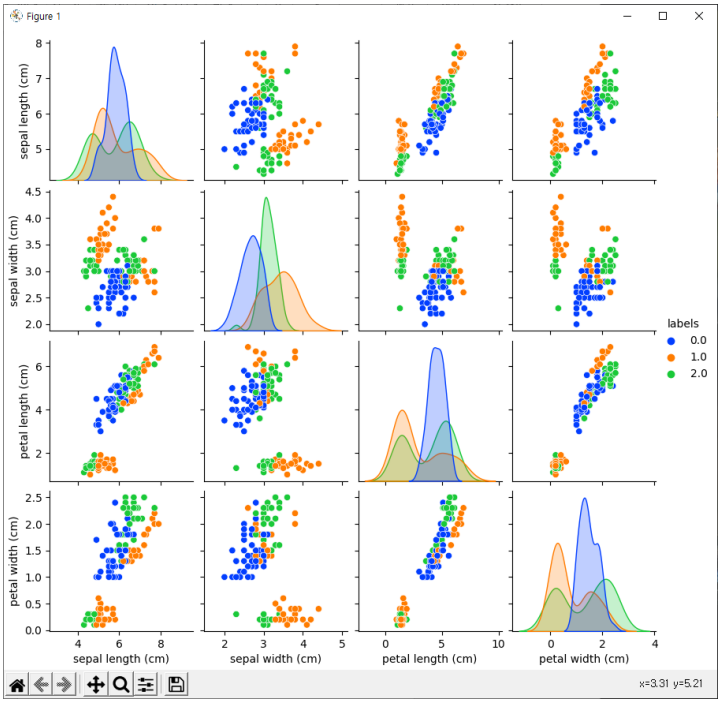

```
pi :              [0.29714643 0.31537369 0.29318718]
count / total :  [0.33333333 0.33333333 0.33333333]
EM Accuracy: 0.19     Hit: 29 / 150
KM Accuracy: 0.89     Hit: 134 / 150
```



```
pi :              [0.31085033 0.31281868 0.33319065]
count / total :   [0.32666667 0.34       0.33333333]
EM Accuracy: 0.83     Hit: 125 / 150
KM Accuracy: 0.89     Hit: 134 / 150
```




```
pi :              [0.3047064  0.31956162 0.28621088]
count / total :   [0.34       0.33333333 0.32666667]
EM Accuracy: 0.57     Hit: 85 / 150
KM Accuracy: 0.89     Hit: 134 / 150
```

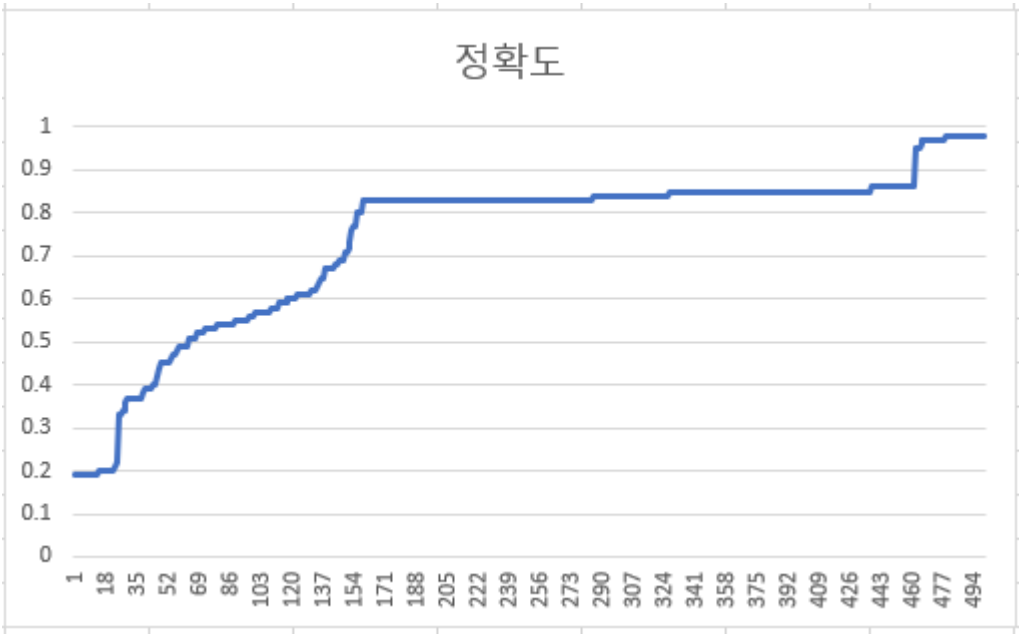


- **결과 분석**

  EM algorithm은 E-step에서 cluster를 업데이트하고, M-step에서 업데이트된 cluster로 mean, sigma, pi를 업데이트한다. 이 양 step을 반복하면 최적 cluster를 찾을 수 있다.

  10회 수행 결과에서 5번은 80% 이상의 정확도, 4번은 55% 근방의 정확도를, 1번은 20%의 완전 틀린 정확도를 보였다. 80% 미만의 정확도를 가지는 결과는 cluster의 일부 또는 전체가 initialization 과정에서 original과는 다른 long shape로 분류되어 EM step을 반복해도 초기 분포를 크게 벗어나지 못해 발생하는 것으로 추정된다. 일부 경우에는 iteration 횟수를 증가시키면 정확도가 증가할 수도 있을 것이다.

- **500회 실행 결과**



  약 70%가 0.8 이상의 정확도를 보였고, 약 13%가 0.5 미만의 정확도를 보였다.

- **Why are these two elements almost the same?**

  M-step에서 pi의 계산식은 아래와 같다.

$$\pi_k = \frac{1}{N} \sum_{n=1}^{N} \gamma(z_{nk})$$

  1/N은 각 data[n]의 확률이며, $y(z_{nk})$는 data[n]이 k에 속할 확률이다. 따라서 pi는 data가 cluster k에 속할 기대확률이라 할 수 있으며, 이는 전체 data 중 cluster k에 속한 data의 확률과 유사한 값을 가지게 된다.

4. Reference

- Scatterplot Matrix. https://rfriend.tistory.com/416
- Covariance Matrix. https://yonghyuc.wordpress.com/2019/07/13/covariance-matrix-%EA%B3%B5%EB%B6%84%EC%82%B0-%ED%96%89%EB%A0%AC/
- 박철수. Clustering and EM (Machine Learning Ch.5)