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.

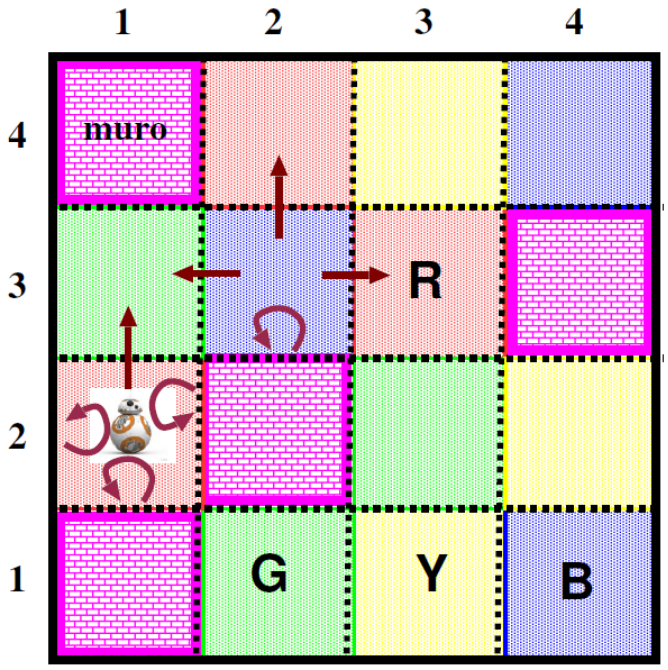

Figura 1: Robô andando por um ambiente

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 sewr 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 π 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.

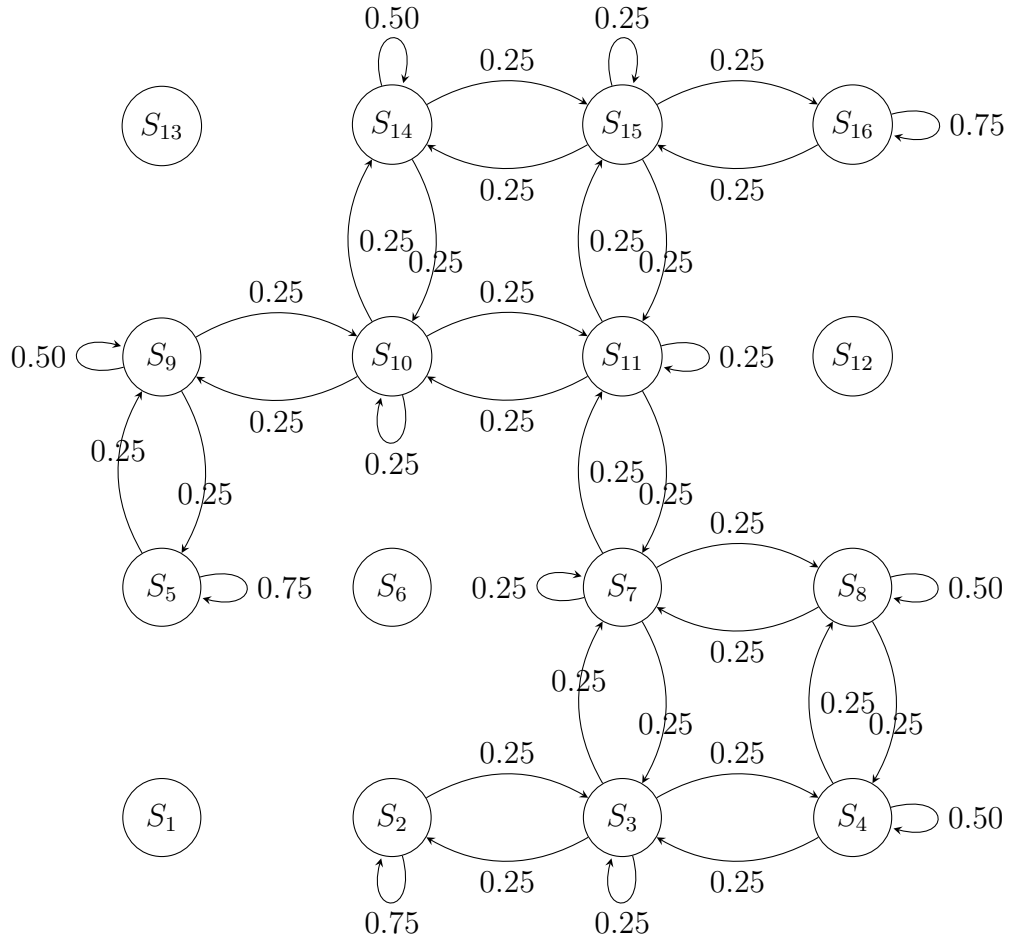

Figura 2: Cadeia de Markov representando o movimento do robô no ambiente.

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

$$\alpha_1(i) = \pi_i b_i(O_1) \tag{1}$$

- 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.

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

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

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

onde λ é o modelo, N é o número de estados, π_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 α 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                 """
```

```
        Inicializa o modelo HMM.

        :param A: Matriz de transição de estados (N x N)
        :param B: Matriz de emissão de observações (N x M)
        :param pi: Vetor de probabilidades iniciais (1 x N)
        """
        self.A = A  # Matriz de transição
        self.B = B  # Matriz de emissão
        self.pi = pi  # Probabilidades iniciais

    def forward(self, O):
        """
        Calcula o alpha para todos os estados e tempos para uma sequência O.

        :param O: Sequência de observações (lista de índices das observações)
        :return: Matriz alpha (T x N)
        """
        N = len(self.pi)  # Número de estados
        T = len(O)  # Comprimento da sequência de observações

        # Inicialização
        alpha = np.zeros((T, N))
        alpha[0, :] = self.pi * self.B[:, O[0]]

        # Recursão
        for t in range(1, T):
            for j in range(N):
                alpha[t, j] = np.sum(alpha[t - 1, :] * self.A[:, j]) * self.B[j, O[t]]

        return alpha

    def P_obs(self, alpha):
        """
        Calcula a probabilidade da sequência de observações, dado o modelo.

        :param alpha: Matriz alpha (T x N) calculada pelo método forward
        :return: Probabilidade da sequência de observações
        """
        return np.sum(alpha[-1, :])
```

Resposta (continuação):

O resultado obtido para a sequência *rryryrbgbryygb* foi:

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

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:

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

O resultado obtido foi cerca de 4 vezes menor que o obtido na sequência anterior, mostrando que essa sequencia 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 λ e a sequência de observações O, sendo dada por:

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

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

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

Para calcular o estado final mais provável, foram implementados os métodos backward e gamma, que calculam as matrizes β e γ, 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

```
    def backward(self, O):
        """
        Calcula o beta para todos os estados e tempos para uma sequência O
            .

        :param O: Sequência de observações (lista de índices das observaçõ
            es)
        :return: Matriz beta (T x N)
        """
        N = len(self.pi)
        T = len(O)

        # Inicialização
        beta = np.zeros((T, N))
        beta[-1, :] = 1

        # Recursão
        for t in range(T - 2, -1, -1):
            for i in range(N):
                beta[t, i] = np.sum(self.A[i, :] * self.B[:, O[t + 1]] *
                    beta[t + 1, :])

        return beta

    def gamma(self, alpha, beta):
        """
        Calcula a matriz gamma para todos os estados e tempos para uma
```

```
            sequência O.

        :param alpha: Matriz alpha (T x N) calculada pelo método forward
        :param beta: Matriz beta (T x N) calculada pelo método backward
        :return: Matriz gamma (T x N)
        """
        return alpha * beta / np.sum(alpha * beta, axis=1).reshape(-1, 1)

    def end_state(self, gamma):
            """
            Calcula o estado mais provável na última observação.

            :param gamma: Matriz gamma (T x N) calculada pelo método gamma
            :return: Estado mais provável na última observação
            """
            end_state = np.argmax(gamma[-1, :])
            P_end_state = gamma[-1, end_state]
            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'}
            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 *rbyrgrbgbryygb* 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 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:

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

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.

$$\delta_1(i) = \pi_i b_i(O_1) \tag{9}$$

e

$$\psi_1(i) = 0 \tag{10}$$

- 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.

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

e

$$\psi_t(j) = argmax_{1\leq i\leq N}[\delta_{t-1}(i)a_{ij}] \tag{12}$$

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

$$q_T^* = argmax_{1\leq i\leq N}\delta_T(i) \tag{13}$$

e

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

- 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.

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

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

```
                def viterbi(self, O):
                    # inicialização
                    N = len(self.pi)
                    T = len(O)
                    delta = np.zeros((T, N))
                    psi = np.zeros((T, N), dtype=int)
                    delta[0, :] = self.pi * self.B[:, O[0]]
                    psi[0, :] = 0

                    # recursão
                    for t in range(1, T):
                        for j in range(N):
                            delta[t, j] = np.max(delta[t - 1, :] * self.A
                                [:, j]) * self.B[j, O[t]]
                            psi[t, j] = np.argmax(delta[t - 1, :] * self.A
                                [:, j])

                    # finalização
                    P_star = np.max(delta[-1, :])
                    q_star = np.argmax(delta[-1, :])

                    # reconstrução do caminho
                    path = [q_star]
                    for t in range(T - 1, 0, -1):
                        q_star = psi[t, q_star]
                        path.insert(0, q_star)

                    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'}
                    path = [state_map[state] for state in path]
                    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 8.3918×10^{-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 9.3242×10^{-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.**
2. **Baseado no item anterior, explique a sua implementação, incluindo as suas escolhas para a inicialização do código.**
3. **Quantas iterações foram necessárias para resolver o problema? Qual o teste de parada utilizado?**
4. **Quais os valores dos parâmetros encontrados? Quantos usuários foram alocados a cada uma das duas classes?**
5. **O resultado da clusterização fez algum sentido? Explique e justifique a sua resposta.**
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.**

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 rearrajnajada em blocos:

$$P = \begin{bmatrix} Q & R \\ 0 & I \end{bmatrix} \tag{16}$$

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:

$$N = (I - Q)^{-1} \tag{17}$$

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

$$b = \begin{bmatrix} S_2 : 41 \\ S_3 : 11 \\ S_4 : 6 \\ S_5 : 31 \\ S_7 : 41 \\ S_8 : 11 \\ S_9 : 41 \\ S_{10} : 6 \\ S_{11} : 31 \\ S_{12} : 31 \\ S_{14} : 31 \\ S_{15} : 11 \end{bmatrix} \tag{18}$$

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

$$E_c = b^T N \tag{19}$$

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

$$E_r = 100 - E_c(S_5), \tag{20}$$

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 4.
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 4: Cálculo do valor médio do valor recebido ao atingir o quadrado do prêmio

```
import numpy as np

# Matriz Q (12x12)
Q = np.array([
    [0.75, 0.25, 0,    0,    0,    0,    0,    0,    0,    0,    0,    0   ],
    [0.25, 0.25, 0.25, 0,    0.25, 0,    0,    0,    0,    0,    0,    0   ],
    [0,    0.25, 0.5,  0,    0,    0.25, 0,    0,    0,    0,    0,    0   ],
    [0,    0,    0,    0.75, 0,    0,    0.25, 0,    0,    0,    0,    0   ],
    [0,    0.25, 0,    0,    0.25, 0.25, 0,    0,    0.25, 0,    0,    0   ],
    [0,    0,    0.25, 0,    0.25, 0.25,  0,    0,    0,    0.25, 0,    0   ],
    [0,    0,    0,    0.25, 0,    0,    0.5,  0.25, 0,    0,    0,    0   ],
    [0,    0,    0,    0,    0,    0,    0.25, 0.25, 0.25, 0,    0.25, 0   ],
    [0,    0,    0,    0,    0.25, 0,    0,    0.25, 0.25, 0,    0,    0.25],
    [0,    0,    0,    0,    0,    0.25, 0,    0,    0.25, 0.25, 0,    0   ],
    [0,    0,    0,    0,    0,    0,    0,    0.25, 0,    0,    0.5,  0.25],
    [0,    0,    0,    0,    0,    0,    0,    0,    0.25, 0,    0.25, 0.25],
])
# print("Somas das linhas de Q:", Q.sum(axis=1))

# Vetor b (custos totais por transição)
b = np.array([
    41, 11, 6, 31, 41, 11, 41, 6, 31, 31, 31, 11
])

# Identidade para o cálculo de N
I = np.eye(len(Q))
# det_I_minus_Q = np.linalg.det(I - Q)
# print("Determinante de (I - Q):", det_I_minus_Q)

# Cálculo da matriz fundamental N
N = np.linalg.inv(I - Q)

# Cálculo do custo total esperado E_c
```

```
E_c = np.dot(N, b)

# Exibe os resultados
print("Matriz Fundamental (N):")
print(N)
print("\nCusto Total Esperado (E_c) com o robô iniciando em cada estado transitório:")
print(E_c)
print("\nCusto Total Esperado (E_c) com o robô iniciando no estado s5:"),
print(E_c[4])

# Calculo do valor recebido
E_r = 100 - E_c[4]
print("\nValor recebido: R$ %.2f" % E_r)
```

Codigos

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.