Summary Report [Week 3]. Numpy & Plotting Data

20190106 KimByungjun (김병준)

● Theoretical Background

1) Multivariate Gaussian Distribution and Independence

다변수 정규분포에 대한 probability distribution function 은 아래와 같다. ($\mathbf{x}$: **column vector)**

$$f(\mathbf{x}) = \sqrt{\frac{1}{(2\pi)^{n}\det(\Sigma)}}\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{T}\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

이변수 정규분포인 경우, $n = 2$ 이며 <u>특별하게 covariance matrix 가 diagonal 한 경우</u> (i.e. $\Sigma = \begin{bmatrix}\mathrm{Cov}(X,X) & \mathrm{Cov}(X,Y)\\ \mathrm{Cov}(X,Y) & \mathrm{Cov}(Y,Y)\end{bmatrix} = \begin{bmatrix}\sigma_X^2 & 0\\ 0 & \sigma_Y^2\end{bmatrix}$) 두 확률 변수은 independent 하다. 일반적인 경우 $\mathrm{Cov}(X,Y) = 0$ 라면 X, Y가 independent 한 것은 아니며, X, Y 가 하나의 multivariate normal distribution 을 따르는 경우로 한정될 때는 독립이다.

Proof. $\det(\Sigma) = \sigma_X^2\sigma_Y^2$, $\Sigma^{-1} = \begin{bmatrix}\frac{1}{\sigma_X^2} & 0\\ 0 & \frac{1}{\sigma_Y^2}\end{bmatrix}$ 이므로, probability density function $f(\mathbf{x}) = f(x,y)$ 는,

$$f(x,y) = \frac{1}{2\pi\sigma_X\sigma_Y}\exp(-\frac{1}{2}[x-\mu_X, y-\mu_Y]\begin{bmatrix}\frac{1}{\sigma_X^2} & 0\\ 0 & \frac{1}{\sigma_Y^2}\end{bmatrix}\begin{bmatrix}x-\mu_X\\ y-\mu_Y\end{bmatrix}) = \frac{1}{2\pi\sigma_X\sigma_Y}\exp(-\frac{1}{2}\{\left(\frac{x-\mu_X}{\sigma_X}\right)^2 + \left(\frac{y-\mu_Y}{\sigma_Y}\right)^2\})$$

$$= \frac{1}{\sqrt{2\pi}\sigma_X}\exp(-\frac{1}{2}\left(\frac{x-\mu_X}{\sigma_X}\right)^2) \times \frac{1}{\sqrt{2\pi}\sigma_Y}\exp(-\frac{1}{2}\left(\frac{y-\mu_Y}{\sigma_Y}\right)^2) = f(x)f(y)$$

이 때, $\frac{f(x,y)}{f(x)} = f(y|x) = f(y)$ 가 성립하므로 diagonal 한 covariance matrix 가 주어질 때, 두 확률변수 X, Y 는 independent 하다.

● Exercises

Generate 1000 data points from each of 3 multivariate normal distributions:

$$\boldsymbol{X \sim N(\mu, \Sigma)\ where\ \mu_1 = [0\ \ 10],\ \mu_2 = [10\ \ 0],\ \mu_3 = [20\ \ 20],\ \Sigma = \begin{bmatrix}3 & 0\\ 0 & 3\end{bmatrix}}$$

Plot the data and centroid of data sampled from each of the distributions.

```
#import numpy library
import numpy as np
```

```
[2] import matplotlib.pyplot as plt
```

[Fig 1] Library / module import

Exercises 해결에 앞서 필요한 함수들을 사용하기 위해 numpy 라이브러리와 matplotlib 라이브러리 내 pyplot 모듈을 위 **[Fig 1]**과 같이 호출한다.

1) Generate 1000 data points from each normal distribution (use np.random.normal)

< Explanation >

주어진 covariance matrix $\boldsymbol{\Sigma}$ 는 covariance 값이 0 인 경우에 해당하며, 위 Theoretical Background 에서 설명하였듯이 $\boldsymbol{X = [x, y]}$ 내 $\boldsymbol{x, y}$ 두 variables 는 서로 independent 하며 각각의 probability distribution 을 분리하여 생각해도 무방하다. 이는, ① np.random.normal 함수가 scalar input 에 대해 적용하는 함수이지 해당 문제와 같이 다변수인 경우에 적용하기 조금 까다롭다

판단되었고, ② 더 나아가 $\mathrm{Cov}(X, Y) \neq 0$ 인 경우에는 쓸 수 없는 (대신 np.random.multivariate_normal 을 사용해야 한다) 함수이므로 $\boldsymbol{x}, \boldsymbol{y}$ 를 분리시켜 각각에 대해 np.random.normal 를 쓸 것이다. Code 는 **[Fig 2]** 및 아래와 같이 진행된다.

Step 1. Sampling Data 의 크기인 n 을 1000 으로 정한다.

Step 2. 다변수 확률 분포 x_1, x_2, x_3의 mean vector **mu1, mu2, mu3**, 그리고 covariance matrix **cov** 를 np.array object 로 정의한다.

Step 3. x1 확률 분포 내 두 변수가 independent 함을 위에서 설명하였다. 이를 바탕으로, 두 변수를 각각 x1_x, x1_y 라 하고, 각 변수에 대해 np.random.normal 함수로 n=1000 개의 data 를 sampling 한다. 함수의 parameter 로 mean 은 x1_x 에선 mu1[0]=0, x1_y 에선 mu1[1]=10 이고

standard deviation 은 x1_x 에선 cov[0][0]**(1/2)= $\sqrt{3}$, x1_y 에선 cov[1][1]**(1/2)=$\sqrt{3}$로 대응되는 좌표에서 값을 가져온다. size 는 n 을 써서 x1_x, x1_y 에 1000 개의 값을 저장한다. x2, x3 에도 똑같이 x2_x, x2_y 와 x3_x, x3_y 를 다른 mean 을 가지고 만든다.

Step 4. 만든 x1_x, x1_y 를 np.vstack 함수를 이용하여 수직으로 1000 개와 1000 개 data 를 결합해 x1 을 만든다. x2, x3 도 동일 방식으로 만든다.

Step 5. x1, x2, x3 의 차원(dimension)을 shape 을 이용해 확인한다.

Result. x1, x2, x3 의 shape 모두 np.vstack() 이후 (2,1000)으로 변한다.

Note. Data 가 잘 추출됐는 지 확인을 위해 [Fig 2]의 빨간색 상자를 uncomment 하면 된다. x1 을 보면 첫번째 성분(x1_x)의 대략적인 평균이 0, 두번째 성분(x1_y)은 약 10 인 걸 추측할 수 있고 x1_x, x1_y 가 잘 결합되어 있음을 확인할 수 있다. x2, x3 도 sampling 이 잘 되어 있고, 1000 개가 많으니 '...' 로 생략되어 있다.

```
[5] n = 1000
    ############################
    mu1 = np.array([0, 10])
    mu2 = np.array([10, 0])
    mu3 = np.array([20, 20])
    cov = np.array([[3, 0],[0, 3]])

    x1_x = np.random.normal(mu1[0],cov[0][0]**(1/2),size=n)
    x1_y = np.random.normal(mu1[1],cov[1][1]**(1/2),size=n)
    x2_x = np.random.normal(mu2[0],cov[0][0]**(1/2),size=n)
    x2_y = np.random.normal(mu2[1],cov[1][1]**(1/2),size=n)
    x3_x = np.random.normal(mu3[0],cov[0][0]**(1/2),size=n)
    x3_y = np.random.normal(mu3[1],cov[1][1]**(1/2),size=n)
    x1 = np.vstack([x1_x, x1_y])
    x2 = np.vstack([x2_x, x2_y])
    x3 = np.vstack([x3_x, x3_y])
    # print("₩nx1 data points : ₩n", x1)
    # print("₩nx2 data points : ₩n", x2)
    # print("₩nx3 data points : ₩n", x3)

    ############################

    x1.shape, x2.shape, x3.shape

    ((2, 1000), (2, 1000), (2, 1000))
```

[Fig 2] Code for Exercise 1

```
x1 data points :
 [[-1.21102193 -2.01564828  3.00504107 ... -0.41440451 -0.55821004
   2.91607227]
 [10.09054891 10.38089687 10.20760943 ... 10.29240568 10.49682729
  11.22275584]]

x2 data points :
 [[ 8.21010518 10.68544772 10.40348715 ...  7.60395241 10.37575772
   9.93229287]
 [-0.32312812 -3.48517116  0.17180039 ...  0.89097219 -0.07187158
  -1.15617227]]

x3 data points :
 [[18.51843368 19.03292144 19.94473474 ... 20.59438574 19.13697578
  19.14347163]
 [22.1081375  23.36045303 23.25844126 ... 18.03155113 21.15255835
  17.45079399]]
```

[Fig 3] Data for x1, x2, x3

```
x1_mean = [x1_x.sum()/n,x1_y.sum()/n]
x2_mean = [x2_x.sum()/n,x2_y.sum()/n]
x3_mean = [x3_x.sum()/n,x3_y.sum()/n]

x1_cov = np.cov(x1_x,x1_y)
x2_cov = np.cov(x2_x,x2_y)
x3_cov = np.cov(x3_x,x3_y)

print("x1 mean :", x1_mean)
print("x1 covariance :₩n", x1_cov)
print("x2 mean :", x2_mean)
print("x2 covariance :₩n", x2_cov)
print("x3 mean :", x3_mean)
print("x3 covariance :₩n", x3_cov)

x1 mean : [0.030414545983788534, 10.032799883231212]
x1 covariance :
 [[3.08784754 0.15405206]
 [0.15405206 3.05935759]]
x2 mean : [10.012932287111006, -0.074388188751877751]
x2 covariance :
 [[ 3.07869917 -0.0070406 ]
 [-0.0070406   3.05275198]]
x3 mean : [19.908202254697155, 20.0281325595139]
x3 covariance :
 [[2.88324741 0.05486394]
 [0.05486394 3.18537139]]
```

[Fig 4] Code / Result for Exercise 2

2) Compute the mean and covariance for x1, x2, x3

< Explanation >

이 문제의 point 는 Exercise1 에서 설정했던 mean, covariance matrix 가 실제 sampling 한 1000 개의 data points 에서 계산한 mean, covariance matrix 와 유사한 지 확인하는 것이다.

Step 1. x1 의 mean (x1_mean)은 x1_x 의 총합, x1_y 의 총합 후 size 인 n 으로 나누면 된다. 총합은 sum()함수로 구하며, x2, x3 도 동일하다.

Step 2. x1 의 covariance matrix(x1_cov)는 np.cov 함수를 사용하여, 아래 Σ 값을 계산한다. x2, x3 도 동일하다.

$$\Sigma = \begin{bmatrix} \text{x1\_x 의 variance} & \mathrm{cov}(\text{x1\_x}, \text{x1\_y}) \\ \mathrm{cov}(\text{x1\_x}, \text{x1\_y}) & \text{x1\_y 의 variance} \end{bmatrix}$$

Step 3. Step 1 과 **Step 2** 에서 구한 수치들을 출력한다.

Result. [Fig 4] 와 같이 x1_mean, x2_mean, x3_mean 은 각각 $\mu_1 = [0 \;\; 10]$, $\mu_2 = [10 \;\; 0]$, $\mu_3 = [20 \;\; 20]$ 와 유사한 값을 보이며, 모든 covariance matrix 는 $\Sigma = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}$ 와 유사함을 알 수 있다. 당연히 무작위로 finite(n=1000)하게 sampling 하였기 때문에 초기 parameter 로 설정한 μ, Σ 와는 오차가 있다.

3) Plot the sampled data and the centroids

< Explanation >

Step 1. Pyplot 모듈 내 plot 함수를 이용한다. 이 때, x 축은 x1_x, y 축은 x1_y 를 인자로 두고, colored dot 형태의 marker 및 구분 가능한 label 을 설정한다.

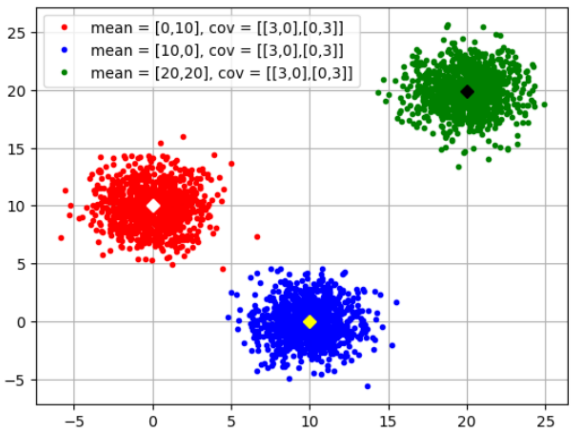
```
###########################
plt.plot(x1_x, x1_y, label='mean = [0,10], cov = [[3,0],[0,3]]', marker = '.', color='red', linestyle="None")
plt.plot(x1_mean[0], x1_mean[1], marker = 'D', color='white', linestyle="None")
plt.plot(x2_x, x2_y, label='mean = [10,0], cov = [[3,0],[0,3]]', marker = '.', color='blue', linestyle="None")
plt.plot(x2_mean[0], x2_mean[1], marker = 'D', color='yellow', linestyle="None")
plt.plot(x3_x, x3_y, label='mean = [20,20], cov = [[3,0],[0,3]]', marker = '.', color='green', linestyle="None")
plt.plot(x3_mean[0], x3_mean[1], marker = 'D', color = 'black', linestyle="None")
###########################

plt.grid()
plt.legend(loc='upper left')
plt.show()
```

[Fig 5] Code for Exercise 3

Step 2. Centroid 표시를 위해 **Exercise 2** 에서 구한 값을 좌표로 만든 (x1_mean[0], x1_mean[1])을 colored diamond 로 좌표평면 상에 표시한다.

Step 3. x2, x3 에 대해서도 똑같이 진행하고(색상은 다르게), grid 생성(<u>plt.grid()</u>), 좌표평면 좌측 상단에 legend(범례) 추가(<u>plt.legend(loc='upper left')</u>) 후 화면상 그래프를 표시한다(<u>plt.show()</u>).

Result. [Fig 6] 의 범례에도 나타냈듯이, **red** 가 x1, **blue** 가 x2, **green** 이 x3 의 distribution 을 나타낸 것이다.



[Fig 6] Result of Exercise 3

Centroid color: white(x1), yellow(x2), black(x3)

① 각 data points group 의 centroid 도 분포 내 diamond 형태로 표시하였는데, group 의 중심에 잘 위치해있으며 grid 상에 (0,10), (10,0), (20,20) 언저리에 잘 걸쳐 있음을 쉽게 알 수 있다.

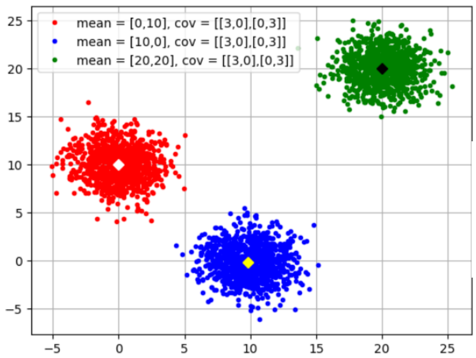
② 세 distribution x1, x2, x3 가 동일한 covariance matrix 를 parameter 로 형성되어 있고, Exercise 2 의 result 에서도 봤듯이 유사한 covariance matrix 를 보였다. 이는 **[Fig 6]**의 data points 가 흩어진 형태를 보면 알 수 있듯이, centroid 를 중심으로 점들이 뭉쳐있는 매우 유사한 형태 및 크기의 group 을 형성한 점과 대응된다. 단지 group 의 형성 위치에서만 큰 차이를 보인다.

③ 좌표평면 상에서 x 성분과 y 성분 간의 연관관계(선형성)는 보이지 않는데, 이는 covariance matrix 상 Cov(X, Y)에 대응되는 non-diagonal 성분의 값이 0 에 매우 가깝기 때문이고 애초에 data sampling 때 , $\Sigma = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}$ 를 인자로 두었기 때문에 당연한 결과이다.

Note. multivariate_normal 함수를 쓰면 **[Fig 7-1]**과 같은 code 와 이를 통해 **[Fig 7-2]** 결과를 얻을 수 있다. Data 를 새로 sampling 했지만 **[Fig 6]**와 매우 유사한 분포를 보임을 알 수 있다.

```
x1 = np.random.multivariate_normal(mean = mu1, cov = cov, size = n)
x2 = np.random.multivariate_normal(mean = mu2, cov = cov, size = n)
x3 = np.random.multivariate_normal(mean = mu3, cov = cov, size = n)

###########################
plt.plot(x1[:,0], x1[:,1], label='mean = [0,10], cov = [[3,0],[0,3]]', marker = '.', color='red', linestyle="None")
plt.plot(x1[:,0].sum()/n, x1[:,1].sum()/n, marker = 'D', color='white', linestyle="None")
plt.plot(x2[:,0], x2[:,1], label='mean = [10,0], cov = [[3,0],[0,3]]', marker = '.', color='blue', linestyle="None")
plt.plot(x2[:,0].sum()/n, x2[:,1].sum()/n, marker = 'D', color='yellow', linestyle="None")
plt.plot(x3[:,0], x3[:,1], label='mean = [20,20], cov = [[3,0],[0,3]]', marker = '.', color='green', linestyle="None")
plt.plot(x3[:,0].sum()/n, x3[:,1].sum()/n, marker = 'D', color='black', linestyle="None")
###########################

plt.grid()
plt.legend(loc='upper left')
plt.show()
```

[Fig 7-1] Using multivariate_normal function



[Fig 7-2] Result from [Fig 7-1]