4. (*Deterministic Annealing*) Considere um problema de *soft clustering* com cinco vetores de dados $\mathbf{x} = (x_1, x_2)$ e três centróides iniciais $\mathbf{y} = (y_1, y_2)$, definidos segundo a tabela a seguir. Considere também que a distância entre dois vetores é quadrática, ou seja, $d(\mathbf{x}, \mathbf{y}) = (x_1 - y_1)^2 + (x_2 - y_2)^2$.

$\mathbf{x}_1$	5	4
$\mathbf{x}_2$	4	5
$\mathbf{x}_3$	5	5
$\mathbf{x}_4$	-5	-4
$\mathbf{x}_5$	-4	-5
$\mathbf{y}_1$	0	0
$\mathbf{y}_2$	1	1
$\mathbf{y}_3$	-1	-1

a) Calcule a matriz de probabilidades $p(\mathbf{y}|\mathbf{x})$ que minimiza $J = D - TH$ com $T = 10$. Calcule também os valores dos centróides atualizados segundo esta matriz.

b) Repita o item (a) para $T = 0.1$ e comente sobre qual é a diferença.

a) Separando a tabela em uma matriz de dados e outra de centróides, temos as seguintes matrizes transpostas:

$$X = \begin{bmatrix} 5 & 4 & 5 & -5 & -4 \\ 4 & 5 & 5 & -4 & -5 \end{bmatrix} \text{ e } Y = \begin{bmatrix} 0 & 1 & -1 \\ 0 & 1 & -1 \end{bmatrix}$$

Para $T = 10$, calculamos a matriz $P_{Y|X}$ como

$$e \begin{bmatrix} e^{-\frac{\|x_1-y_1\|^2}{10}} & e^{-\frac{\|x_2-y_1\|^2}{10}} & e^{-\frac{\|x_3-y_1\|^2}{10}} & e^{-\frac{\|x_4-y_1\|^2}{10}} & e^{-\frac{\|x_5-y_1\|^2}{10}} \\ e^{-\frac{\|x_1-y_2\|^2}{10}} & e^{-\frac{\|x_2-y_2\|^2}{10}} & e^{-\frac{\|x_3-y_2\|^2}{10}} & e^{-\frac{\|x_4-y_2\|^2}{10}} & e^{-\frac{\|x_5-y_2\|^2}{10}} \\ e^{-\frac{\|x_1-y_3\|^2}{10}} & e^{-\frac{\|x_2-y_3\|^2}{10}} & e^{-\frac{\|x_3-y_3\|^2}{10}} & e^{-\frac{\|x_4-y_3\|^2}{10}} & e^{-\frac{\|x_5-y_3\|^2}{10}} \end{bmatrix} = \begin{bmatrix} 0{,}016 & 0{,}016 & 0{,}006 & 0{,}016 & 0{,}016 \\ 0{,}082 & 0{,}082 & 0{,}041 & 0{,}002 & 0{,}002 \\ 0{,}002 & 0{,}002 & 0{,}001 & 0{,}082 & 0{,}082 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 0{,}1 & 0{,}1 & 0{,}04 & 0{,}1 & 0{,}1 \end{bmatrix}$$

Somatório das colunas

Ao normalizar cada coluna com o valor de sua soma, temos:

$$P_{Y|X} = \begin{bmatrix} 0{,}16 & 0{,}16 & 0{,}14 & 0{,}16 & 0{,}16 \\ 0{,}81 & 0{,}81 & 0{,}84 & 0{,}02 & 0{,}02 \\ 0{,}03 & 0{,}03 & 0{,}02 & 0{,}82 & 0{,}82 \end{bmatrix}$$

Para calcular a nova posição dos centróides, temos:

$$y_K = \frac{\sum_x X \cdot P_{K|x}}{\sum_x P_{K|x}} \longrightarrow Y = \begin{bmatrix} 0{,}9 & 4{,}5 & -4{,}2 \\ 0{,}9 & 4{,}5 & -4{,}2 \end{bmatrix}$$

Os cálculos foram feitos utilizando a biblioteca de Python 3 "Numpy" e foram feitos os seguintes comandos:

```
Boltzmann = lambda x,y,T: np.exp(-np.sum(np.power(x[np.newaxis,:,:]-y[:,np.newaxis,:],2),axis=2)/T)
```

Variáveis

Executar a subtração entre todos os vetores de X e de Y de forma combinatória.

Pyx = Boltzmann(X,Y,10)/np.sum(Boltzmann(X,Y,10),axis=1)

newY = np.sum(X[np.newaxis,:,:]*Pyx[:,:,np.newaxis],axis=1)/np.sum(Pyx[:,:,np.newaxis],axis=1)

multiplica todos os vetores de X pelos elementos de cada linha de $P_{Y|X}$, para todas as linhas.

As matrizes X e Y foram definidas da forma transposta. O Log pode ser visto abaixo:

```
>>> X
array([[ 5,  4],
       [ 4,  5],
       [ 5,  5],
       [-5, -4],
       [-4, -5]])
>>> Y
array([[ 0,  0],
       [ 1,  1],
       [-1, -1]])
>>> Boltzmann = lambda x,y,T: np.exp(-np.sum(np.power(x[np.newaxis,:,:]-y[:,np.newaxis,:],2),axis=2)/T)
>>> Pyx = Boltzmann(X,Y,10)/np.sum(Boltzmann(X,Y,10),axis=0)
>>> newY = np.sum(X[np.newaxis,:,:]*Pyx[:,:,np.newaxis],axis=1)/np.sum(Pyx[:,:,np.newaxis],axis=1)
>>> newY
array([[ 0.87652431,  0.87652431],
       [ 4.50887063,  4.50887063],
       [-4.17568114, -4.17568114]])
>>> 
```

Repetindo a mesma operação para T=0,1, temos os novos valores de Y:

$$Y = \begin{bmatrix} 2\times10^{-7} & 4{,}\bar{6} & -4{,}5 \\ 2\times10^{-7} & 4{,}\bar{6} & -4{,}5 \end{bmatrix}$$

Temos que o vetor $\vec{y}_0$ teve uma mudança insignificante, enquanto os vetores $\vec{y}_1$ e $\vec{y}_2$ foram respectivamente para o centro de massa dos três primeiros e dos três últimos dados em X. Como essa solução não melhorar, se esse for um mínimo local os centróides ficam presos nessas dimensões.

4. (*Deterministic Annealing*) Considere um conjunto de dados $\mathbf{X}$ composto por quatro vetores equiprováveis e com coordenadas $(0,0)$, $(0,2)$, $(2,2)$ e $(2,0)$. Considere uma partição suave do $\mathbb{R}^2$ realizada através de dois centróides $\mathbf{y}_1$ e $\mathbf{y}_2$. As distâncias quadráticas entre cada vetor $\mathbf{x}$ e cada vetor $\mathbf{y}$ são dadas a seguir:

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
$\mathbf{y}_1$	1	5	5	1
$\mathbf{y}_2$	5	1	1	5

$e^{-\frac{d}{T}}\dots$ d_{xy}

a) Considerando $T = 5$, calcule a matriz de probabilidades condicionais $p_{\mathbf{Y}|\mathbf{X}}$ que minimiza $J = D - TH = \sum_x p_{\mathbf{X}}(x) \sum_y p_{\mathbf{Y}|\mathbf{X}}(y|x) + T \sum_x p_{\mathbf{X}}(x) \sum_y p_{\mathbf{Y}|\mathbf{X}}(y|x) \log p_{\mathbf{Y}|\mathbf{X}}(y|x)$.

b) Também considerando $T = 5$, calcule vetores $\mathbf{y}_1$ e $\mathbf{y}_2$ atualizados. Compare o erro médio quadrático D_2, associado aos vetores $\mathbf{y}_1$ e $\mathbf{y}_2$ atualizados, com o erro médio quadrático D_1, associado aos vetores $\mathbf{y}_1$ e $\mathbf{y}_2$ anteriores à atualização.

Os mesmos cálculos são feitos em python. Temos:

```
>>> dist = np.array([[1,5,5,1],[5,1,1,5]])
>>> x = np.array([[0,0],[0,2],[2,2],[2,0]])
>>> Pyx = np.exp(-dist/5)/np.sum(np.exp(-dist/5),axis=0)
>>> mewY = np.sum(x[np.newaxis,:,:]*Pyx[:,:,np.newaxis],axis=1)/np.sum(Pyx[:,:,np.newaxis],axis=1)
>>> Pyx
array([[0.68997448, 0.31002552, 0.31002552, 0.68997448],
       [0.31002552, 0.68997448, 0.68997448, 0.31002552]])
>>> newY = mewY
>>> newY
array([[1.        , 0.62005104],
       [1.        , 1.37994896]])
```

Assim:

a) A matriz $P_{Y|X} = \begin{bmatrix} 0,68 & 0,31 & 0,31 & 0,69 \\ 0,32 & 0,69 & 0,69 & 0,31 \end{bmatrix}$

b) Os vetores dos centróides atualizados são:

$$Y = \begin{bmatrix} 1 & 0,62 \\ 1 & 1,37 \end{bmatrix}$$

Para calcular o erro médio quadrático $D_1 = \sum_x p_x(x) \sum_Y P_{Y|X}(y|x) d_{xy}$, temos

$\begin{smallmatrix} 1 & 5 & 5 & 1 \\ 5 & 1 & 1 & 5 \end{smallmatrix}$

$$D_1 = \frac{1}{4} \cdot \left(0,68.1 + 0,32.5\right) + \frac{1}{4} \cdot \left(0,32.5 + 0,68.1\right) + \frac{1}{4}\left(0,32.5 + 0,68.1\right) + \frac{1}{4}\left(0,68.1 + 0,32.5\right)$$

$$D_1 = 0,68 + 5.0,32 = 2,28$$

Para os novos valores de Y, podemos recalcular a tabela de distâncias

```
>>> newY
array([[1.        , 0.62005104],
       [1.        , 1.37994896]])
>>> np.sum(Pyx,axis=0)
array([1., 1., 1., 1.])
>>> 0.68+5*0.32
2.2800000000000002
>>> newY[:,np.newaxis,:]
KeyboardInterrupt
>>> x
array([[0, 0],
       [0, 2],
       [2, 2],
       [2, 0]])
>>> x[np.newaxis,:,:]-newY[:,np.newaxis,:]
array([[[-1.        , -0.62005104],
        [-1.        ,  1.37994896],
        [ 1.        ,  1.37994896],
        [ 1.        , -0.62005104]],

       [[-1.        , -1.37994896],
        [-1.        ,  0.62005104],
        [ 1.        ,  0.62005104],
        [ 1.        , -1.37994896]]])
>>> x.shape
(4, 2)
>>> np.sum(np.power(x[np.newaxis,:,:]-newY[:,np.newaxis,:],2),axis=2)
array([[1.38446329, 2.90425914, 2.90425914, 1.38446329],
       [2.90425914, 1.38446329, 1.38446329, 2.90425914]])
```

Repetimos o cálculo para D_2 e encontramos $D_2 = 1,85$. Observamos que houve uma diminuição do erro médio quadrático.

```
>>> np.sum(np.power(x[np.newaxis,:,:]-newY[:,np.newaxis,:],2),axis=2)
array([[1.38446329, 2.90425914, 2.90425914, 1.38446329],
       [2.90425914, 1.38446329, 1.38446329, 2.90425914]])
>>> D2 = np.sum(np.sum(Pyx*np.sum(np.power(x[np.newaxis,:,:]-newY[:,np.newaxis,:],2),axis=2),axis=0)/4)
>>> D2
1.8556387860811778
>>>
```