Alexandre Dinis Pedro Seixas Caldeira Nº85254
Pedro Mota Veiga Ribeirinho Teixeira Nº84756

Relatório IA: Projecto 2 - IST @ 2018/2019 Grupo 47

Parte 1 - Inferência Exacta em Redes Bayesianas

Descrição dos métodos implementados:

computeProb

O método computeProb começa por verificar se o node não tem pais,para isso usa o tamanho da lista parents , se tal verificar-se ele vai buscar a probabilidade guardada na lista prob e retorna uma lista com 2 elementos correspondendo a False o indice 0 e True o indice 1. Se verificar-se que tem pais, se tiver apenas um vai buscar a probabilidade True tendo em conta o valor da sua evidência e o False terá valor = 1-ProbTrue, sendo que retorna uma lista com o valor de False no índice 0 e o valor True no índice 1. Se tiver mais que um pai vai buscar a lista prob e guarda o índice 1 corresponde a True no índice 1 da lista cprob, e vai tendo em conta os valor das evidências dos pais guardadas no tuplo evid, acedendo dentro da lista de listas até chegar ao final e termos a probabilidade de ser True, a probabilidade False será então 1-ProbTrue e retornamos o lista com a probFalse no índice 0 e a probTrue no índice 1 .

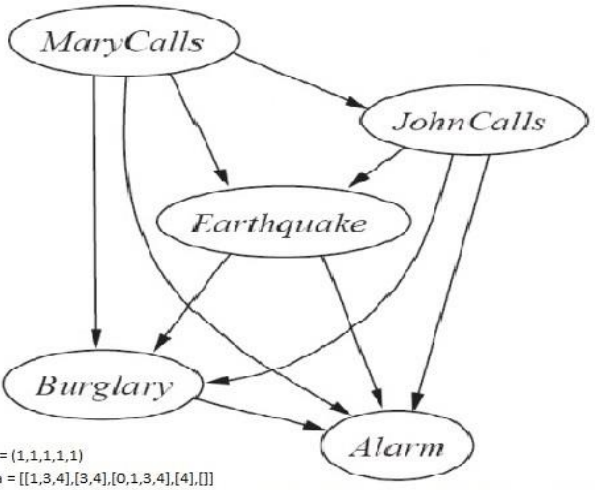
computePostProb

O método computePostProb começa por procurar no tuplo evid o valor do índice do nó igual a -1 que corresponde ao nó de que vamos querer descobrir o valor,e guardamo-lo numa variável.De seguida retiramos quais são os indices dos nós com evidência desconhecida que correspondem às posições do tuplo com uma lista vazia e vamos guarda-los numa lista, e guardamos também de seguida o índice dos restantes nós numa outra lista(os que têm evidência observada).De seguida criamos uma lista m em que guardamos todas as combinações possíveis das evidências dos nós com evidência desconhecida. A seguir usando essa lista criamos todas as combinações de evidências possiveis tendo em conta que apenas variam os indices dos nós com evidência desconhecida, calculamos e vamos multiplicando recursivamente na variavel pt a probabilidade conjunta com o nó a descobrir com evidência a 1(True), e em pf com o nó a descobrir com evidência a 0(False), no final calculamos o alpha que corresponde ao inverso da soma de pt+pf e retornamos o resultado de pt multiplicado pelo alpha.

computeJointProb

O método computeJointProb começa por calcular o tamanho da lista gra que corresponde ao grafo da rede bayesiana para sabermos o número de nós existentes, de seguida cálcula a probabilidade de cada nó com a evidência dada como input e usando o computeProb, sendo que vai multiplicando essa probabilidade pelas anteriores e para tal usa a variável jprob que é retornada no final.

Descrição crítica dos resultados pedidos:




```
ev = (1,1,1,1,1)
gra = [[1,3,4],[3,4],[0,1,3,4],[4],[]]
p1 = Node( np.array([ [[.001,0.21], [0.40,0.43]] , [[0.003,0.25],[0.50,0.33]] ]), gra[0] ) # burglary
p2 = Node( np.array([[.002, 0.23],[0.005, 0.89]]), gra[1] )  # earthquake
p3 = Node( np.array([ [[[.001, 0.46],[0.004, 0.21]], [[0.007, 0.40],[0.002, 0.43]]] ,
[[[0.003, 0.98],[0.006, 0.25]],[[0.001, 0.50],[0.004, 0.33]]] ]), gra[2] )  # alarm
p4 = Node( np.array([.05,.9]), gra[3] )          # johncalls
p5 = Node( np.array([.01,]), gra[4] )            # marycalls
```

Tendo em conta que os outputs do nosso programa deram o esperado nos testes fornecidos fomos testar o nosso programa usando uma ordenação diferente das variáveis da rede bayesiana dada num dos testes com os parâmetros modificados e podemos observar que o nosso programa consegue proceder aos cálculos e construir a rede bayesiana com nós com número de pais superior ao dos testes.

```
p1 false 6.7000e-01 p1 true 3.3000e-01
p1 = 1, p2 = 1, p3 false 6.7000e-01 p3 true 3.3000e-01
sum joint 1.000 (1)
```

Complexidade Computacional e Métodos Alternativos:

Para calcular as probabilidades conjuntas, computeJoint, estamos a usar uma inferência por enumeração em que o algoritmo é linear com o número de variáveis, podemos então verificar que a sua complexidade computacional será sempre $O(2^n)$, para acelarar este processo de cálculo poderiamos usar eliminação de variáveis, em que valores intermédios calculados são guardados para

se evitar voltar a ter de os calcular e desse modo vamos consumir menos recursos e poupar tempo para calcular as probabilidades conjuntas nos métodos computeJoint e computePostProb no caso de termos redes bayesianas com um elevado número de variáveis e relações entre si.

Parte 2 – Aprendizagem por Reforço

Qual é a função de recompensa?

$$1º \text{ exemplo } R(x,a) = \begin{cases} 1 \text{ se } x = 0 \text{ V } x = 6 \\ 0 \text{ cc} \end{cases}$$

Sendo o 2º exemplo demasiado complexo para se definir num sistema de equações, submeto apenas as observações possíveis R(x, 2) = -1, R(x, 0) = -1 para qualquer x pertencente no intervalo [0, nX] sendo o x o correspondente a um estado

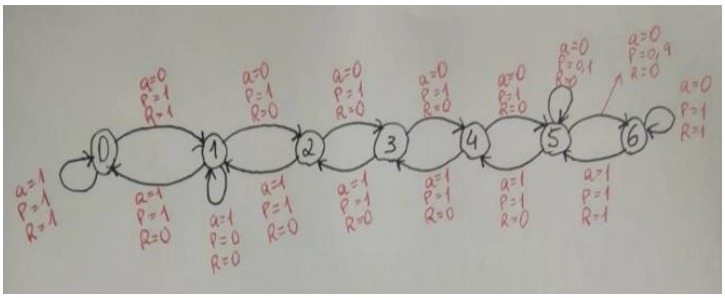
Qual é a política óptima?

A política óptima é baseada num equilibrio entre a "exploration" e a "explotation", ou seja o robot numa fase inicial como desconhece o ambiente em que está inserido e vai adquirir uma política de "exploration" , onde este vai tomar ações aleatorias para ganhar informação sobre as recompensas no trajecto.
Após um número suficiente de tentativas de criação de vários trajectos, o agente já tem informação necessária para alterar a sua política, para uma de "explotation", em que este mediante o estado
em que se encontra, vai agir de acordo com a ação que lhe retorna maior utilidade (recompensa), convergindo desta forma para uma política óptima.

3-Descrição da forma como o agente se move e qual é o impacto de cada ação em cada estado?

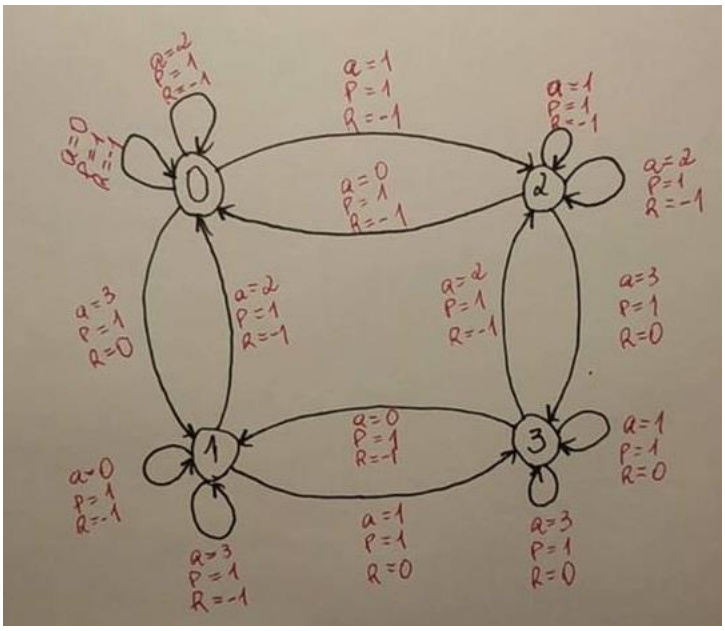
O agente move-se de acordo com a política adquirida, e a função de recompensa é que dita o impacto de cada ação em cada estado.

Exemplo 1:



(Exemplo1: mainRL)

Exemplo 2:



(Exemplo2: RL-TestSet2)

Descrição crítica dos resultados obtidos:

Depois da análise e da execução do código algumas vezes, percebemos que o factor alfa(taxa de aprendizagem) utilizado na equação de aproximação de Q's e o 'n' correspondente ao numero de tentativas estão directamente relacionados, no aspecto em que se reduzir o facto alfa terei que aumentar o 'n' e vice-versa, de outra forma o programa não consegue fazer um bom cálculo de aproximação de Q de forma a encontrar a trajectória óptima. No entanto é importante focar que o numero de tentativas tem sempre que ser alto (>100) senão o agente não tem possibilidade de obter a informação necessária, para poder passar de uma política de "exploration" para uma de "explotation" e por sua vez alcançar uma trajectória óptima.

Alexandre Dinis Pedro Seixas Caldeira Nº85254
Pedro Mota Veiga Ribeirinho Teixeira Nº84756

Métodos implementados:

Q-learning foi o método de apredizagem por reforço implementado, cujo conceito passa por descobrir dentro de uma lista de pares [estado, ação] (matriz Q) qual deles terá maior utilidade para o agente ou seja, qual deles maximiza a recompensa do agente.
A criação desta matriz, é feita na chamada à função "trace2Q", onde após uma inicialização desta matriz a 0's são calculados os valores de Q(estado, ação) através da equação do Q-learning (baseada na equação de Bellman), que fica:

Q(s, a) = Q(s, a) + alfa*(R(s) + gamma*max(Q(s', a')) - Q(s, a))

s - Estado
a - Ação
alfa - Taxa de aprendizagem
R(s) - O rendimento de um determinado estado
gamma - Fator de redução
s´ - Estado seguinte
a' - Ação seguinte

Uma vez que temos a matriz Q completa, para terminar o Q-learning resta apena chamar a função policy que define qual a política necessária, que permite a ação do agente.

Vantagens (+) Desvantanges (-):

Desvantagem: processo muito massivo/inefeciente para um grande número de estados

Vantagem: converge para uma solução óptima

Complexidade computacional:

Função runPolicy - O(N) - sendo N o número de tentativas (1º argumento)

Função traces2Q - O(N * C) - sendo N o número de linha dado no trace2Q e C o número de vezes que o while se repete que depende do problema

Função policy - no melhor caso é O(1) no pior caso é O(size(par[x]))

Alternativas de implementação:

Uma alternativa à implementação feita seria usar a função VI(), que usa uma fórmula diferente de aproximação de Q's mas que retorna directamente qual a política óptima a ser usada através da função Q2pol.