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 4

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

13 de novembro de 2024

Questão 1

O objetivo desta questão é adquirir experiência com o algoritmo Expectation Maximization (EM). Para isso, você deve implementar sua solução, que deverá estar bem explicada. A implementação pode ser feita facilmente em Octave, R, Python, etc.

O dataset fornecido contém o resultado de uma votação de mil usuários em 5 tipos de filmes. Cada usuário dá uma nota de 1 a 4 para cada tipo de filme, onde 1 significa "não gosto"e 4 "gosto muito". Os tipos de filmes são, por exemplo, Ação, Comédia, Romance, Suspense e Ficção Científica.

O objetivo é verificar se é possível classificar os usuários usando 2 classes. Para isso, você escolhe o modelo Naive Bayes pela simplicidade e deve utilizar o algoritmo EM para essa tarefa.

(a) **Quantos parâmetros deverão ser estimados para o modelo Naive Bayes a ser construído?**

Para um modelo Naive Bayes com duas classes e cinco tipos de filmes, onde cada usuário atribui uma nota de 1 a 4, precisamos estimar os seguintes parâmetros:

- Probabilidades a priori para cada classe $P(C_k)$, totalizando 2 parâmetros.
- Probabilidades condicionais para cada possível nota (1 a 4) em cada tipo de filme, por classe $P(X_{i,j}|C_k)$. Com 5 tipos de filmes e 4 possíveis notas, temos $5 \times 4 \times 2 = 40$ parâmetros.

Entretanto, como a soma das probabilidades a priori deve ser 1 e a soma das probabilidades condicionais para cada tipo de filme para cada classe também deve ser 1, podemos reduzir o número de parâmetros. Assim, temos um total de $1 + 5 \times 3 \times 2 = 31$ parâmetros a serem estimados.

(b) **Explique as equações usadas para resolver o problema.**

Para resolver o problema utilizando o modelo Naive Bayes com Expectation Maximization (EM), utilizei as seguintes equações:

Definição das Variáveis:

- C_k: Classe k (onde $k = 1, 2$, pois temos duas classes).
- X_i: Observação do i-ésimo usuário, representando suas notas para os diferentes tipos de filmes.
- F_j: Tipo de filme j (ex.: Ação, Comédia, etc., onde $j = 1, 2, \ldots, 5$ para cinco tipos de filmes).
- $X_{i,j}$: Nota atribuída pelo i-ésimo usuário ao tipo de filme F_j, variando entre 1 e 4.
- $P(C_k)$: Probabilidade a priori de um usuário pertencer à classe C_k.
- $P(X_{i,j}|C_k)$: Probabilidade condicional de uma nota específica $X_{i,j}$ em um tipo de filme F_j, dado que o usuário pertence à classe C_k.
- $r_{i,k}$: Responsabilidade de cada classe C_k pela observação X_i, representando a probabilidade de X_i pertencer à classe C_k dado o conjunto de notas.

Explicação das Equações:

- **Probabilidades a priori:** A probabilidade a priori de cada classe C_k é estimada pela proporção de usuários na classe C_k:

$$P(C_k) = \frac{\text{Número de observações em } C_k}{\text{Número total de observações (usuários)}}$$

- **Probabilidade Condicional de uma Feature:** Com todos os usuários avaliando cada tipo de filme, a probabilidade condicional para cada nota $X_{i,j}$ atribuída a F_j dado C_k é dada por:

$$P(X_{i,j}|C_k) = \frac{\text{Número de vezes que a nota } X_{i,j} \text{ foi observada para } F_j \text{ em } C_k}{\text{Número total de usuários em } C_k}$$

- **Responsabilidade (E-step):** A responsabilidade $r_{i,k}$ representa a probabilidade de que uma observação X_i pertence à classe C_k dado o conjunto de notas, e é dada por:

$$r_{i,k} = P(C_k|X_i) = \frac{P(C_k)\prod_{j=1}^{5} P(X_{i,j}|C_k)}{\sum_{l=1}^{2} P(C_l)\prod_{j=1}^{5} P(X_{i,j}|C_l)}$$

- **Atualização de parâmetros (M-step):** Utilizando as responsabilidades $r_{i,k}$, atualizamos as probabilidades a priori:

$$P(C_k) = \frac{\sum_{i=1}^{n} r_{i,k}}{n}$$

e as probabilidades condicionais:

$$P(X_{i,j}|C_k) = \frac{\sum_{i=1}^{n} r_{i,k} \cdot \mathbb{1}(X_{i,j} = x)}{\sum_{i=1}^{n} r_{i,k}}$$

onde $\mathbb{1}(X_{i,j} = x)$ é uma função indicadora que vale 1 quando $X_{i,j} = x$ e 0 caso contrário. Essas atualizações continuam até atingir o critério de parada.

(c) **Baseado no item anterior, explique sua implementação, incluindo as escolhas para a inicialização do código.**

O código 1 implementa o algoritmo Expectation-Maximization (EM) para um modelo Naive Bayes, com o objetivo de estimar as probabilidades a priori das classes e as probabilidades condicionais de cada nota para cada tipo de filme.

Primeiro, as constantes do problema são definidas: número de classes (2), tipos de filmes (5) e valores possíveis para as notas (de 1 a 4). As probabilidades a priori das classes são inicializadas aleatoriamente usando uma distribuição Dirichlet, garantindo que a soma das probabilidades seja 1. As probabilidades condicionais para cada combinação de nota, tipo de filme e classe também são inicializadas com uma distribuição Dirichlet, criando um array tridimensional [classe, tipo de filme, nota].

O algoritmo EM é implementado em uma função que realiza o processo iterativo até convergir.

No **E-step**, o código calcula as responsabilidades para cada observação e classe. A responsabilidade é calculada multiplicando a probabilidade a priori da classe pela probabilidade condicional da sequência de notas observadas para cada tipo de filme. Esses valores são então normalizados para que a soma das responsabilidades para cada observação seja 1.

No **M-step**, as probabilidades a priori e condicionais são atualizadas. As novas probabilidades a priori são obtidas pela média das responsabilidades de cada classe sobre todas as observações. As probabilidades condicionais são recalculadas como somas ponderadas das responsabilidades para cada nota, tipo de filme e classe. Em seguida, elas são normalizadas para garantir que cada conjunto de probabilidades condicionais some 1.

O **critério de parada** verifica a diferença entre as probabilidades antigas e novas. Se essa diferença for menor que um valor de tolerância, o algoritmo é interrompido, indicando convergência. Caso contrário, o processo é repetido até o limite de iterações.

Código 1: Algoritmo de Expectation-Maximization (EM)

```
import numpy as np
import pandas as pd

# Carregar o dataset onde os valores estão separados por espaços
# "header=None" indica que o arquivo não possui cabeçalhos
# "names=[...]" define os nomes das colunas para cada tipo de filme
data = pd.read_csv("lista_4/lista4-data_scores.csv", sep='\s+', header=
    None, names=['filme_1', 'filme_2', 'filme_3', 'filme_4', 'filme_5'])

# Inicializações de constantes
num_classes = 2         # Número de classes (duas)
num_filmes = 5          # Número de tipos de filmes
num_notas = 4           # Número de notas possíveis (1 a 4)

# Inicializar as probabilidades a priori das classes aleatoriamente usando
    a distribuição Dirichlet
# Isso garante que a soma das probabilidades seja 1
p_class = np.random.dirichlet(np.ones(num_classes), size=1).flatten()

# Inicializar as probabilidades condicionais para cada nota em cada tipo
    de filme em cada classe
# Utiliza a distribuição Dirichlet para garantir que as probabilidades
    condicionais somem 1
p_feature_given_class = np.random.dirichlet(np.ones(num_notas), size=(
    num_classes, num_filmes))

# Função de Expectativa-Maximização (E-step e M-step)
def expectation_maximization(data, p_class, p_feature_given_class,
    max_iter=100, tol=1e-4):
    # Iniciar o loop de iterações do algoritmo EM
    for iteration in range(max_iter):
        # Matriz para armazenar as responsabilidades de cada observação
            para cada classe
        responsibilities = np.zeros((len(data), num_classes))

        # E-step: calcular as responsabilidades
```

```
        for i in range(len(data)):                # Itera sobre cada observação
            for k in range(num_classes):          # Itera sobre cada classe
                prob = p_class[k]                 # Probabilidade a priori da classe k
                for j in range(num_filmes):       # Itera sobre cada tipo de filme
                    # Ajusta a nota para índice (0-3) e multiplica pela probabilidade condicional
                    nota = data.iloc[i, j] - 1
                    prob *= p_feature_given_class[k, j, nota]
                responsibilities[i, k] = prob  # Guarda a probabilidade acumulada para a classe k

            # Normaliza as responsabilidades para que somem 1 para cada observação
            responsibilities[i, :] /= responsibilities[i, :].sum()

        # M-step: atualizar as probabilidades a priori e condicionais
        # Atualiza as probabilidades a priori como a média das responsabilidades
        new_p_class = responsibilities.mean(axis=0)

        # Cria um novo array para armazenar as probabilidades condicionais atualizadas
        new_p_feature_given_class = np.zeros_like(p_feature_given_class)

        # Calcula as novas probabilidades condicionais
        for k in range(num_classes):              # Itera sobre cada classe
            for j in range(num_filmes):           # Itera sobre cada tipo de filme
                for nota in range(num_notas):     # Itera sobre cada nota
                    # Soma ponderada das responsabilidades para cada nota no tipo de filme e classe
                    new_p_feature_given_class[k, j, nota] = np.sum(responsibilities[:, k] * (data.iloc[:, j] == (nota + 1)))
                # Normaliza as probabilidades para que somem 1 para cada tipo de filme na classe k
                new_p_feature_given_class[k, j, :] /= responsibilities[:, k].sum()

        # Critério de parada: verifica se a diferença entre as probabilidades antigas e novas é menor que a tolerância
        if np.linalg.norm(p_class - new_p_class) < tol and np.linalg.norm(p_feature_given_class - new_p_feature_given_class) < tol:
            # Imprime mensagem de convergência quando o critério de parada é atendido
            print(f"Convergência na iteração {iteration}")
            break

        # Atualiza as probabilidades para a próxima iteração
        p_class, p_feature_given_class = new_p_class, new_p_feature_given_class
```

```
66
67      return p_class, p_feature_given_class, responsibilities
68
69  if __name__ == '__main__':
70      # Executar o algoritmo EM
71      p_class_final, p_feature_given_class_final, responsibilities =
            expectation_maximization(data, p_class, p_feature_given_class)
72
73      # Imprimir as probabilidades a priori e condicionais finais
74      print("Probabilidades a priori finais:")
75      print(p_class_final)
76      print("\nProbabilidades condicionais finais:")
77      print(p_feature_given_class_final)
78
79      # Calcular quantos usuários foram alocados a cada classe após a
            convergência
80      class_allocation = responsibilities.argmax(axis=1)  # Classe com maior
            responsabilidade para cada observação
81      class_counts = np.bincount(class_allocation, minlength=num_classes)
82
83      print("\nQuantidade de usuários alocados em cada classe:")
84      print(f"Classe 0: {class_counts[0]}")
85      print(f"Classe 1: {class_counts[1]}")
```

(d) **Quantas iterações foram necessárias para resolver o problema? Qual o teste de parada utilizado?**

Foram necessárias 32 iterações para resolver o problema. O critério de parada utilizado foi a diferença entre os valores dos parâmetros estimados em duas iterações consecutivas. O algoritmo para quando a diferença entre os valores dos parâmetros estimados é menor que 10^{-4}.

(e) **Quais os valores dos parâmetros encontrados? Quantos usuários foram alocados a cada uma das duas classes?**

Os valores dos parâmetros encontrados são apresentados nas tabelas 1, 2 e 3.

Tabela 1: Probabilidades a priori finais

Classe 0	Classe 1
0.4203	0.5797

Tabela 2: Probabilidades condicionais finais - Classe 0

	Nota 1	Nota 2	Nota 3	Nota 4
Tipo 1	0.5703	0.2006	0.1323	0.0967
Tipo 2	0.5760	0.1860	0.1265	0.1115
Tipo 3	0.2374	0.3492	0.2163	0.1971
Tipo 4	0.1095	0.1224	0.2991	0.4690
Tipo 5	0.1416	0.0896	0.2924	0.4763

Tabela 3: Probabilidades condicionais finais - Classe 1

	Nota 1	Nota 2	Nota 3	Nota 4
Tipo 1	0.0902	0.0926	0.3077	0.5095
Tipo 2	0.1068	0.0843	0.3188	0.4902
Tipo 3	0.3160	0.3178	0.1744	0.1918
Tipo 4	0.6037	0.1993	0.0885	0.1085
Tipo 5	0.6201	0.1990	0.0726	0.1083

A tabela 4 mostra a quantidade de usuários alocados em cada classe, o que foi calculado com base nas responsabilidades obtidas após a convergência do algoritmo. Foi atribuída para cada usuário a classe com maior responsabilidade.

Tabela 4: Quantidade de usuários alocados em cada classe

Classe 0	Classe 1
425	575

(f) **O resultado da clusterização fez algum sentido? Explique e justifique sua resposta.**

O resultado da clusterização parece fazer sentido, pois o modelo identificou dois grupos com padrões distintos de avaliação de filmes. As probabilidades a priori mostram uma divisão significativa entre as classes, com uma leve predominância de usuários na Classe 1. As probabilidades condicionais indicam que a Classe 0 tende a concentrar notas mais altas para certos tipos de filmes (tipos 4 e 5), enquanto a Classe 1 tende a concentrar notas mais altas para outros tipos (tipos 1 e 2). Assim, o modelo parece ter capturado diferentes perfis de avaliação entre os grupos de usuários, sugerindo que a clusterização reflete padrões de preferência reais.

(g) **Qual a probabilidade do i-ésimo usuário ser alocado ao cluster 1? Explique sua resposta de forma genérica e escolha um dos 1000 usuários para exemplificar.**

A probabilidade do primeiro usuário ser alocado ao cluster 1 é calculada com base na probabilidade a priori do cluster 1 e nas probabilidades condicionais de suas notas para cada tipo de filme, dados encontrados nas tabelas 1, 2 e 3. As notas do primeiro usuário são 3, 1, 2, 3 e 3 para os cinco tipos de filmes.

Para o **Cluster 1**, temos:

$$P(C_1) = 0.5797$$

$$P(\text{nota 3 no filme 1}|C_1) = 0.3077, \quad P(\text{nota 1 no filme 2}|C_1) = 0.1068$$

$$P(\text{nota 2 no filme 3}|C_1) = 0.3178, \quad P(\text{nota 3 no filme 4}|C_1) = 0.0885$$

$$P(\text{nota 3 no filme 5}|C_1) = 0.0726$$

Calculando a probabilidade conjunta para o cluster 1:

$$P(C_1) \times 0.3077 \times 0.1068 \times 0.3178 \times 0.0885 \times 0.0726 = 3.769 \times 10^{-5}$$

Para o **Cluster 0**, temos:

$$P(C_0) = 0.4203$$

$$P(\text{nota 3 no filme 1}|C_0) = 0.1323, \quad P(\text{nota 1 no filme 2}|C_0) = 0.5761$$

$$P(\text{nota 2 no filme 3}|C_0) = 0.3492, \quad P(\text{nota 3 no filme 4}|C_0) = 0.2991$$

$$P(\text{nota 3 no filme 5}|C_0) = 0.2924$$

Calculando a probabilidade conjunta para o cluster 0:

$$P(C_0) \times 0.1323 \times 0.5761 \times 0.3492 \times 0.2991 \times 0.2924 = 0.0028$$

A probabilidade do usuário pertencer ao cluster 1 é então:

$$\frac{3.769 \times 10^{-5}}{3.769 \times 10^{-5} + 0.0028} \approx 0.0134$$

Portanto, a probabilidade do primeiro usuário ser alocado ao cluster 1 é aproximadamente 1.34%, indicando que ele provavelmente pertence ao cluster 0.

(h) **Um usuário que votou 1,2,3,4,2 deveria ser alocado ao cluster 0 ou 1? Justifique.**

Para determinar se um usuário que votou nas notas 1, 2, 3, 4 e 2 deveria ser alocado ao cluster 0 ou 1, calculamos a probabilidade de essa sequência ter sido gerada por cada cluster e comparamos os resultados.

Para o **Cluster 0**:

$$P(C_0) = 0.4203$$

$$P(\text{nota 1 no filme 1}|C_0) = 0.5703, \quad P(\text{nota 2 no filme 2}|C_0) = 0.1860$$

$$P(\text{nota 3 no filme 3}|C_0) = 0.2163, \quad P(\text{nota 4 no filme 4}|C_0) = 0.4690$$

$$P(\text{nota 2 no filme 5}|C_0) = 0.0896$$

A probabilidade conjunta para o cluster 0 é:

$$P(C_0) \times 0.5703 \times 0.1860 \times 0.2163 \times 0.4690 \times 0.0896 = 0.0004$$

Para o **Cluster 1**:

$$P(C_1) = 0.5797$$

$$P(\text{nota 1 no filme 1}|C_1) = 0.0902, \quad P(\text{nota 2 no filme 2}|C_1) = 0.0843$$

$$P(\text{nota 3 no filme 3}|C_1) = 0.1744, \quad P(\text{nota 4 no filme 4}|C_1) = 0.1085$$

$$P(\text{nota 2 no filme 5}|C_1) = 0.1990$$

A probabilidade conjunta para o cluster 1 é:

$$P(C_1) \times 0.0902 \times 0.0843 \times 0.1744 \times 0.1085 \times 0.1990 = 1.79 \times 10^{-5}$$

A probabilidade do usuário pertencer ao cluster 0 é, portanto, maior que a do cluster 1. Logo, este usuário deveria ser alocado ao **cluster 0**.

(i) **Como você classificaria um usuário que votou 3,2,?,2,3? Isto é, o usuário não votou na feature número 3. Explique.**

Para classificar um usuário que votou nas notas 3, 2, ?, 2 e 3, onde a terceira nota está ausente, calculamos a probabilidade de essa sequência de votos ter sido gerada por cada cluster, "marginalizando"sobre a feature ausente, e comparando os resultados. Esse método é válido porque o modelo Naive Bayes assume independência condicional entre as features, permitindo ignorar a feature ausente sem impactar as demais.

Os dados utilizados para o cálculo foram retirados das Tabelas 1, 2 e 3.

Primeiro, para o **Cluster 0**, utilizamos a probabilidade a priori $P(C_0) = 0.4203$ e as probabilidades condicionais apenas para as notas observadas:

$$\begin{aligned} P(C_0) &\times P(\text{nota 3 no filme 1}|C_0) \\ &\times P(\text{nota 2 no filme 2}|C_0) \\ &\times P(\text{nota 2 no filme 4}|C_0) \\ &\times P(\text{nota 3 no filme 5}|C_0) \end{aligned}$$

Substituindo os valores, obtemos:

$$0.4203 \times 0.1323 \times 0.1860 \times 0.2991 \times 0.2924 = 0.000962$$

Para o **Cluster 1**, utilizamos a probabilidade a priori $P(C_1) = 0.5797$ e as probabilidades condicionais para as notas observadas:

$$\begin{aligned} P(C_1) &\times P(\text{nota 3 no filme 1}|C_1) \\ &\times P(\text{nota 2 no filme 2}|C_1) \\ &\times P(\text{nota 2 no filme 4}|C_1) \\ &\times P(\text{nota 3 no filme 5}|C_1) \end{aligned}$$

Substituindo os valores, obtemos:

$$0.5797 \times 0.3077 \times 0.0843 \times 0.1085 \times 0.0726 = 0.000111$$

Essas probabilidades conjuntas representam as probabilidades não normalizadas de que as notas observadas tenham sido geradas por cada cluster. Para obter as probabilidades de pertença normalizadas (ou responsabilidades) para cada cluster, dividimos cada probabilidade conjunta pela soma das probabilidades conjuntas:

$$\begin{aligned} P(C_0|X) &= \frac{0.000962}{0.000962 + 0.000111} \approx 0.8967 \\ P(C_1|X) &= \frac{0.000111}{0.000962 + 0.000111} \approx 0.1033 \end{aligned}$$

Após a normalização, a soma das probabilidades de pertença aos clusters é 1. Como $P(C_0|X) \approx 0.8967$ é maior que $P(C_1|X) \approx 0.1033$, concluímos que o usuário com as notas (3, 2, ?, 2, 3) deveria ser alocado ao **Cluster 0**.

Questão 2

Um robô pode se mover pelos quadrados da figura abaixo. A cada unidade de tempo (minuto, por exemplo), o robô tenta se mover para um dos quadrados adjacentes ao que ocupa e escolhe uma dentre as possíveis ações: (a) mover para Norte; (b) mover para Sul; (c) mover para Leste; (d) mover para Oeste. O robô não consegue enxergar a sua volta, então escolhe aleatoriamente uma dentre as 4 ações possíveis para só então fazer um movimento. Caso o robô tenha escolhido se mover na direção de um dos quadrados em rosa (os com tijolinhos na Figura 1) ou para fora do ambiente, ele colide com a parede e permanece na mesma posição até a nova tentativa.

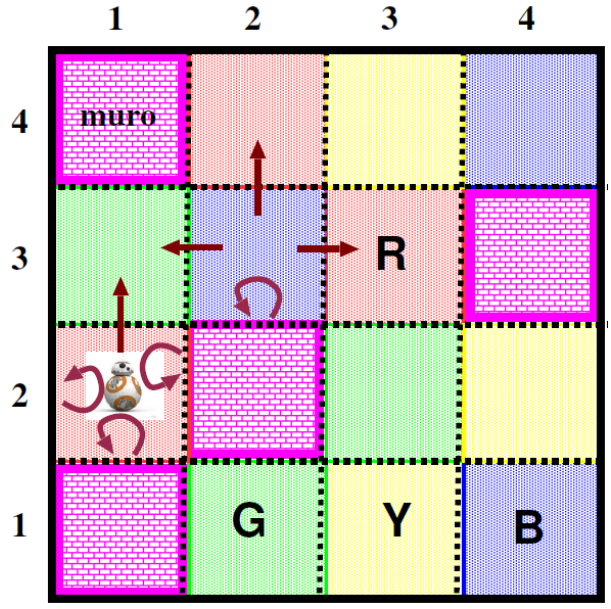

Figura 1: Robô andando por um ambiente

1. **Construa a cadeia de Markov que representa o movimento do robô pelo ambiente, e mostre a matriz de transição de estados do sistema. Para facilitar a correção, indique claramente quais os estados do sistema e exemplifique indicando as probabilidades de transição para um movimento.**

   Considerando que a probabilidade de o robô escolher uma das ações possíveis é igual para todas as ações, a Cadeia de Markov que representa o movimento do robô no ambiente é dada na Figura 2.

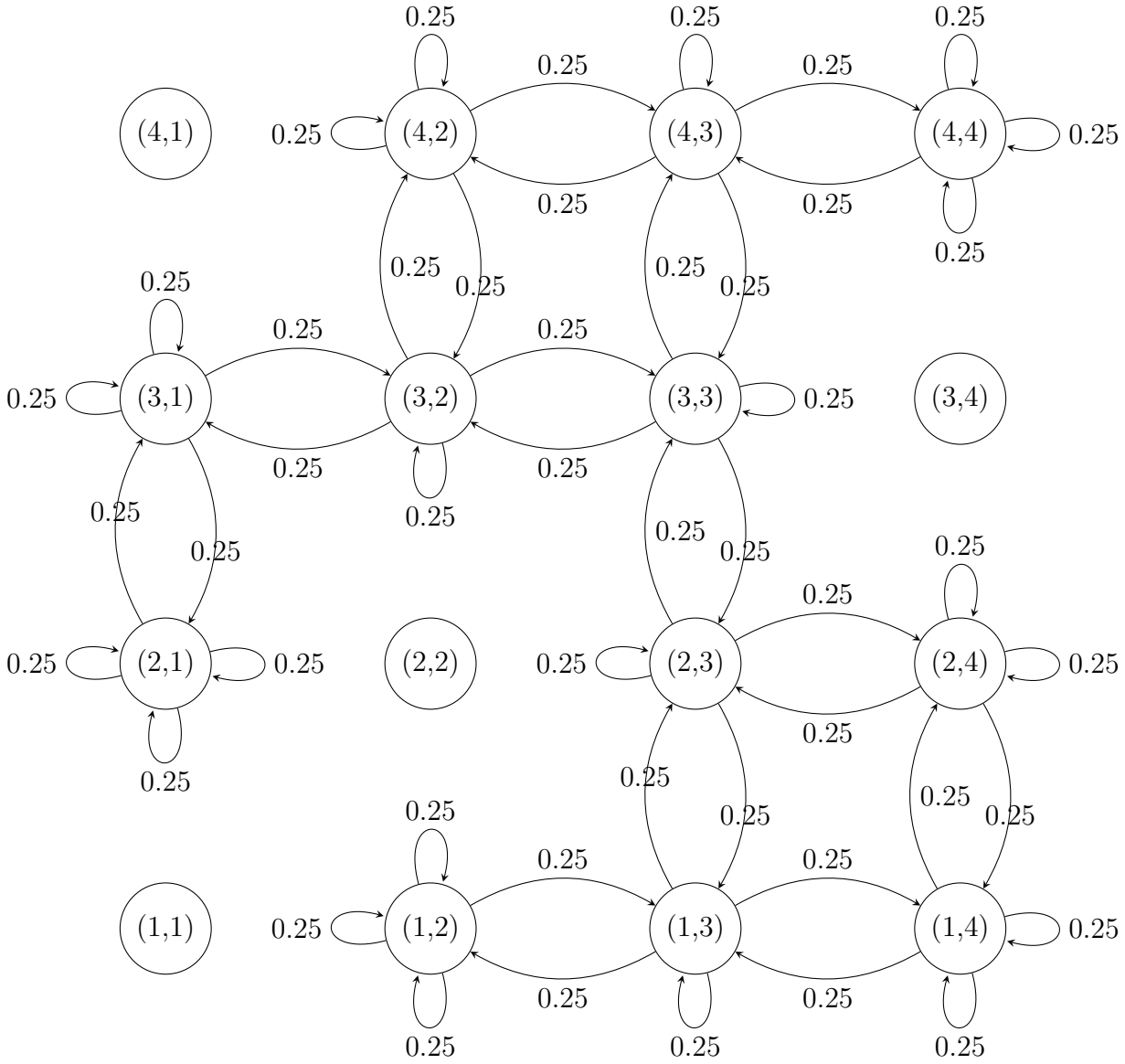

Figura 2: Cadeia de Markov representando o movimento do robô no ambiente.

Simplificando as transições redundantes nos estados que possuem mais de uma transição retornando para ele mesmo, a Cadeia de Markov que representa o movimento do robô no ambiente sem redundâncias é dada na Figura 3.

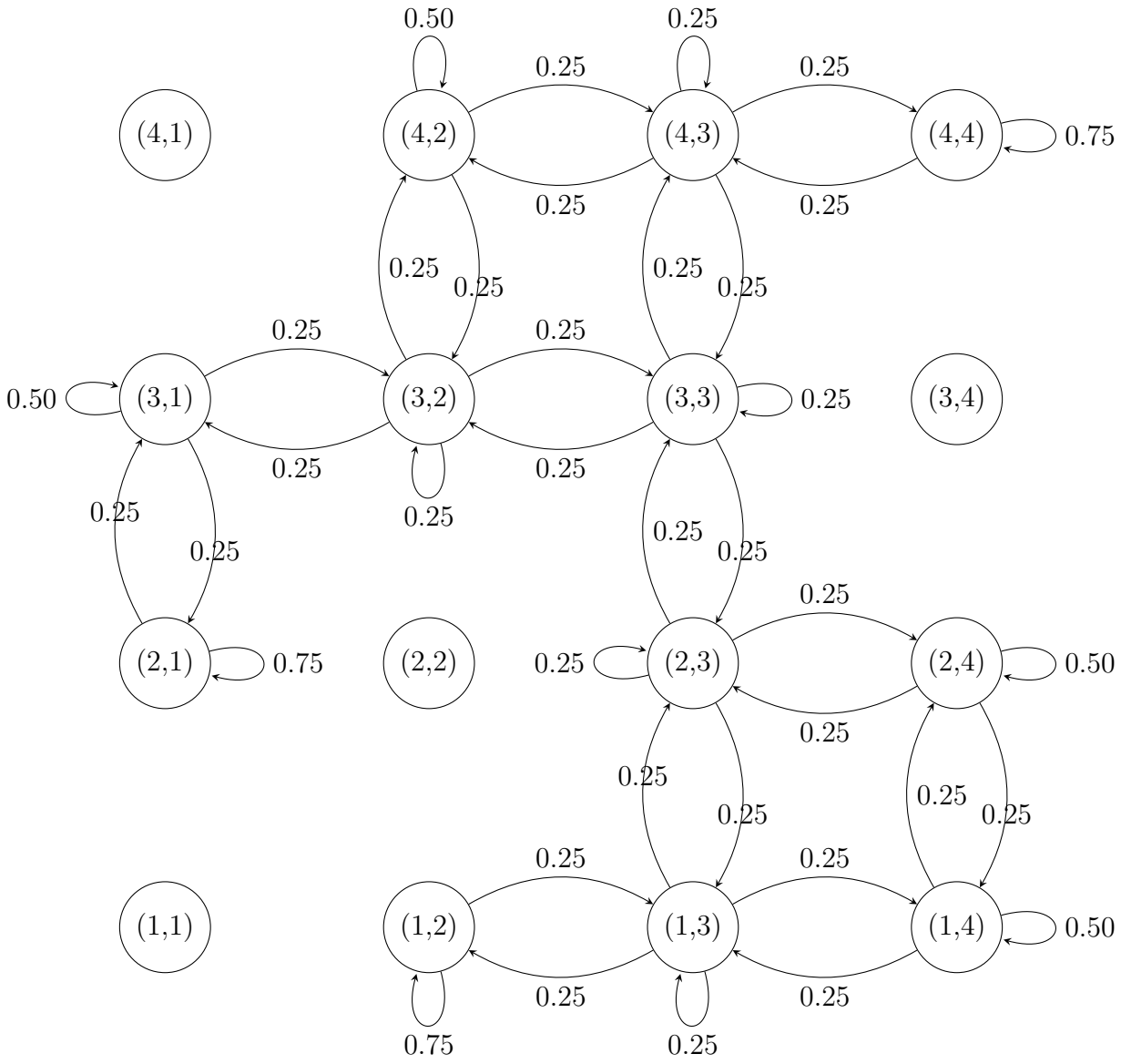

Figura 3: Cadeia de Markov representando o movimento do robô no ambiente, sem redundâncias.

Assim, a matriz de transição de estados P do sistema é dada por:

$P =$

	$(1,1)$	$(1,2)$	$(1,3)$	$(1,4)$	$(2,1)$	$(2,2)$	$(2,3)$	$(2,4)$	$(3,1)$	$(3,2)$	$(3,3)$	$(3,4)$	$(4,1)$	$(4,2)$	$(4,3)$	$(4,4)$
$(1,1)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$(1,2)$	0	0.75	0.25	0	0	0	0	0	0	0	0	0	0	0	0	0
$(1,3)$	0	0.25	0.25	0.25	0	0	0.25	0	0	0	0	0	0	0	0	0
$(1,4)$	0	0	0.25	0.50	0	0	0	0.25	0	0	0	0	0	0	0	0
$(2,1)$	0	0	0	0	0.75	0	0	0	0.25	0	0	0	0	0	0	0
$(2,2)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$(2,3)$	0	0	0.25	0	0	0	0.25	0.25	0	0	0.25	0	0	0	0	0
$(2,4)$	0	0	0	0.25	0	0	0.25	0.50	0	0	0	0	0	0	0	0
$(3,1)$	0	0	0	0	0.25	0	0	0	0.50	0.25	0	0	0	0	0	0
$(3,2)$	0	0	0	0	0	0	0	0	0.25	0.25	0.25	0	0	0.25	0	0
$(3,3)$	0	0	0	0	0	0	0.25	0	0	0.25	0.25	0	0	0	0.25	0
$(3,4)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$(4,1)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$(4,2)$	0	0	0	0	0	0	0	0	0	0.25	0	0	0	0.50	0.25	0
$(4,3)$	0	0	0	0	0	0	0	0	0	0	0.25	0	0	0.25	0.25	0.25
$(4,4)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.25	0.75

2. **Suponha que o robô inicia a sua caminhada no quadrado indicado na figura (quadrado (2, 1)). Qual a probabilidade do robô se encontrar no quadrado (1, 3) vinte minutos após o início?**

   Seja $v(t)$ o vetor de probabilidade do robô se encontrar em cada um dos estados após t minutos. O vetor de probabilidade inicial é dado por

$$v(0) = \begin{bmatrix} (1,1) : 0.0000 \\ (1,2) : 0.0000 \\ (1,3) : 0.0000 \\ (1,4) : 0.0000 \\ (2,1) : 1.0000 \\ (2,2) : 0.0000 \\ (2,3) : 0.0000 \\ (2,4) : 0.0000 \\ (3,1) : 0.0000 \\ (3,2) : 0.0000 \\ (3,3) : 0.0000 \\ (3,4) : 0.0000 \\ (4,1) : 0.0000 \\ (4,2) : 0.0000 \\ (4,3) : 0.0000 \\ (4,4) : 0.0000 \end{bmatrix}^T .$$

   A probabilidade do robô se encontrar no quadrado (1, 3) após 20 minutos é dada por

$$v(20) = v(0) \cdot P^{20},$$

   onde P é a matriz de transição de estados do sistema, deduzida no item anterior.

   Utilizando um código em Python para fazer os cálculos, temos:

$$v(20) = \begin{bmatrix} (1,1) : 0.0000 \\ (1,2) : 0.0215 \\ (1,3) : 0.0294 \\ (1,4) : 0.0273 \\ (2,1) : 0.2002 \\ (2,2) : 0.0000 \\ (2,3) : 0.0468 \\ (2,4) : 0.0333 \\ (3,1) : 0.1711 \\ (3,2) : 0.1215 \\ (3,3) : 0.0840 \\ (3,4) : 0.0000 \\ (4,1) : 0.0000 \\ (4,2) : 0.1034 \\ (4,3) : 0.0859 \\ (4,4) : 0.0756 \end{bmatrix}^T$$

Assim, a probabilidade do robô se encontrar no quadrado (1, 3) vinte minutos após o início é de **0.0294**.

3. **O robô está caminhando há muito tempo $t \to \infty$. Onde você apostaria que o robô se encontra?**

   Quando o robô está caminhando por um longo período de tempo, o comportamento dele tende a se estabilizar em um **estado estacionário** à medida que $t \to \infty$. Este estado estacionário representa as probabilidades de o robô estar em cada estado, independentemente de seu estado inicial.

   Para uma matriz de transição P, o vetor de estado estacionário π é aquele que satisfaz a equação:

   $$\pi P = \pi,$$

   ou seja, o vetor de estado estacionário é invariável sob a multiplicação pela matriz de transição.

   O estado estacionário π pode ser obtido de forma numérica resolvendo o sistema de equações $\pi P = \pi$, juntamente com a condição de que a soma das probabilidades no vetor π deve ser igual a 1 ($\sum \pi_i = 1$).

   Para isso, foram implementadas as funções abaixo em Python:

```
    # Função para inicializar o vetor de probabilidade com um 1 aleatório, evitando estados proibidos
    estados_proibidos = [(0, 0), (3, 0), (1, 1), (2, 3)]
    def vetor_inicial_aleatorio(n):
        vetor = np.zeros(n)
        while True:
```

```
            posicao = np.random.randint(0, n)  # Posição aleatória entre 0
                e n-1
            # Verificar se a posição é proibida
            if (posicao // 4, posicao % 4) not in estados_proibidos:
                vetor[posicao] = 1  # Definir a posição aleatória como 1
                break
        return vetor

    # Função para calcular o estado estacionário
    def estado_estacionario(P, tol=1e-6, max_iter=1000):
        n = P.shape[0]
        pi = vetor_inicial_aleatorio(n)
        print(f"pi: {pi}")
        for _ in range(max_iter):
            pi_novo = np.dot(pi, P)
            # Verifica a norma L2 da diferença
            if np.linalg.norm(pi_novo - pi) < tol:
                return pi_novo
            pi = pi_novo
        return pi
```

Utilizando as funções acima, obtemos o vetor de estado estacionário π para a matriz de transição P deduzida no item 1:

$$\pi = \begin{bmatrix} (1,1) : 0.0000 \\ (1,2) : 0.0833 \\ (1,3) : 0.0833 \\ (1,4) : 0.0833 \\ (2,1) : 0.0833 \\ (2,2) : 0.0000 \\ (2,3) : 0.0833 \\ (2,4) : 0.0833 \\ (3,1) : 0.0833 \\ (3,2) : 0.0833 \\ (3,3) : 0.0833 \\ (3,4) : 0.0000 \\ (4,1) : 0.0000 \\ (4,2) : 0.0833 \\ (4,3) : 0.0833 \\ (4,4) : 0.0833 \end{bmatrix}^T$$

Assim, podemos concluir que, após um longo período de tempo, o robô estará em qualquer um dos estados não proibidos com probabilidade de **0.0833**.

4. **Quantos minutos em média leva para o robô retornar ao ponto de partida? É preciso pensar um pouquinho. Proponha uma possível solução a ser discutida na próxima aula. (Para a próxima aula, não precisa resolver esse item, mas será essencial mostrar que você pensou numa solução para apresentar)**

Para isso, precisamos calcular o tempo médio de retorno ao ponto de partida, que é o tempo esperado para o robô retornar ao estado inicial (2, 1) após sair dele.

A probabilidade de o robô estar no estado i no estado estacionário, denotada por π_i, também pode ser vista como a frequência relativa de visitas ao estado i após um longo tempo. Assim, o tempo médio de retorno ao estado i a partir do estado inicial é dado por

$$\mathbb{E}[\tau_i] = \frac{1}{\pi_i},$$

onde τ_i é o tempo de retorno ao estado i a partir do estado inicial.

Como $\pi_{(2,1)}$ foi calculado no item anterior como 0.0833, o tempo médio de retorno ao ponto de partida é de aproximadamente 12 minutos.

Codigos

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