Projecto Inteligência Artificial (LEIC 3^o Ano, 1^o Semestre 2018/2019)

Dúvidas Tagus: andreas.wichert@tecnico.ulisboa.ptluis.sa.couto@tecnico.ulisboa.ptmanuel.lopes@tecnico.ulisboa.pt

Dúvidas Alameda: ernestomorgados@tecnico.ulisboa.pt fausto.almeida@tecnico.ulisboa.pt

22 de Novembro de 2018

Resumo

1 Inferência Exacta em Redes Bayesianas

Neste projecto vamos testar alguns algoritmos que lidam com a incerteza no mundo. Iremos testar métodos de modelação e inferência com redes Bayesianas.

1.1 Bibliografia e ambiente de desenvolvimento

A matéria necessária ao desenvolvimento do projecto pode ser encontrada no livro de texto adoptado Artificial Intelligence a Modern Approach. O projecto deverá ser implementado em Python 3.6.2 usando as funções dadas.

2 Entregas e Prazos

Deverá ser submetido 1 zip (com o nome CGGG em que C é A - Alameda ou T - Tagus, e GGG é o número do grupo) com 2 ficheiros contendo o código de cada problema sem alterar o nome dos ficheiros assim como 1 pdf com o relatório. Os ficheiros de código devem conter em comentário, na primeira linha, os números e os nomes dos alunos do grupo, bem como o número do grupo. Não é necessário incluir os ficheiros disponibilizados pelo corpo docente. O relatório em pdf deverá ter o mesmo nome.

As entregas têm que ser feitas até ao limite definido a seguir, data e hora, não sendo aceites projectos fora de prazo sob pretexto algum.

Entrega - até às 23:59 do dia 07/12/2018

2.1 Condições de realização e discussão dos projectos

Projectos muito semelhantes serão considerados cópia e rejeitados. A detecção de semelhanças entre projectos será realizada utilizando software especializado. Em caso de cópia, todos os alunos envolvidos terão 0 no projecto, serão reprovados na cadeira e referenciados para o conselho pedagógico.

Os trabalhos serão realizados em grupos de 2 pessoas mas cada pessoa deverá ser capaz de explicar todo o trabalho.

Alguns alunos serão chamados, de forma aleatória ou caso seja necessário confirmar a aquisição de competências, **individualmente** para uma discussão oral do trabalho e/ou uma demonstração do funcionamento do programa.

3 Relatório (2+5 valores)

O relatório tem um limite máximo de 3 páginas com duas colunas cada.

4 Redes Bayesianas (10 val)

Para efeitos deste projeto as redes serão acíclicas. Para o processo de inferência o algoritmo de eliminação não é necessário mas discussão da diferença entre eles. Os testes, os fornecidos e outros, irão incluir no máximo 10 nós.

4.1 Estrutura da Rede

Uma rede pode ser definida como um grafo da seguinte forma:

```
gra = [[],[],[0,1],[2],[2]]
```

Com uma lista em que cada elemento representa os pais de cada uma das variáveis.
Este é o exemplo da rede Bayesiana da pag. 512 do livro [1].

4.2 Nós (2 val)

Definir uma classe para cada nó da rede usando o seguinte interface.

```
class Node():
    def __init__(self, prob, parents = []):
        self.parents = ...
        self.prob = ...
...

    def computeProb(self, evid):
...
```

Exemplo de utilização:

```
p1 = Node( np.array([.001]), [] )    # burglary
print(p1.computeProb(ev))
[0.999, 0.001]


p3 = Node( np.array([[.001,.29],[.94,.95]]), [0,1] )    # alarm
print(p3.computeProb(ev))
ev = (0,0,1,1,1)
[0.999, 0.001]
print(p3.computeProb(ev))
ev = (0,1,1,1,1)
print(p3.computeProb(ev))
[0.70999999999999996, 0.28999999999999998]
ev = (1,0,1,1,1)
print(p3.computeProb(ev))
[0.060000000000000053, 0.93999999999999995]
ev = (1,1,1,1,1)
print(p3.computeProb(ev))
[0.050000000000000044, 0.94999999999999996]
```

No caso de uma variável ter um pai apenas define-se apenas o valor positivo. Se uma variável tiver mais do que um pai inclui-se como parâmetros um array multidimensional. No exemplo anterior o vector evidência representa a informção que temos sobre cada uma das variáveis (0 - false, 1 - true). A função computeProb tem em conta a evidência dos nós pai para devolver os valores de probabilidade do nó ser false e de ser true.

4.3 Rede Bayesiana (4 val)

Temos agora todos os componentes para definir uma rede Bayesiana. Definir a seguinte classe:

```
class BN():
    def __init__(self, gra, prob):
      ...

    def computePostProb(self, evid):
...

    def computeJointProb(self, evid):
...
```

Que tem como elementos o grafo e as respectivas probabilidades.

```
gra = [[],[],[0,1],[2],[2]]

p1 = Node( np.array([.001]), gra[0] )                    # burglary
print( "%.4e" % p1.computeProb(ev)[0])

p2 = Node( np.array([.002]), gra[1] )                    # earthquake

p3 = Node( np.array([[.001,.29],[.94,.95]]), gra[2] )    # alarm
print( "%.4e" % p3.computeProb(ev)[0])

p4 = Node( np.array([.05,.9]), gra[3] )                  # johncalls

p5 = Node( np.array([.01,.7]), gra[4] )                  # marycalls
prob = [p1,p2,p3,p4,p5]

gra = [[],[],[0,1],[2],[2]]
bn = BN(gra, prob)
```

Ter-se-á de definir dois outros métodos. `computeJointProb` que dado uma evidência (sem valores desconhecidos) calcula a probabilidade conjunta (2 val), e `computePostProb` que dada uma evidência calcula a probabilidade a-posteriori de uma variável (4 val).

Iremos usar como notaçcão na evidência:

- 0 - false; 1 - true
- [] para indicar desconhecido
- -1 para indicar a variável para a qual se quer calcular a posterior (só pode haver uma variável a -1)

Exemplo de utilizaçcão:

```
ev = (1,1,1,1,1)
jp = []
for e1 in [0,1]:
    for e2 in [0,1]:
        for e3 in [0,1]:
            for e4 in [0,1]:
                for e5 in [0,1]:
                    jp.append(bn.computeJointProb((e1, e2, e3, e4, e5)))

print("sum joint %.3f (1)" % sum(jp))
sum joint 1.000 (1)

ev = (-1,[],[],1,1)
print("ev : ")
print(ev)
print( "post : %.4g (0.2842)" % bn.computePostProb(ev)  )
ev :
(-1, [], [], 1, 1)
post : 0.2842 (0.2842)

ev = ([],-1,[],1,1)
print("ev : ")
print(ev)
print( "post : %.3f (0.176)" % bn.computePostProb(ev)  )
ev :
([], -1, [], 1, 1)
post : 0.176 (0.176)

ev = ([],0,1,-1,[])
print("ev : ")
print(ev)
print( "post : %.3f (0.900)" % bn.computePostProb(ev)  )
ev :
([], 0, 1, -1, [])
post : 0.900 (0.900)
```

4.4 Relatório (2 val)

No relatório deve incluir-se:

- descrição crítica dos resultados pedidos
- descrição dos métodos implementados incluindo vantagens/desvantagens e limitações
- discussão da complexidade computacional e possíveis métodos alternativos

5 P2 - Aprendizagem por Reforço (10 valores)

Vamos neste exercício aprender por interação com o mundo. Vamos imaginar que um robot interagiu com o mundo recebendo diferentes recompensas.

Essas recompensas estão disponíveis num ficheiro onde cada linha contem o estado inicial, a ação executada, o estado seguinte e a recompensa. Como é uma sequência o estado seguinte numa linha é o inicial na linha seguinte. Pela interação do agente com o mundo poderíamos perceber a forma do mundo em que ele vive. Temos no entanto que responder as seguintes perguntas.

- Qual é o valor de cada ação em cada estado do mundo?
- Qual é a política a seguir em cada estado?
- Seguindo a política aprendida para onde é que o agente vai se começar no estado 3?

As funções a realizar deverão ser implementadas em RLsol.py, este ficheiro já tem algumas funções que não deverão ser alteradas. O script mainRL.py testa algumas das funcionalidades.

5.1 QLearning (3 valor)

Implementar o algoritmo Q-Learning que a partir de uma trajetória recebida calcule os valores Q para cada ação. Essa funcionalidade deverá ser implementada na função traces2Q. Esta função recebe um traço de trajetória em que cada linha inclui (estado inicial, ação, estado final, recompensa). A função deverá retornar uma aproximação da função Q.

```
class myRL:

    def __init__(self, nS, nA, gamma):
        self.nS = nS
        self.nA = nA
        self.gamma = gamma
        self.Q = np.zeros((nS,nA))

    def traces2Q(self, trace):
        # implementar esta funcao
        self.Q = np.zeros((self.nS,self.nA))

        return self.Q
```

5.2 Q2Pol (1 val)

Para calcular a trajetória é necessário uma política. É necessário implementar a função:

```
    def policy(self, x, poltype = 'exploration', par = []):
        # implementar esta funcao

        if poltype == 'exploitation':
            a =

        elif poltype == 'exploration':
            a =

        return a
```

Esta função deverá ser capaz de calcular uma política para exploração e para seguir a politica óptima (exploitation). Esta função irá ser usada para gerar trajetórias.

5.3 Gerar trajetória (1 valor)

Após gerar novos dados Com a política de exploração definida podemos gerar novos dados:

```
J,traj = fmdp.runPolicy( <escolher numero de amostras> ,3,poltype = "exploration")
```

Onde J é a recompensa acumulado e traj é o trace da trajetória com o formato descrito anteriormente. Para os quais podemos aprender os valor Q respectivos `Qr = fmdp.traces2Q(traj)` e gerar a trajetória óptima para um dado estado inicial usando a política de exploitation `J,traj = fmdp.runPolicy(3,3,poltype = "exploitation", polpar = Qr)`

5.4 Execução do código.

Está disponibilizado um exemplo de teste podendo haver outros testes parecidos na avaliação final. Ao passar todos os testes deverá ler-se:

```
exercicio 1
Aproximação de Q dentro do previsto. OK

Trajetória óptima. OK

exercicio 2
Aproximação de Q dentro do previsto. OK
```

5.5 Relatório (5 val)

No relatório é necessário:

Para cada um dos ambientes, e por inspecção das trajectórias, fazer uma representação gráfica do ambiente no qual o agente se move (1val). Descrever qual a função de recompensa? (1val) Qual é a politica óptima? (1val) Descrição do forma como o agente se move (Qual é o impacto de cada acção em cada estado)? (1val)

No relatório deve incluir-se também:

- descrição crítica dos resultados pedidos
- descrição dos métodos implementados incluindo vantagens/desvantagens e limitações
- discussão da complexidade computacional e possíveis métodos alternativos

Referências

[1] Staurt J Russell and Peter Norvig. Artificial intelligence (a modern approach) 3rd Edition, 2010.