

Universidade Federal do Rio de Janeiro  
Instituto Alberto Luiz Coimbra de  
Pós-Graduação e Pesquisa de Engenharia



Programa de Engenharia de Sistemas e  
Computação

CPS863 - Aprendizado de Máquina  
Prof. Dr. Edmundo de Souza e Silva  
(PESC/COPPE/UFRJ)

*Lista de Exercícios 5*

Luiz Henrique Souza Caldas  
email: lhscaldas@cos.ufrj.br

19 de novembro de 2024

## Questão 1 - HMM

Considere o robô da lista anterior, que pode se mover pelos quadrados da figura abaixo.

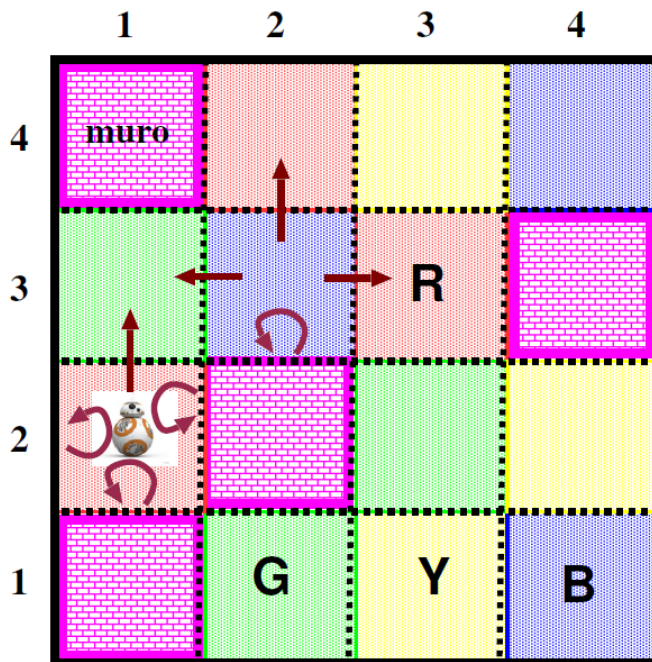


Figura 1: Robô andando por um ambiente

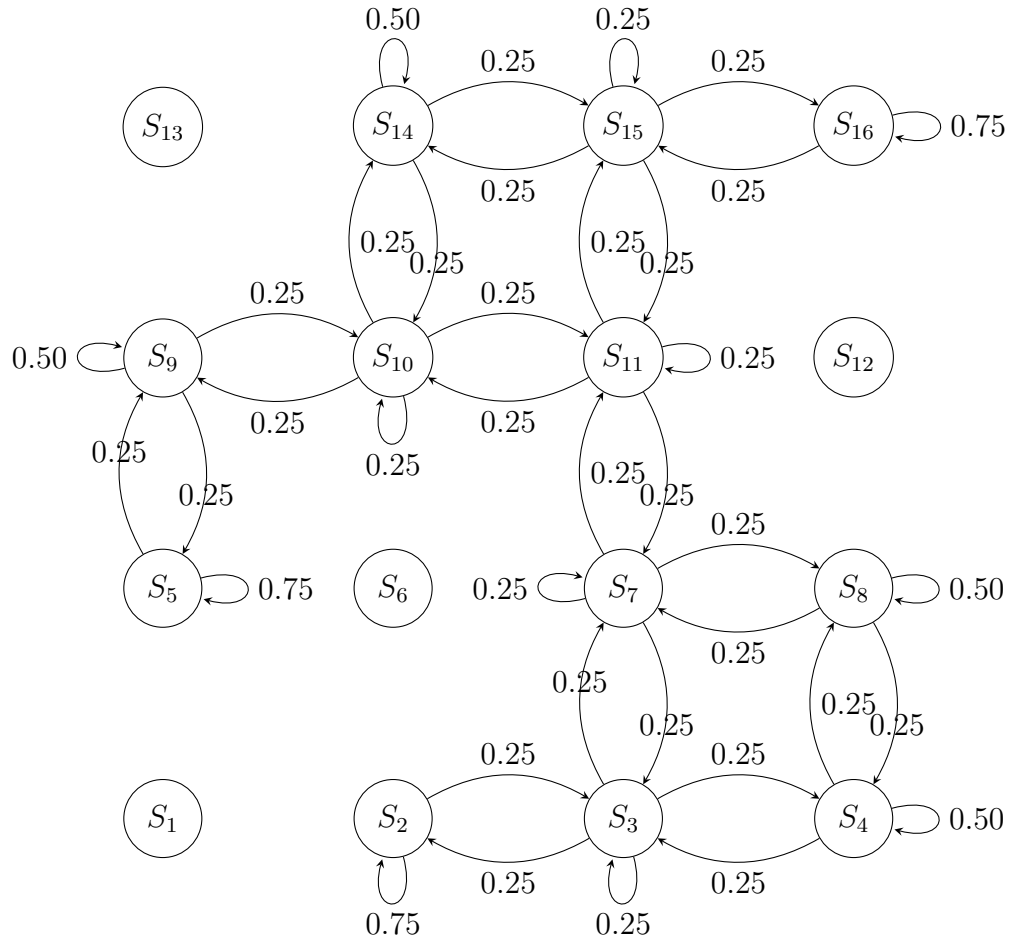
Para tentar melhorar a previsibilidade de se detectar a posição do robô da Figura 1 sensores são colocados no ambiente onde o robô circula. Há 4 tipos de sensores (**R**, **B**, **Y**, **G**), conforme mostrado na Figura 1. Quando o robô está em qualquer um dos quadrados, o respectivo sensor emite um sinal (para um receptor) com a letra igual ao tipo do sensor. Entretanto, os sensores não são perfeitos e podem emitir um sinal errado com probabilidade 0.1. Por exemplo, quando o robô está num dos quadrados azuis, emite um sinal **b** com probabilidade 0.9, ou um dos restantes sinais **r** ou **y** ou **g**, com probabilidade 0.1/3. Como outro exemplo, suponha que o robô esteja na posição inicial conforme mostrado na Figura 1. Em 3 unidades de tempo, uma possível sequência de sinais recebidos poderiam ser **r g b**, se o robô for para norte e depois para leste. Entretanto, mesmo com o mesmo movimento, os sinais recebidos poderiam ser também **r g g** ou **b b b**, etc.

Seu objetivo é determinar a posição do robô, a partir dos sinais recebidos dos sensores.

- Explique como você fará uma HMM que possa permitir prever a posição do robô a partir dos sinais recebidos.

Resposta:

1. **Estados:** Assim como no problema da lista anterior, os estados são as posições possíveis do robô, com a diferença de que agora os estados não podem ser diretamente observáveis. Para facilitar a notação, os estados foram numerados em ordem crescente da esquerda para a direita e de cima para baixo. Assim, o antigo estado (1,1) passou a se chamar  $S_1$ , (1,2) passou a se chamar  $S_2$  e assim por diante. O modelo da Cadeia de Markov pode ser visto na figura 2.
2. **Observações:** As observações são as emissões dos sensores, formadas pelos símbolos **R**, **B**, **Y**, **G**.
3. **Matriz de transição:** A matriz de transição é a mesma da lista anterior, com as probabilidades de transição entre os estados. Nesta lista foi utilizada a letra  $A$  para representar a matriz de transição, seguindo a simbologia de Rabiner (1989) [1]. A matriz  $A$  é mostrada na tabela 1.
4. **Matriz de emissão:** A matriz de emissão é a probabilidade de cada estado emitir cada uma das observações. Cada estado tem uma probabilidade de 0.9 de emitir a cor dele mesmo e 0.1 de emitir qualquer outra cor ( $0.1/3 \approx 0.0333$  para cada cor). Os estados proibidos (rosa) possuem probabilidade de emissão 0 para todos os símbolos. A matriz de emissão é mostrada na tabela 2.
5. **Probabilidade inicial:** O vetor de probabilidades iniciais (renomeado para  $\pi$  pelo mesmo motivo da matriz de transição) é o mesmo da lista anterior, com a probabilidade 1 do robô começar no estado  $S_5$  (posição 2,1)  $\pi_5 = 1$  e nula para os demais estados.



**Figura 2:** Cadeia de Markov representando o movimento do robô no ambiente.

	$S_1$	$S_2$	$S_3$	$S_4$	$S_5$	$S_6$	$S_7$	$S_8$	$S_9$	$S_{10}$	$S_{11}$	$S_{12}$	$S_{13}$	$S_{14}$	$S_{15}$	$S_{16}$
$S_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$S_2$	0	0.75	0.25	0	0	0	0	0	0	0	0	0	0	0	0	0
$S_3$	0	0.25	0.25	0.25	0	0	0.25	0	0	0	0	0	0	0	0	0
$S_4$	0	0	0.25	0.50	0	0	0	0.25	0	0	0	0	0	0	0	0
$S_5$	0	0	0	0	0.75	0	0	0	0.25	0	0	0	0	0	0	0
$S_6$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$S_7$	0	0	0.25	0	0	0	0.25	0.25	0	0	0.25	0	0	0	0	0
$S_8$	0	0	0	0.25	0	0	0.25	0.50	0	0	0	0	0	0	0	0
$S_9$	0	0	0	0	0.25	0	0	0	0.50	0.25	0	0	0	0	0	0
$S_{10}$	0	0	0	0	0	0	0	0	0.25	0.25	0.25	0	0	0.25	0	0
$S_{11}$	0	0	0	0	0	0	0.25	0	0	0.25	0.25	0	0	0	0.25	0
$S_{12}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$S_{13}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$S_{14}$	0	0	0	0	0	0	0	0	0	0.25	0	0	0	0.50	0.25	0
$S_{15}$	0	0	0	0	0	0	0	0	0	0	0.25	0	0	0.25	0.25	0.25
$S_{16}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.25	0.75

**Tabela 1:** Matriz de Transição  $A$

	$S_1$	$S_2$	$S_3$	$S_4$	$S_5$	$S_6$	$S_7$	$S_8$	$S_9$	$S_{10}$	$S_{11}$	$S_{12}$	$S_{13}$	$S_{14}$	$S_{15}$	$S_{16}$
R (Vermelho)	0	0.0333	0.0333	0.0333	0.9000	0	0.0333	0.0333	0.0333	0.0333	0.9000	0	0	0.9000	0.0333	0.0333
B (Azul)	0	0.0333	0.0333	0.9000	0.0333	0	0.0333	0.0333	0.0333	0.9000	0.0333	0	0	0.0333	0.0333	0.9000
Y (Amarelo)	0	0.0333	0.9000	0.0333	0.0333	0	0.0333	0.9000	0.0333	0.0333	0.0333	0	0	0.0333	0.9000	0.0333
G (Verde)	0	0.9000	0.0333	0.0333	0.0333	0	0.9000	0.0333	0.9000	0.0333	0.0333	0	0	0.0333	0.0333	0.0333

**Tabela 2:** Matriz de Emissão  $B$

- Suponha que o receptor de sinais tenha recebido a sequência r r y r y r b g b r y y g b. Qual a probabilidade desta sequência ocorrer? Explique e implemente o algoritmo necessário para responder a pergunta.

Resposta:

Assim como no problema 1 de [1], devemos utilizar a etapa forward do algoritmo forward-backward para calcular a probabilidade da sequência de observações. Esta etapa pode ser dividida em 3 passos:

- Inicialização: a variável  $\alpha_1(i)$  é calculada para cada estado  $i$  como a probabilidade de estar no estado  $i$  e emitir a observação  $O_1$ .

$$\alpha_1(i) = \pi_i b_i(O_1) \quad (1)$$

- Indução: a variável  $\alpha_t(i)$  é calculada para cada estado  $i$  como a probabilidade de estar no estado  $i$  e emitir a observação  $O_t$  dado que a sequência de observações até o tempo  $t - 1$  foi observada.

$$\alpha_t(i) = P(O_1, O_2, \dots, O_t, q_t = S_i | \lambda) = \sum_{j=1}^N \alpha_{t-1}(j) a_{ji} b_i(O_t) \quad (2)$$

- Terminio: a probabilidade total da sequência de observações é dada por:

$$P(O | \lambda) = \sum_{i=1}^N \alpha_T(i) \quad (3)$$

onde  $\lambda$  é o modelo,  $N$  é o número de estados,  $\pi_i$  é a probabilidade inicial de estar no estado  $i$ ,  $a_{ji}$  é a probabilidade de transição do estado  $j$  para o estado  $i$  e  $b_i(O_t)$  é a probabilidade de emitir a observação  $O_t$  no estado  $i$ .

Para isso, foi implementada em python a classe HMM, que recebe as matrizes  $A$  e  $B$  ao ser inicializada e contém os métodos forward para calcular a probabilidades  $\alpha$  de uma dada sequência e método P\_obs, que calcula a probabilidade total daquela observação, para um dado alpha. O código fonte da classe HMM pode ser visto no código 1.

**Código 1:** Classe HMM

```

1 class HMM:
2     def __init__(self, A, B, pi):
3         """

```

```

4         Inicializa o modelo HMM.
5
6         :param A: Matriz de transição de estados (N x N)
7         :param B: Matriz de emissão de observações (N x M)
8         :param pi: Vetor de probabilidades iniciais (1 x N)
9         """
10        self.A = A # Matriz de transição
11        self.B = B # Matriz de emissão
12        self.pi = pi # Probabilidades iniciais
13
14    def forward(self, O):
15        """
16        Calcula o alpha para todos os estados e tempos para uma
17        sequência O.
18
19        :param O: Sequência de observações (lista de índices das
20        observações)
21        :return: Matriz alpha (T x N)
22        """
23        N = len(self.pi) # Número de estados
24        T = len(O) # Comprimento da sequência de observações
25
26        # Inicialização
27        alpha = np.zeros((T, N))
28        alpha[0, :] = self.pi * self.B[:, O[0]]
29
30        # Recursão
31        for t in range(1, T):
32            for j in range(N):
33                alpha[t, j] = np.sum(alpha[t - 1, :] * self.A[:, j
34                ]) * self.B[j, O[t]]
35
36        return alpha
37
38    def P_obs(self, alpha):
39        """
40        Calcula a probabilidade da sequência de observações, dado
41        o modelo.
42
43        :param alpha: Matriz alpha (T x N) calculada pelo método
44        forward
45        :return: Probabilidade da sequência de observações
46        """
47        return np.sum(alpha[-1, :])

```

Resposta (continuação):

O resultado obtido para a sequência *rryryrbgbryygb* foi:

$$P(O|\lambda) = 4.3021 \times 10^{-10} \quad (4)$$

A probabilidade obtida foi um valor muito pequeno, porém esse resultado é esperado. Isso ocorre pois:

- O robô possui muitas possibilidades de movimento;
- A sequência de observações é relativamente longa;
- A probabilidade de emissão de uma cor diferente da esperada para um estado é de 0.0333 para cada cor.

Juntos, esses fatores contribuem para que o resultado seja um valor muito pequeno.

- **Repita o item anterior para a sequência *r b y r g r b g b r y y g b*. (Obviamente não precisa reimplementar o algoritmo!)**

Resposta:

Utilizando o mesmo código do item anterior, o resultado obtido para a sequência *rbyrgrbgbryygb* foi:

$$P(O|\lambda) = 1.0925 \times 10^{-10} \quad (5)$$

O resultado obtido foi cerca de 4 vezes menor que o obtido na sequência anterior, mostrando que essa sequência é menos provável de ocorrer.

- **Para a primeira sequência acima, qual o quadrado mais provável onde estará o robô na última posição (isto é, o quadrado de onde foi emitido o último sinal)? Explique e implemente o algoritmo necessário.**

Resposta:

A solução para este item esta na parte inicial do problema 2 de [1], onde a variável  $\gamma = P(q_t = S_i | O, \lambda)$  é definida como a probabilidade de estar no estado  $S_i$  no tempo  $t$  dado o modelo  $\lambda$  e a sequência de observações  $O$ , sendo dada por:

$$\gamma_t(i) = P(q_t = S_i | O, \lambda) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^N \alpha_t(j)\beta_t(j)} \quad (6)$$

onde  $\beta_t(i)$  é a probabilidade de observar a sequência  $O_{t+1}, O_{t+2}, \dots, O_T$  dado que o sistema está no estado  $S_i$  no tempo  $t$ , sendo calculada na etapa backward do algoritmo forward-backward:

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \dots, O_T | q_t = S_i, \lambda) = \sum_{j=1}^N a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \quad (7)$$

Para calcular o estado final mais provável, foram implementados os métodos backward e gamma, que calculam as matrizes  $\beta$  e  $\gamma$ , respectivamente. Além disso, foi implementado o método end\_state, que recebe a matriz gamma e retorna o estado mais provável na última observação e o valor da probabilidade para aquele estado ser o último. O código fonte dos métodos backward, gamma e end\_state pode ser visto no código 2.

**Código 2:** backward, gamma e end\_state

```
1 def backward(self, O):
2     """
3     Calcula o beta para todos os estados e tempos para uma sequência O
4     .
5     :param O: Sequência de observações (lista de índices das observações)
6     :return: Matriz beta (T x N)
7     """
8     N = len(self.pi)
9     T = len(O)
10
11     # Inicialização
12     beta = np.zeros((T, N))
13     beta[-1, :] = 1
14
15     # Recursão
16     for t in range(T - 2, -1, -1):
17         for i in range(N):
18             beta[t, i] = np.sum(self.A[i, :] * self.B[:, O[t + 1]] *
19                                 beta[t + 1, :])
20
21     return beta
22
23 def gamma(self, alpha, beta):
24     """
25     Calcula a matriz gamma para todos os estados e tempos para uma
```



```

25         sequência 0.
26
27         :param alpha: Matriz alpha (T x N) calculada pelo método forward
28         :param beta: Matriz beta (T x N) calculada pelo método backward
29         :return: Matriz gamma (T x N)
30         """
31         return alpha * beta / np.sum(alpha * beta, axis=1).reshape(-1, 1)
32
33     def end_state(self, gamma):
34         """
35         Calcula o estado mais provável na última observação.
36
37         :param gamma: Matriz gamma (T x N) calculada pelo método gamma
38         :return: Estado mais provável na última observação
39         """
40         end_state = np.argmax(gamma[-1, :])
41         P_end_state = gamma[-1, end_state]
42         state_map = {0: 'S1', 1: 'S2', 2: 'S3', 3: 'S4', 4: 'S5', 5: 'S6', 6: 'S7', 7: 'S8', 8: 'S9', 9: 'S10', 10: 'S11', 11: 'S12', 12: 'S13', 13: 'S14', 14: 'S15', 15: 'S16'}
43         return state_map[end_state], P_end_state

```

Resposta (continuação):

Executando o código acima, o resultado obtido para a sequência *rryryrbgbryygb* foi que o estado mais provável na última observação é o estado  $S_{11}$  (X=3, Y=3), com probabilidade de 0.6055.

- Para a segunda sequência acima, qual o quadrado mais provável onde estará o robô?

Resposta:

Executando o mesmo código do item anterior, o resultado obtido para a sequência *rbyrgrbgbyygb* foi que o estado mais provável na última observação também é o estado  $S_{11}$  (X=3, Y=3), porém com probabilidade de 0.6092, muito próxima da probabilidade obtida na sequência anterior.

- Para a primeira sequência acima, qual o caminho mais provável percorrido pelo robô? Explique o algoritmo usado, mas não precisa implementar. Use uma biblioteca de Python ou outra linguagem preferida.

### Resposta:

Com base no problema 2 de [1], o caminho mais provável percorrido pelo robô é obtido através do algoritmo de Viterbi, que calcula a sequência de estados mais provável para uma dada sequência de observações. O algoritmo de Viterbi é baseado no cálculo da variável  $\delta_t(i)$ , que é a probabilidade da sequência mais provável de estados até o tempo  $t$  e que termina no estado  $i$ , sendo dada por:

$$\delta_t(i) = \max_{q_1, q_2, \dots, q_t} [P(q_1, q_2, \dots, q_t = S_i, O_1, O_2, \dots, O_t | \lambda)] \quad (8)$$

A cada passo também é calculada a variável  $\psi_t(i)$ , que é o estado anterior mais provável para o estado  $i$  no tempo  $t$ .

- Inicialização: a variável  $\delta_1(i)$  é calculada para cada estado  $i$  como a probabilidade de estar no estado  $i$  e emitir a observação  $O_1$ .

$$\delta_1(i) = \pi_i b_i(O_1) \quad (9)$$

e

$$\psi_1(i) = 0 \quad (10)$$

- Recursão: a variável  $\delta_t(i)$  é calculada para cada estado  $i$  como a probabilidade de estar no estado  $i$  e emitir a observação  $O_t$  dado que a sequência de observações até o tempo  $t - 1$  foi observada. A variável  $\psi_t(i)$  é calculada como o estado anterior mais provável para o estado  $i$ .

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] b_j(O_t) \quad (11)$$

e

$$\psi_t(j) = \operatorname{argmax}_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] \quad (12)$$

- Finalização: O estado final mais provável e a sua probabilidade são calculados.

$$q_T^* = \operatorname{argmax}_{1 \leq i \leq N} \delta_T(i) \quad (13)$$

e

$$P^* = \max_{1 \leq i \leq N} \delta_T(i) \quad (14)$$

- Retrocesso: O caminho mais provável é obtido retrocedendo a partir do estado final mais provável, utilizando os valores de  $\psi_t(i)$  calculados na etapa de recursão.

$$q_t^* = \psi_{t+1}(q_{t+1}^*) \quad (15)$$

Foi implementado na classe HMM o método viterbi, que recebe a sequência de observações e retorna o caminho mais provável percorrido pelo robô. O código fonte do método viterbi pode ser visto no código 3.

### Código 3: viterbi

```
1 def viterbi(self, 0):
2     # inicialização
3     N = len(self.pi)
4     T = len(O)
5     delta = np.zeros((T, N))
6     psi = np.zeros((T, N), dtype=int)
7     delta[0, :] = self.pi * self.B[:, 0[0]]
8     psi[0, :] = 0
9
10    # recursão
11    for t in range(1, T):
12        for j in range(N):
13            delta[t, j] = np.max(delta[t - 1, :] * self.A
14                                 [:, j]) * self.B[j, 0[t]]
15            psi[t, j] = np.argmax(delta[t - 1, :] * self.A
16                                 [:, j])
17
18    # finalização
19    P_star = np.max(delta[-1, :])
20    q_star = np.argmax(delta[-1, :])
21
22    # reconstrução do caminho
23    path = [q_star]
24    for t in range(T - 1, 0, -1):
25        q_star = psi[t, q_star]
26        path.insert(0, q_star)
27
28    state_map = {0: 'S1', 1: 'S2', 2: 'S3', 3: 'S4', 4: 'S5', 5: 'S6', 6: 'S7', 7: 'S8', 8: 'S9', 9: 'S10', 10: 'S11', 11: 'S12', 12: 'S13', 13: 'S14', 14: 'S15', 15: 'S16'}
29    path = [state_map[state] for state in path]
30    return path, P_star
```

#### Resposta (continuação):

Executando o código acima, o resultado obtido para a sequência *rryryrbgbryygb* foi que a sequência de estados mais provável percorrida pelo robô foi: ['S5', 'S5', 'S5', 'S5', 'S5', 'S5', 'S9', 'S10', 'S11', 'S7', 'S8', 'S8', 'S7', 'S11'], com probabilidade de  $8.3918 \times 10^{-11}$ .

Apesar de não ter sido pedido no item, o caminho mais provável para a segunda sequência, *rbyrgrbgbryygb*, foi: ['S5', 'S9', 'S5', 'S5', 'S5', 'S5', 'S9', 'S10', 'S11', 'S7', 'S8', 'S8', 'S7', 'S11'], com probabilidade de  $9.3242 \times 10^{-12}$ .

## Questão 2 - Para exercitar o EM mais uma vez

Nesta tarefa, você usará o algoritmo Expectation-Maximization (EM) para inferir nota de filmes em um conjunto de dados. As notas são de 0.0 - 10.0 com uma casa decimal. O conjunto de dados contém as notas de clientes para quatro filmes de diferentes categorias (Sci-Fi e Romance). Os clientes são divididos em três classes com base em suas preferências, mas também é desconhecida a classe do cliente.

### 1. Explique as equações usadas para resolver o problema.

Resposta:

Podemos utilizar uma Mistura de Gaussianas Multivariadas (GMM) para agrupar os clientes em 3 classes. A função de verossimilhança é dada por:

$$p(x_i|C_k, \Theta) = \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(x_i - \mu_k)^T \Sigma_k^{-1}(x_i - \mu_k)\right), \quad (16)$$

onde  $x_i$  é o vetor de notas do i-ésimo cliente,  $C_k$  é a k-ésima classe,  $\Theta = \{\mu_k, \Sigma_k\}$  é o conjunto de parâmetros do modelo,  $\mu_k$  é a média da k-ésima classe e  $\Sigma_k$  é a matriz de covariância da k-ésima classe. A probabilidade de um cliente pertencer a uma classe é dada por:

$$p(C_k|x_i, \Theta) = \frac{P(x_i|C_k, \Theta)P(C_k)}{\sum_{j=1}^K p(x_i|C_j, \Theta)p(C_j)}, \quad (17)$$

onde  $P(C_k) = \pi_k$  é a probabilidade a priori de um cliente pertencer a uma classe e  $K$  é o número de classes.

O algoritmo EM é utilizado para encontrar os parâmetros do modelo que maximizam a verossimilhança e pode ser dividido em duas etapas:

- **Expectation (E-step):** Calcula a probabilidade de um cliente pertencer a uma classe (chamada responsabilidade ( $r_{ik}$ )), dada as notas que ele deu e os parâmetros do modelo.

$$r_{ik} = \frac{P(x_i|C_k, \Theta)P(C_k)}{\sum_{j=1}^K p(x_i|C_j, \Theta)p(C_j)}. \quad (18)$$

Resposta (continuação):

- **Maximization (M-step):** Atualiza os parâmetros do modelo com base nas responsabilidades calculadas no passo anterior.

$$\mu_k = \frac{\sum_{i=1}^N r_{ik} x_i}{\sum_{i=1}^N r_{ik}}, \quad (19)$$

$$\Sigma_k = \frac{\sum_{i=1}^N r_{ik} (x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{i=1}^N r_{ik}}, \quad (20)$$

$$\pi_k = \frac{\sum_{i=1}^N r_{ik}}{N}. \quad (21)$$

Para lidar com os clientes com valores faltantes, o calculo da verossimilhança é feito considerando apenas as notas que o cliente deu. Consequentemente, o calculo da responsabilidade no E-step e os calculos da média e da matriz de covariância no M-step deverão ser modificados para considerar isso:

- **Expectation (E-step):** A fórmula da responsabilidade em si não muda:

$$r_{ik} = \frac{P(x_i | C_k, \Theta) P(C_k)}{\sum_{j=1}^K p(x_i | C_j, \Theta) P(C_j)}, \quad (22)$$

Porém agora a verossimilhança é calculada considerando apenas as notas que o cliente deu (observadas):

$$p(x_i | C_k, \Theta) = \frac{1}{(2\pi)^{d_{obs}/2} |\Sigma_k^{obs}|^{1/2}} \exp \left( -\frac{1}{2} (x_i^{obs} - \mu_k^{obs})^T (\Sigma_k^{obs})^{-1} (x_i^{obs} - \mu_k^{obs}) \right). \quad (23)$$

- **Maximization (M-step):** A média e a matriz de covariância são atualizadas considerando apenas as notas que o cliente deu:

$$\mu_{kj} = \frac{\sum_{i=1}^N r_{ik} x_{ij}}{\sum_{i=1}^N r_{ik}}, \quad (24)$$

para  $x_{ij}$  observados.

A matriz de covariância é atualizada de forma similar:

$$\Sigma_k = \frac{\sum_{i=1}^N r_{ik} \cdot ((x_i - \mu_k)(x_i - \mu_k)^T \odot M_i)}{\sum_{i=1}^N r_{ik}}, \quad (25)$$

onde  $M_i$  é uma matriz diagonal que indica quais notas do cliente  $i$  são observadas.

A probabilidade a priori é atualizada igual era antes.

2. Baseado no item anterior, explique a sua implementação, incluindo as suas escolhas para a inicialização do código.

Resposta:

Baseado no item anterior, foi implementado o código 4 em Python.

**Código 4:** Implementação do algoritmo EM para clusterização de clientes.

```
1 import numpy as np
2 import pandas as pd
3
4 def em_algorithm(data, num_classes, max_iters=1000, tol=1e-6):
5     """
6     Algoritmo EM otimizado com vetorização para lidar com dados faltantes.
7
8     Args:
9         data (pd.DataFrame): Dados de entrada (clientes x características)
10            , com NaN para valores faltantes.
11         num_classes (int): Número de classes (clusters) no modelo.
12         max_iters (int): Número máximo de iterações.
13         tol (float): Tolerância para critério de convergência.
14
15     Returns:
16         dict: Parâmetros estimados (médias, covariâncias e probabilidades
17            a priori).
18         pd.DataFrame: Responsabilidades finais para cada cliente e classe.
19     """
20     n, d = data.shape
21     np.random.seed(42)
22
23     # Inicializar parâmetros
24     means = np.random.rand(num_classes, d) # Médias aleatórias
25     covariances = [np.eye(d) for _ in range(num_classes)] # Covariâncias
26     # como matrizes identidade
27     priors = np.ones(num_classes) / num_classes # Probabilidades a priori
28     # uniformes
29
30     # Matriz de responsabilidades
31     responsibilities = np.zeros((n, num_classes))
32
33     # Converter o DataFrame em matriz NumPy para maior eficiência
34     data_np = data.to_numpy()
35
36     for iteration in range(max_iters):
37         # --- E-step: Calcular responsabilidades ---
38         for k in range(num_classes):
39             mean_k = means[k]
40             cov_k = covariances[k]
41             prior_k = priors[k]
42
43             for i in range(n):
44                 observed = ~np.isnan(data_np[i]) # Dimensões observadas
45                 x_obs = data_np[i, observed]
```

```

42         mean_obs = mean_k[observed]
43         cov_obs = cov_k[np.ix_(observed, observed)]
44
45         if len(x_obs) > 0:
46             diff = x_obs - mean_obs
47             likelihood = (
48                 np.exp(-0.5 * diff.T @ np.linalg.inv(cov_obs) @
49                     diff) /
50                 np.sqrt((2 * np.pi) ** len(x_obs) * np.linalg.det(
51                     cov_obs))
52             )
53         else:
54             likelihood = 1e-6
55
56         responsibilities[i, k] = prior_k * likelihood
57
58     # Normalizar responsabilidades
59     responsibilities /= responsibilities.sum(axis=1, keepdims=True)
60
61     # --- M-step: Atualizar parâmetros ---
62     Nk = responsibilities.sum(axis=0) # Soma total de
63     # responsabilidades para cada classe
64
65     # Atualizar médias
66     for k in range(num_classes):
67         weighted_sum = np.zeros(d)
68         for j in range(d):
69             observed = ~np.isnan(data_np[:, j])
70             weighted_sum[j] = np.dot(responsibilities[observed, k],
71                 data_np[observed, j])
72         means[k] = weighted_sum / Nk[k]
73
74     # Atualizar covariâncias
75     for k in range(num_classes):
76         cov_k = np.zeros((d, d))
77         for i in range(n):
78             observed = ~np.isnan(data_np[i])
79             x_obs = data_np[i, observed]
80             mean_obs = means[k, observed]
81             if len(x_obs) > 0:
82                 diff = x_obs - mean_obs
83                 cov_k[np.ix_(observed, observed)] += responsibilities[
84                     i, k] * np.outer(diff, diff)
85         covariances[k] = cov_k / Nk[k]
86
87     # Atualizar probabilidades a priori
88     priors = Nk / n
89
90     # Verificar convergência
91     if iteration > 0 and np.abs(responsibilities -
92         prev_responsibilities).max() < tol:
93         break
94
95     prev_responsibilities = responsibilities.copy()

```

```

90
91     return {
92         "means": means,
93         "covariances": covariances,
94         "priors": priors
95     }, pd.DataFrame(responsibilities, columns=[f"Class_{k+1}" for k in
96         range(num_classes)], index=data.index), iteration
97
98
99 if __name__ == "__main__":
100     # Exemplo de uso
101     data = pd.read_csv("lista_5/customer_ratings.csv")
102     data = data.drop(columns=["Customer Class"]) # Remover coluna
103         Customer Class
104     params, resps, its = em_algorithm(data, num_classes=3)
105
106     print(f"Convergência após {its+1} iterações.")
107     print("Médias:")
108     print(params["means"])
109     print("\nCovariâncias:")
110     print(params["covariances"])
111     print("\nProbabilidades a priori:")
112     print(params["priors"])
113
114     # Determinar a classe para cada cliente
115     class_assignments = resps.idxmax(axis=1)
116
117     # Contar quantos clientes foram alocados para cada classe
118     class_counts = class_assignments.value_counts()
119
120     print("\nQuantidade de usuários por classe:")
121     print(class_counts)
122
123     # Calcular a probabilidade de um cliente específico ser de uma classe
124     # específica
125     cliente_id = 0 # ID do cliente (linha no DataFrame original)
126     classe_k = "Class_3" # Nome da classe (coluna no DataFrame de
127         responsabilidades)
128
129     probabilidade = resps.loc[cliente_id, classe_k]
130
131     print(f"A probabilidade do cliente {cliente_id+1} pertencer à classe {
132         classe_k} é {probabilidade:.4f}")

```

3. Quantas iterações foram necessárias para resolver o problema? Qual o teste de parada utilizado?



Resposta:

Foram necessárias 845 iterações. O teste de parada utilizado foi a maior diferença entre as responsabilidades atuais e anteriores ser menor que  $1e - 6$ , caso contrário o algoritmo para de iterar após 1000 iterações.

4. Quais os valores dos parâmetros encontrados? Quantos usuários foram alocados a cada uma das duas classes?

Resposta:

- Médias:

$$\begin{bmatrix} 2.80142676 & 2.82231989 & 5.28452529 & 5.71271474 \\ 1.92130354 & 1.52285243 & 7.59991985 & 5.84940941 \\ 5.71049743 & 5.66366009 & 2.56139984 & 2.04457428 \end{bmatrix}$$

- Covariâncias:

- Classe 1:

$$\begin{bmatrix} 1.86255092 & 0.61817744 & 0.29823839 & 0.08422267 \\ 0.61817744 & 2.0491576 & 0.54000828 & 0.43614989 \\ 0.29823839 & 0.54000828 & 2.69371827 & 1.43494672 \\ 0.08422267 & 0.43614989 & 1.43494672 & 3.16638644 \end{bmatrix}$$

- Classe 2:

$$\begin{bmatrix} 0.28133565 & 0.20948873 & 0.45271546 & 0.48887734 \\ 0.20948873 & 0.41354931 & 0.4529761 & 0.57520209 \\ 0.45271546 & 0.4529761 & 1.47924585 & 0.80620238 \\ 0.48887734 & 0.57520209 & 0.80620238 & 3.89680154 \end{bmatrix}$$

- Classe 3:

$$\begin{bmatrix} 3.34855618 & 1.36090645 & 0.50617658 & 0.41959007 \\ 1.36090645 & 2.53600431 & 0.4211376 & 0.37739653 \\ 0.50617658 & 0.4211376 & 1.59672276 & 0.37515429 \\ 0.41959007 & 0.37739653 & 0.37515429 & 1.54897683 \end{bmatrix}$$

- Probabilidades a priori:

$$\begin{bmatrix} 0.51228627 \\ 0.01069369 \\ 0.47702004 \end{bmatrix}$$

Resposta (continuação):

- **Número de usuários alocados a cada classe:**
  - **Classe 1:** 515 usuários
  - **Classe 2:** 5 usuários
  - **Classe 3:** 480 usuários
  - **Total:** 1000 usuários
  - **Proporção:** 51.5% dos usuários estão na classe 1, 0.5% na classe 2 e 48% na classe 3.

5. O resultado da clusterização fez algum sentido? Explique e justifique a sua resposta.

Resposta:

A classe 2 tem médias maiores para os filmes de Romance, enquanto que a classe 3 tem médias maiores para os filmes de Sci-Fi. A classe 1, apesar de ter médias maiores para os filmes de Romance dentro das duas próprias médias, tem médias mais equilibradas para ambos os tipos de filme se comparada com as outras duas classes.

Considerando o dataset original, pode-se associar a classe 1 a Balanced, classe 2 a Romance Lover e a classe 3 a Sci-Fi Love, mostrando que a clusterização fez um certo sentido.

Entretanto, observando o dataset original, temos:

- 419 clientes da classe Sci-Fi Lover
- 379 clientes da classe Romance Lover
- 202 clientes da classe Balanced

O que indica que a clusterização não capturou bem os tipos de cliente, o que provavelmente ocorreu pelo grande número de valores faltantes no dataset. Ainda assim, a clusterização feita fez sentido, considerando as médias dos filmes de Sci-Fi e Romance para cada classe.

6. Qual a probabilidade do  $i$ -ésimo cliente ser um cliente que gosta mais de Sci-Fi? Explique sua resposta de forma genérica e escolha um dos 1000 usuários para exemplificar.

Resposta:

Essa probabilidade é dada pela proporia responsabilidade  $r_{ik}$  do cliente  $i$  para a classe  $k$ .

Para o cliente 1, temos:

$$x_1 = [6.1, 7.4, 3.3, 3.7, \text{Sci} - \text{Fi Lover}]$$

Após a clusterização, como associamos a classe Sci-Fi Lover ao cluster 3, temos que a probabilidade do cliente 1 ser um cliente que gosta mais de Sci-Fi é dada por

$$r_{13} = 0.9994$$

O que indica que o cliente 1 tem uma probabilidade muito alta de ser um cliente que gosta mais de Sci-Fi, condizente com a ultima coluna do dataset original.

## Questão 3 - Markov Reward Models

Considere a Questão 1, e o seguinte problema. A Figura 1 é modificada, de forma que o muro no quadrado  $[3, 4]$  é retirado, e dá lugar a um quadrado vermelho. Além disso, o robô ganha um prêmio de R\$100,00 ao atingir o quadrado  $[4, 4]$  (azul), mas perde:

- R\$40,00 cada vez que passa por um quadrado verde;
- R\$30,00 cada vez que passa por um quadrado vermelho;
- R\$5,00 cada vez que passa por um quadrado azul;
- R\$10,00 cada vez que passa por um quadrado amarelo.

O robô perde R\$1,00 a cada movimento, mesmo que sendo para o mesmo quadrado. Suponha que o robô escolhe uma das 4 direções aleatoriamente e caso a direção seja uma parede ele permanece no mesmo quadrado, (exatamente como no problema da lista anterior) e perde dinheiro conforme explicado acima.

1. Ignore a indicação dos sensores e mostre os passos necessários para calcular o valor médio do valor recebido ao atingir o quadrado do prêmio.

Resposta:

Como o enunciado diz para ignorar a indicação dos sensores, o problema se torna uma Cadeia de Markov comum. A modificação no quadrado  $[3, 4]$  (estado  $S_{12}$ ) e o fato de que o quadrado  $[4, 4]$  é agora um estado absorvente, alteram a matriz de transição  $P$  para a exibida na tabela 3.

Removendo-se os estados proibidos, matriz de transição  $P$  é dada pode ser rearranjada em blocos:

$$P = \begin{bmatrix} Q & R \\ 0 & I \end{bmatrix} \quad (26)$$

onde  $Q$  é a matriz de transição dos estados não absorventes (tabela 4),  $R$  é a matriz de transição dos estados absorventes (tabela 5) e  $I$  é a matriz identidade.

O número esperado de visitas a cada estado é dado pela matriz fundamental:

$$N = (I - Q)^{-1} \quad (27)$$

Seja  $b$  o vetor que representa o custo ao se entrar em cada estado transitório:

$$b = \begin{bmatrix} S_2 : 41 \\ S_3 : 11 \\ S_4 : 6 \\ S_5 : 31 \\ S_7 : 41 \\ S_8 : 11 \\ S_9 : 41 \\ S_{10} : 6 \\ S_{11} : 31 \\ S_{12} : 31 \\ S_{14} : 31 \\ S_{15} : 11 \end{bmatrix} \quad (28)$$

O custo total esperado ao se atingir o estado  $S_{16}$ , iniciando o movimento em cada estado transitório, é dado por:

$$E_c = b^T N \quad (29)$$

Assim, o valor recebido ao atingir o quadrado do prêmio é:

$$E_r = 100 - E_c(S_5), \quad (30)$$

Onde  $E_c(S_5)$  é o custo total esperado ao se atingir o estado  $S_{16}$ , iniciando o movimento no estado  $S_5$ .

O procedimento foi implementado em Python e seu código é exibido no código 5.

O resultado foi que o robô irá perder em média R\$ -1126.00 ao atingir o quadrado do prêmio. Ele sempre irá perder dinheiro, pois o prêmio esperado ao se atingir o quadrado  $[4, 4]$  é muito baixo, se comparado ao custo esperado ao se atingir cada estado transitório.

	$S_1$	$S_2$	$S_3$	$S_4$	$S_5$	$S_6$	$S_7$	$S_8$	$S_9$	$S_{10}$	$S_{11}$	$S_{12}$	$S_{13}$	$S_{14}$	$S_{15}$	$S_{16}$
$S_1$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$S_2$	0	0.75	0.25	0	0	0	0	0	0	0	0	0	0	0	0	0
$S_3$	0	0.25	0.25	0.25	0	0	0.25	0	0	0	0	0	0	0	0	0
$S_4$	0	0	0.25	0.50	0	0	0	0.25	0	0	0	0	0	0	0	0
$S_5$	0	0	0	0	0.75	0	0	0	0.25	0	0	0	0	0	0	0
$S_6$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$S_7$	0	0	0.25	0	0	0	0.25	0.25	0	0	0.25	0	0	0	0	0
$S_8$	0	0	0	0.25	0	0	0.25	0.25	0	0	0	0.25	0	0	0	0
$S_9$	0	0	0	0	0.25	0	0	0	0.50	0.25	0	0	0	0	0	0
$S_{10}$	0	0	0	0	0	0	0	0	0.25	0.25	0.25	0	0	0.25	0	0
$S_{11}$	0	0	0	0	0	0	0.25	0	0	0.25	0.25	0.25	0	0	0.25	0
$S_{12}$	0	0	0	0	0	0	0	0.25	0	0	0.25	0.25	0	0	0	0.25
$S_{13}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$S_{14}$	0	0	0	0	0	0	0	0	0	0.25	0	0	0	0.50	0.25	0
$S_{15}$	0	0	0	0	0	0	0	0	0	0	0.25	0	0	0.25	0.25	0.25
$S_{16}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

**Tabela 3:** Matriz de Transição  $P$

	$S_2$	$S_3$	$S_4$	$S_5$	$S_7$	$S_8$	$S_9$	$S_{10}$	$S_{11}$	$S_{12}$	$S_{14}$	$S_{15}$
$S_2$	0.75	0.25	0	0	0	0	0	0	0	0	0	0
$S_3$	0.25	0.25	0.25	0	0.25	0	0	0	0	0	0	0
$S_4$	0	0.25	0.50	0	0	0.25	0	0	0	0	0	0
$S_5$	0	0	0	0.75	0	0	0.25	0	0	0	0	0
$S_7$	0	0.25	0	0	0.25	0.25	0	0	0.25	0	0	0
$S_8$	0	0	0.25	0	0.25	0.25	0	0	0	0.25	0	0
$S_9$	0	0	0	0.25	0	0	0.50	0.25	0	0	0	0
$S_{10}$	0	0	0	0	0	0	0.25	0.25	0.25	0	0.25	0
$S_{11}$	0	0	0	0	0.25	0	0	0.25	0.25	0	0	0.25
$S_{12}$	0	0	0	0	0	0.25	0	0	0.25	0.25	0	0
$S_{14}$	0	0	0	0	0	0	0	0.25	0	0	0.50	0.25
$S_{15}$	0	0	0	0	0	0	0	0	0.25	0	0.25	0.25

**Tabela 4:** Matriz de Transição de estados não absorventes  $Q$

	$S_{16}$
$S_2$	0
$S_3$	0
$S_4$	0
$S_5$	0
$S_7$	0
$S_8$	0
$S_9$	0
$S_{10}$	0
$S_{11}$	0
$S_{12}$	0.25
$S_{14}$	0
$S_{15}$	0.25

**Tabela 5:** Matriz de Transição dos estados absorventes  $R$

**Código 5:** Cálculo do valor médio do valor recebido ao atingir o quadrado do prêmio

```

1 import numpy as np
2
3 # Matriz Q (12x12)
4 Q = np.array([
5     [0.75, 0.25, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
6     [0.25, 0.25, 0.25, 0, 0.25, 0, 0, 0, 0, 0, 0, 0],
7     [0, 0.25, 0.5, 0, 0, 0.25, 0, 0, 0, 0, 0, 0],
8     [0, 0, 0, 0.75, 0, 0, 0.25, 0, 0, 0, 0, 0],
9     [0, 0.25, 0, 0, 0.25, 0.25, 0, 0, 0.25, 0, 0, 0],
10    [0, 0, 0.25, 0, 0.25, 0.25, 0, 0, 0, 0.25, 0, 0],
11    [0, 0, 0, 0.25, 0, 0, 0.5, 0.25, 0, 0, 0, 0],
12    [0, 0, 0, 0, 0, 0, 0.25, 0.25, 0.25, 0, 0.25, 0],
13    [0, 0, 0, 0, 0.25, 0, 0, 0.25, 0.25, 0, 0, 0.25],
14    [0, 0, 0, 0, 0, 0.25, 0, 0, 0.25, 0.25, 0, 0],
15    [0, 0, 0, 0, 0, 0, 0, 0.25, 0, 0, 0.5, 0.25],
16    [0, 0, 0, 0, 0, 0, 0, 0, 0.25, 0, 0.25, 0.25],
17 ])
18 # print("Somadas linhas de Q:", Q.sum(axis=1))
19
20 # Vetor b (custos totais por transição)
21 b = np.array([
22     41, 11, 6, 31, 41, 11, 41, 6, 31, 31, 31, 11
23 ])
24
25 # Identidade para o cálculo de N
26 I = np.eye(len(Q))
27 # det_I_minus_Q = np.linalg.det(I - Q)
28 # print("Determinante de (I - Q):", det_I_minus_Q)
29
30 # Cálculo da matriz fundamental N
31 N = np.linalg.inv(I - Q)
32
33 # Cálculo do custo total esperado E_c

```

```

34 E_c = np.dot(N, b)
35
36 # Exibe os resultados
37 print("Matriz Fundamental (N):")
38 print(N)
39 print("\nCusto Total Esperado (E_c) com o robô iniciando em cada estado transit
    ório:")
40 print(E_c)
41 print("\nCusto Total Esperado (E_c) com o robô iniciando no estado s5:"),
42 print(E_c[4])
43
44 # Calculo do valor recebido
45 E_r = 100 - E_c[4]
46 print("\nValor recebido: R$ %.2f" % E_r)

```



## Códigos

Os códigos utilizados para a resolução dos exercícios estão disponíveis no repositório do GitHub:  
<https://github.com/lhscaldas/cps863/>

## Referências

- [1] RABINER, L. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, v. 77, n. 2, p. 257–286, 1989.