COMPITO DI APPLICAZIONI DI INTELLIGENZA ARTIFICIALE

16 settembre 2005 (Punteggio su 30/30; Tempo 2h)

Esercizio 1 (punti 8)

Dato il seguente training set S:

Allegati	**HTML**	**Classe**
3	Sì	Spam
1	?	NoSpam
2	Sì	Spam
3	No	Spam
2	No	NoSpam
1	Sì	Spam
1	No	NoSpam
2	Sì	Spam
1	?	Spam
3	No	NoSpam
3	No	NoSpam
2	Sì	Spam
1	Sì	NoSpam

a) Si calcoli l'entropia del training set rispetto all'attributo Classe (punti 1)
b) Si calcoli il guadagno dei due attributi rispetto a questi esempi di training (punti 4)
c) si costruisca un albero decisionale ad un solo livello per il training set dato, indicando le etichette delle foglie (numero di esempi finiti nella foglia/numero di esempi finiti nella foglia non appartenenti alla classe della foglia). (punti 1,5)
d) si classifichi l'istanza: (punti 1,5)

1	?

Esercizio 2 (punti 8)

Il SAT è un problema costituito da un insieme di variabili booleane (0 - 1) soggette ad un insieme di vincoli, espressi in forma a clausole. Ogni clausola è un OR di letterali, ciascun letterale è una variabile o una variabile negata. Ad esempio:

$$X \vee \neg Y \vee \neg Z \qquad\qquad \neg X \vee Y \vee \neg Z$$

è un problema SAT. Una soluzione è un assegnamento alle variabili per cui tutte le clausole sono soddisfatte (ad es X=1, Y=1, Z=0).

Il MAX-SAT è un problema in cui si cerca di massimizzare il numero delle clausole soddisfatte di un SAT. Si scriva un programma CLP che calcola il numero massimo di clausole soddisfatte. Si supponga che il predicato venga invocato con i seguenti parametri:

maxsat(*Clausole*,*Variabili*,*N*)

dove *Clausole* è una lista di clausole, ciascuna delle quali è una variabile che rappresenta un letterale; *Variabili* è la lista delle variabili ed *N* è il numero di clausole soddisfatte (in output).

Esempio:

```
:- A #= 1-NA, % Na è la variabile corrispondente ad A negato
   B #= 1-NB, maxsat([[A,B],[NA,NB],[A,NB],[NA,B]],[A,B],N).
yes  NA = 1 NB = 1 A = 0 B = 0 N = 3
```

Esercizio 3 (Punti 8)

Si consideri uno stato iniziale descitto dalle seguenti formule atomiche:
`[state(camera_one,on), state(camera_two,on), state(camera_three,on), in(camera_one, roomB), in(camera_two, roomB), in(camera_three, roomB), object_in(oggetto1,roomA)]`
Inoltre lo stato iniziale contiene predicati diverso per ogni coppia di parametri che rappresentano una stanza o una macchina fotografica. Es. diverso(camera_one, camera_two)

Si supponga di voler raggiungere il goal:
take_picture(oggetto1)

Le azioni sono modellate opportunamente come segue:
make_photo(Object)
PRECOND: object_in(Object, Room), in(CameraX,Room), in(CameraY,Room), state(CameraX, on), state(CameraY, on), diverso(CameraX,CameraY)
EFFECT: take_picture(Object)

move_camera(Camera,Room1,Room2)
PRECOND: in(Camera,Room1)
EFFECT: in(Camera,Room2), not in(Camera,Room1)

Si risolva il problema utilizzando l'algoritmo STRIPS.

Esercizio 4 (Punti 6)

Cosa e' la teta sussunzione?

Perche' la pianificazione deduttiva puo' essere inefficiente?

Quali sono i passi fondamentali della macchina astratta CLP?

SOLUZIONE

Esercizio 1:

a) info(S)=-6/13*$\log_2$ 6/13-7/13*$\log_2$ 7/13=0.996

b)

$info_{Allegati}$(S)=4/13*(-2/4*$\log_2$2/4-2/4*$\log_2$ 2/4)+4/13*(-1/4*$\log_2$1/4-3/4*$\log_2$ 3/4)+5/13*(-3/5*$\log_2$ 3/5-2/5*$\log_2$ 2/5)=

=0.308*1+0.308*0.811+0.385*0.971=0.932

gain(Allegati) =0.996-0.932=0,064

splitinfo(Allegati)=-4/13*$\log_2$(4/13)-4/13*$\log_2$(4/13)-5/13*$\log_2$(5/13)=1.577

gainratio(Allegati) =0.064/1.577=0,041

Per caclolare il guadagno dell'attributo HTML non si usa l'entropia calcolata su tutto il training set ma solo sugli esempi che hanno HTML noto (insieme F):

info(F)=-5/11*$\log_2$ 5/11– 6/11*$\log_2$ 6/11=0.994

$info_{HTML}$(F)=5/11*(-4/5*$\log_2$4/5-1/5*$\log_2$ 1/5)+6/11(-1/6*$\log_2$ 1/6-5/6*$\log_2$ 5/6) =

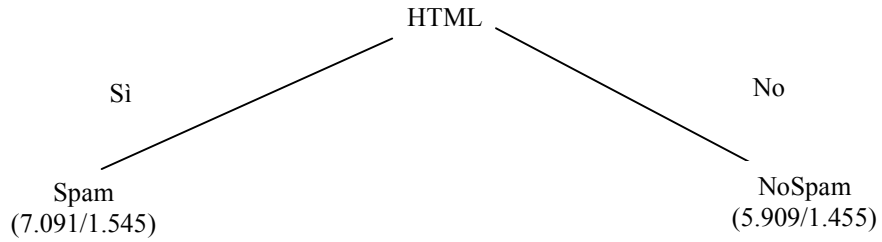
=0.455*0.722+0.545*0.650=0.683

gain(HTML) =11/13*(0.994-0.683)=0.263

splitinfo(HTML) =-6/13*$\log_2$(6/13)-5/13*$\log_2$(5/13)-2/13*$\log_2$(2/13)=1.460

gainratio(HTML) =0.263/1.460=0.180

c)



d) l'istanza viene divisa in due parti, una di peso 7.091/13=0.545 e l'altra di peso 5.909/13=0.455. La prima parte viene mandara lungo il ramo Sì e viene classificata come Spam con probabilita' 5.546/7.091 =78.2% e come NoSpam con probabilita' 1.545/7.091=21.8%. La seconda parte viene mandata lungo il ramo No e viene classificata come NoSpam con probabilita' 4.454/5.909=75.4% e come Spam con probabilità 1.455/5.909=24.6%. Quindi in totale la classificazione dell'istanza è

Spam: 0.545*78.2%+0.455*24.6%=53.8%

NoSpam: 0.545*21.8% + 0.455*75.4%=46.2%

Esercizio 2:

```
:- lib(fd).
:- lib(fd_global).

maxsat(Clausole,Variabili,Soddisfatte):-
    Variabili :: 0..1,     % Domini delle variabili
    length(Clausole,NumClausole),     % Calcolo quante clausole ci sono
    Soddisfatte #= NumClausole - Costo, % Funzione obiettivo:
                               % minimizzare le clausole insoddisfatte
    imponi_clausole(Clausole,ListaCosti),
     % data la lista dei costi delle singole clausole, calcolo
     % il costo totale
    sumlist(ListaCosti,Costo),
    minimize(labeling(Variabili),Costo).

imponi_clausole([],[]).
imponi_clausole([Clausola|Clausole],[Costo|Costi]):-
% Una clausola e` soddisfatta se almeno uno dei suoi letterali vale 1
% quindi se la somma dei valori dei letterali e` maggiore di zero.
    sumlist(Clausola,Sum),
% Se la somma e` uguale a zero ho un costo di 1
% altrimenti ho un costo di 0 per quella clausola
    Sum #= 0 #<=> Costo,
    imponi_clausole(Clausole,Costi).
```

Esercizio 3:

STATO	GOAL
`state(camera_one,on)`	`take_picture(oggetto1)`
`state(camera_two,on)`	
`state(camera_three,on)`	
`in(camera_one, roomB)`	
`in(camera_two, roomB)`	
`in(camera_three, roomB)`	
`object_in(oggetto1,roomA)`	

Inserisco l'azione make_photo

STATO	GOAL
`state(camera_one,on)`	`object_in(oggetto1,Room)`
`state(camera_two,on)`	`in(CameraX,Room)`
`state(camera_three,on)`	`in(CameraY,Room)`
`in(camera_one, roomB)`	`diverso(CameraX,CameraY)`
`in(camera_two, roomB)`	`state(CameraX,on)`
`in(camera_three, roomB)`	`state(CameraY,on)`
`object_in(oggetto1,roomA)`	`make_photo(oggetto1)`

Room / roomA

STATO	GOAL
`state(camera_one,on)`	
`state(camera_two,on)`	`in(CameraX,roomA)`
`state(camera_three,on)`	`in(CameraY,roomA)`
`in(camera_one, roomB)`	`diverso(CameraX,CameraY)`
`in(camera_two, roomB)`	`state(CameraX,on)`
`in(camera_three, roomB)`	`state(CameraY,on)`
`object_in(oggetto1,roomA)`	`make_photo(oggetto1)`

Uso l'azione move

STATO	GOAL
`state(camera_one,on)`	`in(CameraX,RoomX)`
	`move(CameraX,RoomX,roomA)`
`state(camera_two,on)`	`in(CameraX,roomA)`
`state(camera_three,on)`	`in(CameraY,roomA)`
`in(camera_one, roomB)`	`diverso(CameraX,CameraY)`
`in(camera_two, roomB)`	`state(CameraX,on)`
`in(camera_three, roomB)`	`state(CameraY,on)`
`object_in(oggetto1,roomA)`	`make_photo(oggetto1)`

CameraX / camera_one RoomX / roomB

STATO	GOAL
`state(camera_one,on)`	
	`move(camera_one,roomB,roomA)`
`state(camera_two,on)`	`in(camera_one,roomA)`
`state(camera_three,on)`	`in(CameraY,roomA)`
`in(camera_one, roomB)`	`diverso(camera_one,CameraY)`
`in(camera_two, roomB)`	`state(camera_one,on)`
`in(camera_three, roomB)`	`state(CameraY,on)`
`object_in(oggetto1,roomA)`	`make_photo(oggetto1)`

Effettuo l'azione move

STATO	GOAL
`state(camera_one,on)`	
`state(camera_two,on)`	`in(camera_one,roomA)`
`state(camera_three,on)`	`in(CameraY,roomA)`
`in(camera_one, roomA)`	`diverso(camera_one,CameraY)`
`in(camera_two, roomB)`	`state(camera_one,on)`
`in(camera_three, roomB)`	`state(CameraY,on)`
`object_in(oggetto1,roomA)`	`make_photo(oggetto1)`

in(camera_one,roomA) e' vero nello stato iniziale e inserisco la azione move

STATO	GOAL
`state(camera_one,on)`	`in(CameraY,RoomX)`
`state(camera_two,on)`	`move(CameraY,RoomX,roomA)`
`state(camera_three,on)`	`in(CameraY,roomA)`
`in(camera_one, roomA)`	`diverso(camera_one,CameraY)`
`in(camera_two, roomB)`	`state(camera_one,on)`
`in(camera_three, roomB)`	`state(CameraY,on)`
`object_in(oggetto1,roomA)`	`make_photo(oggetto1)`

Unifico CameraY / camera_two e RoomX / roomB

STATO	GOAL
`state(camera_one,on)`	
`state(camera_two,on)`	`move(camera_two,roomB,roomA)`
`state(camera_three,on)`	`in(camera_two,roomA)`
`in(camera_one, roomA)`	`diverso(camera_one, camera_two)`
`in(camera_two, roomB)`	`state(camera_one,on)`
`in(camera_three, roomB)`	`state(camera_two,on)`
`object_in(oggetto1,roomA)`	`make_photo(oggetto1)`

Effettuo l'azione move

STATO	GOAL
`state(camera_one,on)`	
`state(camera_two,on)`	
`state(camera_three,on)`	`in(camera_two,roomA)`
`in(camera_one, roomA)`	`diverso(camera_one, camera_two)`
`in(camera_two, roomA)`	`state(camera_one,on)`
`in(camera_three, roomB)`	`state(camera_two,on)`
`object_in(oggetto1,roomA)`	`make_photo(oggetto1)`

Tutte le precondizioni prima di make_photo sono vere nello stato iniziale

STATO	GOAL
`state(camera_one,on)`	`make_photo(oggetto1)`
`state(camera_two,on)`	
`state(camera_three,on)`	
`in(camera_one, roomA)`	
`in(camera_two, roomA)`	
`in(camera_three, roomB)`	
`object_in(oggetto1,roomA)`	

Effettuo make_photo e ottengo il goal

STATO	GOAL
`state(camera_one,on)`	
`state(camera_two,on)`	
`state(camera_three,on)`	
`in(camera_one, roomA)`	
`in(camera_two, roomA)`	
`in(camera_three, roomB)`	
`object_in(oggetto1,roomA)`	
`take_picture(oggetto1)`	