Grupo 15: 87671 - João Freitas; 87693 - Pedro Soares

Inteligência Artificial – Projeto 2 - 2018/2019

1ª Parte - Redes Bayesianas

Descrição crítica dos resultados:

Usando as redes Bayesianas fornecidas na página da cadeira, é possível concluir que os métodos implementados estão corretos dado que o resultado obtido é sempre igual ao valor esperado.

Método computeProb

Este método devolve uma lista com a probabilidade de um nó ser verdadeiro ou falso dada uma certa evidência.

Complexidade: O(k)
K=número de pais

Método computeJointProb

O método usado para calcular probabilidades conjuntas consistiu em:

$$\mathbf{P}(\mathbf{x_1}, \dots, \mathbf{x}_i) = \prod_{i=1}^{n} \boldsymbol{P}(\boldsymbol{x_i} \mid \boldsymbol{parents}(\boldsymbol{x_i}))$$

Complexidade: O(n2^k)
n=número de nós
k=número de pais

Este método não é viável para redes Bayesianas com elevado número de nós, dado que é exponencial com o número de pais. Para melhorar a sua complexidade poderia usar-se o método Noisy-Or:

$$\boldsymbol{P}(\boldsymbol{x_i} \mid \boldsymbol{parents}(\boldsymbol{x_i})) = \mathbf{1} - \prod_{\{j:x_j=true\}} \boldsymbol{q_j}$$

Usando esta forma de calcular probabilidades condicionadas, o método computeJointProb possuiria **complexidade O(nk).**

Método computePostProb

Para calcular probabilidades à posteriori foi utilizado o método de **inferência por enumeração**. Ou seja, sabendo que esta probabilidade é proporcional à soma das probabilidades conjuntas de todas as hipóteses possíveis para a evidência dada, então:

$$\boldsymbol{P}(\boldsymbol{x} \mid \boldsymbol{e}) = \boldsymbol{\alpha}\, \boldsymbol{P}(\boldsymbol{x}, \boldsymbol{e}) = \boldsymbol{\alpha} \sum_{\boldsymbol{y}} \boldsymbol{P}(\boldsymbol{x}, \boldsymbol{e}, \boldsymbol{y})$$

Complexidade: O(n2^n)
n=número de nós

A complexidade poderia ser melhorada usando o **método de eliminação de variáveis** dado que são feitos cálculos repetidos.

Exemplo da rede Bayesiana usada no projeto (inferência por enumeração):

$$\boldsymbol{P(b|j,m)} = \boldsymbol{\alpha} \sum_{e} \sum_{a} \boldsymbol{P(b)P(e)P(a|b,e)P(j,a)\,P(m|a)}$$

Como se pode observar P(b) é constante e pode ser movido para fora dos somatórios, e P(e) pode ser movido para fora do somatório de a.

Exemplo da rede Bayesiana usada no projeto (algoritmo de eliminação de variáveis):

$$\boldsymbol{P(b|j,m)} = \boldsymbol{\alpha\, P(b)} \sum_{e} \boldsymbol{P(e)} \sum_{a} \boldsymbol{P(a|b,e)P(j|a)P(m|a)}$$

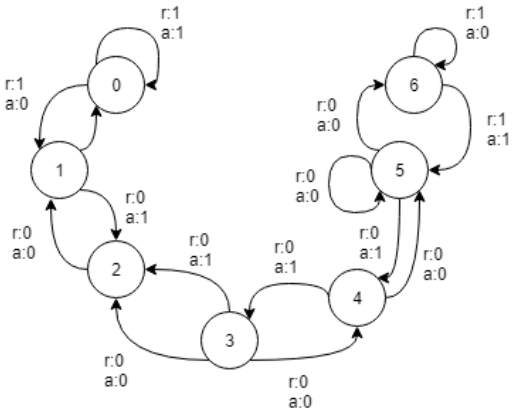
2ª Parte – Q-Learning

Descrição crítica dos resultados:

Usando os testes fornecidos na página da cadeira com 3000 amostras, as aproximações dos valores de Q estavam sempre dentro do valor previsto e foi conseguida a obtenção da trajetória ótima pelo o agente.

Ambiente 1

Representação gráfica:



Função de recompensa R(s,a):

$$R(s,a) = \begin{cases} 0\ se\ s = (1,2,3,4,5), a = (0,1) \\ 1\ se\ s = (0,6)\ , a = (0,1) \end{cases}$$

Desrição da forma como se move o agente:

Durante a análise da trajetória do agente é possível observar-se que ele se move de acordo com um padrão. Sempre que começa o processo de aprendizagem, o agente tenta procurar um estado em que possa permanecer em ciclo de modo a receber sempre recompensa 1. Foi possível tirar esta conclusão ao analisar o output do método runPolicy com uma política de exploitation.

Exemplo de output:

(3,1,2,0)

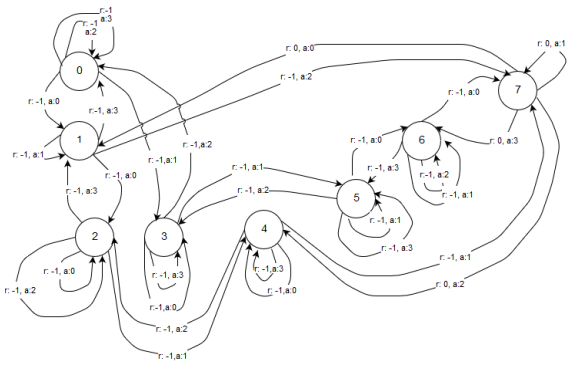
(2,1,1,0)

(1,1,0,0)

A partir deste exemplo observamos que o agente termina no estado 0 de modo a ficar em ciclo para receber recompensa 1.

Ambiente 2

Representação gráfica:



Função de recompensa R(s,a):

$$R(s,a) = \begin{cases} -1 \; se \; s = (0,1,2,3,4,5,6), a = (0,1,2,3) \\ 0 \; se \; s = (7), a = (0,1,2,3) \end{cases}$$

Desrição da forma como se move o agente:

Durante a análise da trajetória do agente é possível observar-se que ele se move de acordo com um padrão. À semelhança do caso do ambiente anterior, o agente tenta procurar um estado em que possa permanecer em ciclo de modo a não ser penalizado (receber recompensa negativa).

Método traces2Q

Este método corresponde à implementação do algoritmo de Q-Learning:

$$\boldsymbol{Q_{t+1}}(\boldsymbol{x},\boldsymbol{a}) = \boldsymbol{Q_t}(\boldsymbol{x},\boldsymbol{a}) + \propto * (\boldsymbol{r} + \boldsymbol{\gamma} * \boldsymbol{max_{a' \epsilon A}} \boldsymbol{Q_t}(\boldsymbol{x}',\boldsymbol{a}') - \boldsymbol{Q_t}(\boldsymbol{x},\boldsymbol{a}))$$

Este algoritmo é corrido em ciclo até haver uma diferença pouco significativa (0.01) entre o valor de Q atual e o seu predecessor.

Complexidade: O(len(trace))
trace=trajetórias

Alternativa:
Uma alternativa ao algoritmo Q-Learning seria SARSA Learning (State-Action-Reward-State-Action), que aplica uma equação semelhante à do Q-Learning, com a exceção de não maximizar o novo $\boldsymbol{Q_t}(\boldsymbol{x}',\boldsymbol{a}')$.
Em SARSA, é usada a mesma política greedy. O que o diferencia é o facto de esperar até uma ação ser realmente tomada e faz o backup do valor Q dessa ação.
O Q-Learning é mais flexível que o SARSA, sendo que um agente em Q-Learning consegue aprender como agir mesmo quando guiado por uma política aleatória, por exemplo. No entanto, o SARSA tem a vantagem de ser mais realista, quando, nomeadamente, a política é parcialmente controlada por outros agentes, é melhor aprender a função Q para aquilo que realmente vai acontecer do que aquilo que o agente gostaria que acontecesse.

Método policy

Este método calcula a política para exploração e para seguir uma política ótima. Para exploração ('exploration'), o método devolve uma ação aleatória para incentivar a exploração do ambiente pelo agente. Para seguir uma política ótima ('exploitation'), o método devolve a ação que lhe garante o maior valor de Q.

Complexidade: O(len(nA))
nA=número de ações

Política ótima

A política ótima em ambos os ambientes, consiste em escolher ação com o maior valor de Q.