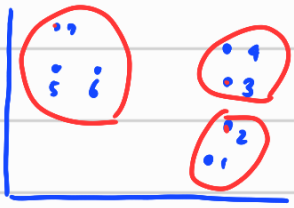
1. $x_1 = \begin{pmatrix}18\\5\end{pmatrix}$ $x_2 = \begin{pmatrix}20\\9\end{pmatrix}$ $x_3 = \begin{pmatrix}20\\14\end{pmatrix}$ $x_4 = \begin{pmatrix}20\\17\end{pmatrix}$ $x_5 = \begin{pmatrix}5\\15\end{pmatrix}$ $x_6 = \begin{pmatrix}9\\15\end{pmatrix}$ $x_7 = \begin{pmatrix}6\\20\end{pmatrix}$

$z_1 = (20, 9)^T$
$z_2 = (20, 14)^T$
$z_3 = (6, 20)^T$



$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

$J = 4.47 + 0 + 0 + 3 + 5.1 + 5.83 = 18.4$

새로운 대표값 $z_1 = \begin{pmatrix}19\\7\end{pmatrix}$ $z_2 = \begin{pmatrix}20\\15.5\end{pmatrix}$ $z_3 = \begin{pmatrix}6.67\\16.67\end{pmatrix}$

$J = 2.23 + 2.23 + 1.5 + 1.5 + 2.36 + 2.36 + 1.36 = 14.54$

예제 6-1 과 비교 했을 때 J의 값이 17.29 와 14.54 이므로 $z_1 = x_2$ $z_2 = x_3$ $z_3 = x_7$ 선택

## 3. k-means vs k-medoids 시간 복잡도 비교

k-means 의 시간 복잡도는 $O(N)$으로 데이터 수에 대하여 선형 시간 복잡도를 가짐

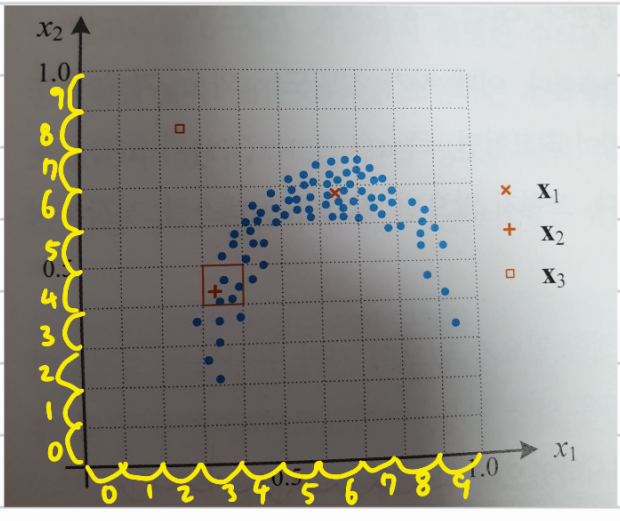
k-medoids 는 대표점에서 다른 샘플들 간의 거리 합이 최소화 되는 것을 대표하고 매번 update 를 한다.

따라서 샘플(군집) 에 따라 시간복잡도가 형성된다.

하지만 medoids 도 데이터 N에 따라도 영향을 받기 때문에 $O(KN)$ where K = number of sample

이므로 샘플의 개수 배만큼 시간을 더 쓸 것이다.

6.




6-1

$P(x_1=2, x_2=3)$ 결합 확률 $= \frac{1}{80}$

$P(x_1=2 \mid x_2=3) = \frac{P(x_1=2, x_2=3)}{P(x_2=3)} = \frac{\frac{1}{80}}{\frac{5}{80}} = \frac{1}{5}$

$P(x_2=3) = \frac{5}{80} = \frac{1}{16}$

6-2

$P(x_1=3, x_2=5) = \frac{3}{80}$

$P(x_2=5 \mid x_1=3) = \frac{P(x_1=3, x_2=5)}{P(x_1=3)} = \frac{\frac{3}{80}}{\frac{12}{80}} = \frac{3}{12} = \frac{1}{4}$

$P(x_1=3) = \frac{12}{80} = \frac{3}{20}$

9. $M = \begin{bmatrix} 3.5 \\ 3 \end{bmatrix}$ 공분산 행렬 : $\frac{1}{n}\sum (x-\mu)(x-\mu)^T \Rightarrow \begin{bmatrix} 2.25 & 1.125 \\ 1.125 & 0.75 \end{bmatrix}$

eigen value (고지) = $(\Sigma - I\lambda)u = 0$ ($\lambda$: eigen value)

$\therefore \lambda_1 = 2.852 \quad \lambda_2 = 0.147$

eigen vector = $[0.8817, -0.47179]$ or $[0.47179, 0.8817]$

```python
arr = np.array([[1,2],[2,2],[3,2],[3,3],[4,3],[4,4],[5,4],[6,4]])
mu = np.mean(arr,axis=0)
print('평균 : ',mu)
sum_ = 0
for i in arr:
    tmp1 = (i-mu).reshape(2,1)
    tmp2 = (i.T-mu.T).reshape(1,2)
    sum_ += np.dot(tmp1,tmp2)
cov = sum_/len(arr)
print('공분산행렬 : ',cov)
print('고유값 : ',np.linalg.eig(cov)[0])
print('고유벡터 : ',np.linalg.eig(cov)[1])
```

```
평균 :  [3.5 3. ]
공분산행렬 :  [[2.25  1.125]
 [1.125 0.75 ]]
고유값 :  [2.85208173 0.14791827]
고유벡터 :  [[ 0.8816746  -0.47185793]
 [ 0.47185793  0.8816746 ]]
```