Ciência de Computadores

Sistemas Inteligentes

Segundo teste (duração: 1 hora)

Data: 3 de Junho de 2013

1) Qual das seguintes alternativas melhor descreve a forma como as probabilidades **a priori** são obtidas numa rede Bayesiana:

(a) as probabilidades **a priori** são calculadas utilizando a fórmula de Bayes

(b) as probabilidades **a priori** são calculadas a partir de observações

(c) as probabilidades **a priori** são calculadas a partir de dados da literatura

(d) as probabilidades **a priori** são calculadas a partir de observações ou a partir de dados da literatura

(e) nenhuma das alternativas descreve a melhor forma de se obter as probabilidades **a priori** de uma rede Bayesiana

2) O resultado da unificação entre os termos `mais(X,Mais(Y,Z))` com o termo `mais(2,Mais(5,T))`, em Prolog, onde os nomes de variáveis são iniciados por letra maiúscula e as constantes por letra minúscula, é:

(a) `X=2, Y=5, Z=T`

(b) `X=2, Y=Mais(5,T), Z=T`

(c) `X=2, Y=5, Z=Mais(5,T)`

(d) nenhuma das alternativas acima está correta

3) Qual é a melhor interpretação para o programa abaixo:

```
p(X,cons(X,Y)).

p(X,cons(Y,Z)) :- p(X,Z).
```

(a) p é um predicado que implementa o conceito de último elemento de uma lista

(b) p é um predicado que implementa o conceito de pertinência de um elemento a uma lista

(c) p é um predicado que implementa o conceito de remoção de um elemento de uma lista

(d) p é um predicado que implementa o conceito de remoção de vários elementos de uma lista

(e) nenhuma das alternativas anteriores corresponde à interpretação de p

4) Dada a fórmula de cálculo de ganho de informação abaixo, indique qual é o atributo na Tabela 1 que será escolhido como raiz da árvore de decisão induzida.

$$\text{ganho de informação}(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - Restante(A)$$

$$Restante(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p+n} I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i})$$

$$I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n} log_2 \frac{p}{p+n} - \frac{n}{p+n} log_2 \frac{n}{p+n}$$

v é o número de valores do atributo A, p é o número de instâncias de uma classe e n é o número de instâncias da outra classe.

Tabela 1: Tabela de atributos do problema de indução de árvore de decisão

Instância	A1	A2	A3	A4	Classe
1	0	0	1	0	1
2	0	0	1	0	0
3	1	0	0	1	0
4	1	0	1	0	0
5	0	1	0	1	1

(a) A1

(b) A2

(c) A3

(d) A4

(e) todos os atributos têm a mesma chance de serem escolhidos como raiz da árvore induzida

5) No seguinte resultado produzido pelo pacote WEKA, usou-se um algoritmo de árvore de decisão para gerar um classificador de tipos de plantas. As plantas podem ter tipo T1, T2 ou T3.

```
 a  b  c   <-- classified as
49  1  0 |  a = T1
 0 47  3 |  b = T2
 0  2 48 |  c = T3
```

O que significam os números da matriz de confusão acima gerada pelo algoritmo?

(a) os erros cometidos pelo classificador no conjunto de treino

(b) os acertos cometidos pelo classificador no conjunto de treino

(c) os erros e os acertos cometidos pelo classificador no conjunto de treino

(d) nenhuma das alternativas anteriores está correta

6) O algoritmo *random restart hill climbing*:

(a) nunca encontra uma solução ótima para um problema

(b) sempre encontra uma solução ótima para um problema

(c) eventualmente vai encontrar uma solução ótima para um problema

(d) pode ou não encontrar uma solução ótima para um problema

7) A diferença entre o algoritmo *hill climbing* e o *simulated annealing* é:

(a) *hill climbing* aceita uma solução pior se souber que mais tarde pode enontrar uma solução ótima

(b) *simulated annealing* aceita uma solução pior com uma certa probabilidade

(c) *hill climbing* aceita uma solução pior com uma certa probabilidade

(d) não há diferenças fundamentais entre estes dois algoritmos

8) No mundo dos blocos, dado o estado inicial:

```
%
%
%    +-+   +-+  +-+
%    |c|   |b|  |a|
%    +-+   +-+  +-+
%    |e|   |d|  |f|
%----+-+---+-+--+-+----------------- floor
```

e o estado final:

```
%
%    +-+    +-+
%    |e|    |b|
%    +-+    +-+
%    |a|    |c|
%----+-+----+-+--------------------- floor
```

que tipo de ações poderia utilizar para construir um plano que leve um agente a chegar do estado inicial ao final:

(a) `goto(x,y)`, onde x é o lugar inicial e y é o lugar de destino. Com pré-condição: canGo(x,y) and at(x) e pós-condição: at(y)

(b) `pickup(u,x)`, onde u é um bloco localizado em x. Com pré-condição: clean(u) and at(u,x) e pós-condição: at(u,OtherPlace)

(c) `putdown(u,x)`, onde u é um bloco depositado no local x. Com pré-condição: holding(u) and at(x) e pós-condição: notHolding(u) and at(x)

(d) `move(b,x,y)`, mover bloco b do local x para o local y. Com pré-condição: at(b,x) and clean(b) and clean(y) e pós-condição: on(y,b) and clean(x)

(e) nenhum dos tipos anteriores me faz gerar um plano adequado a este problema

9) Dados os pseudo-códigos abaixo, qual deles implementa corretamente o algoritmo de poda alfa-beta, relativamente ao procedimento de MAX?

(a)

```
function MAX-VALUE(state,alfa,beta) returns a utility value
   inputs: state -> estado corrente no jogo
           alfa  -> valor da melhor alternativa para MAX
           beta  -> valor da melhor alternativa para MIN
   if TERMINAL_TEST(state) then return UTILITY(state)
   v <- +infinito
   for a, s in SUCCESSORS(state) do
     v <- MAX(v, MIN-VALUE(s,alfa,beta))
     if (v >= beta ) then return v % momento da poda
     alfa <- MAX(alfa,v)
   return v
```

(b)

```
function MAX-VALUE(state,alfa,beta) returns a utility value
   inputs: state -> estado corrente no jogo
           alfa  -> valor da melhor alternativa para MAX
           beta  -> valor da melhor alternativa para MIN
   if TERMINAL_TEST(state) then return UTILITY(state)
   v <- +infinito
   for a, s in SUCCESSORS(state) do
     v <- MAX(v, MIN-VALUE(s,alfa,beta))
     if (v <= beta ) then return v % momento da poda
     alfa <- MAX(alfa,v)
   return v
```

(c)

```
function MAX-VALUE(state,alfa,beta) returns a utility value
   inputs: state -> estado corrente no jogo
           alfa  -> valor da melhor alternativa para MAX
```

```
        beta  -> valor da melhor alternativa para MIN
   if TERMINAL_TEST(state) then return UTILITY(state)
   v <- -infinito
   for a, s in SUCCESSORS(state) do
     v <- MAX(v, MIN-VALUE(s,alfa,beta))
     if (v >= beta ) then return v % momento da poda
     alfa <- MAX(alfa,v)
   return v
```

(d)

```
function MAX-VALUE(state,alfa,beta) returns a utility value
   inputs: state -> estado corrente no jogo
           alfa  -> valor da melhor alternativa para MAX
           beta  -> valor da melhor alternativa para MIN
   if TERMINAL_TEST(state) then return UTILITY(state)
   v <- -infinito
   for a, s in SUCCESSORS(state) do
     v <- MAX(v, MIN-VALUE(s,alfa,beta))
     if (v >= alfa ) then return v % momento da poda
     beta <- MAX(beta,v)
   return v
```

(e) nenhuma das alternativas anteriores implementa corretamente o passo de MAX da poda alfa-beta

10) Num algoritmo genético, a operação de *crossover*:

(a) sempre aumenta o número de indivíduos da população original

(b) sempre cruza dois indivíduos gerando dois descendentes

(c) pode cruzar vários indivíduos e gerar vários descendentes

(d) pode cruzar dois indivíduos gerando mais que dois descendentes

(e) a operação de *crossover* não faz nada do que é dito nas alternativas anteriores