ProbLog exercise

Luc De Raedt and Robin Manhaeve

WASP Course AI and ML

In these exercises, we will build a ProbLog model that encodes a game of Poker. We will do this step by step, by first constructing a Bayesian network. Next, we will use Prolog to encode the logic of poker hands. In part 2 that willl follow later (after handing your solutions to assignment 1), we will combine both into a ProbLog model that can perform complex inference on this probabilistic logic program.

Setup

We will use ProbLog throughout this exercise. You can try out ProbLog in the online editor here: https://dtai.cs.kuleuven.be/problog/editor.html. You can also install ProbLog locally by running the following command in your terminal: `pip install problog`. This requires Python >= 3.6 to be installed. For further information on installing and using ProbLog, please visit https://github.com/ML-KULeuven/problog. We have included automated tests to help you check your code. For this, ProbLog needs to be installed locally. You can then run the test suite by running the following command in your terminal: `python test.py`.

1 Bayesian networks

You are player 1 in a two player card game. You and your opponent each draw a card from your personal infinite stack of cards that contain an equal number of jacks (J), queens (Q), kings (K), and aces (A). The person who draws the highest card wins. If there is a tie, a coin gets flipped. If the coin comes up heads, player 1 (you) win. If it comes up tails, the opponent wins. You are playing fair, but you suspect there is a 1/5 chance that your opponent is cheating. A cheating opponent has replaced all the jack cards with ace cards. A cheating opponent also replaces the coin with a coin that always comes up tails.

- Encode this game as a Bayesian network using the following random variables: Draw1, Draw2, Highest, Coin, Winner, Cheater. Draw the graph and define the conditional probability table for each node.
- Is Draw1 marginally independent of Coin ? (Yes / No)
- Is Draw1 marginally independent of Coin given that we know Winner ? (Yes / No)
- Encode the same Bayesian network as a ProbLog program. What is the probability that player 1 wins? What is the probability that player 2 wins?
- Given the observations in Table 1, learn the probability that player 2 is a cheater (keep the other parameters fixed). Use the learning tab from the online editor to do this. What is the final probability?

Draw1	Winner
jack	player2
ace	player1
ace	player2
king	player1
queen	player2

Table 1: Game data for Exercise 1.

2 Prolog

Note that in the Bayesian network example, the Highest and Winner variables actually encode a deterministic reasoning step. We will now encode a more complex version of this problem. Instead of only one card, the players now have multiple cards. Additionally, instead of only reasoning about the high cards, we also reason about: one pair, two pair, three of a kind and straight as defined here: https://en.wikipedia.org/wiki/List_of_poker_hands. Note that the rank of the card is important when comparing two hands of the same type, i.e. a pair of queens is better than a pair of jacks.

Part of the code is given in `exercise2.pl`. Complete the code by implementing the following predicates:

- `hand(Cards, Hand)`: Unifies Hand with all possible hands that can be created from the given Cards.
- `better(BetterHand, WorseHand)`: Succeeds if BetterHand wins over WorseHand in the rules of poker.

```
%%%% Implement the following predicates.

hand(Cards,Hand).

%% Example for pair
% hand(Cards,pair(Rank)) :-
%     select(Rank,Cards,Cards2),
%     member(Rank,Cards2).


better(BetterHand,WorseHand).


%%%% Provided code

:- use_module(library(lists)).

game_outcome(Cards1,Cards2,Outcome) :-
    best_hand(Cards1,Hand1),
    best_hand(Cards2,Hand2),
    outcome(Hand1,Hand2,Outcome).

outcome(Hand1,Hand2,player1) :- better(Hand1,Hand2).
outcome(Hand1,Hand2,player2) :- better(Hand2,Hand1).
outcome(Hand1,Hand2,tie) :- \+better(Hand1,Hand2), \+better(Hand2,Hand1).

best_hand(Cards,Hand) :-
    hand(Cards,Hand),
    \+ (hand(Cards,Hand2), better(Hand2,Hand)).
```

3 ProbLog

Adapt and combine the Bayesian neural network from Exercise 1 and the Prolog program from Exercise 2 to encode the game where each player now draws 4 cards from their own stack. Use the `game_outcome/3` predicate to determine who has the best hand. In the case of a tie, the coin is used to determine the winner. The chance that the opponent is a cheater, and its impact on the game remains the same. The code from Exercise 1 will need to be slightly adapted. Complete the following program (`exercise3.pl`):

```
%%%% Insert and modify the ProbLog code from Exercise 1 here

% Encode the different cards as follows: card(Player,N,Rank)
% This means that the N-th card drawn by Player is of the given Rank.


%%%% Insert Prolog code from Exercise 2

hand(Cards,Hand).
better(BetterHand,WorseHand).

%%%% Provided code
:-use_module(library(apply)).
:-use_module(library(lists)).

% The following predicate will sample a Hand as a list of ranks for the given player.
% It expects that there are probabilistic facts of the form card(Player,N,Rank) as specified above

draw_hand(Player,Hand) :- maplist(card(Player),[1,2,3,4],Hand).

game_outcome(Cards1,Cards2,Outcome) :-
    best_hand(Cards1,Hand1),
    best_hand(Cards2,Hand2),
    outcome(Hand1,Hand2,Outcome).

outcome(Hand1,Hand2,player1) :- better(Hand1,Hand2).
outcome(Hand1,Hand2,player2) :- better(Hand2,Hand1).
outcome(Hand1,Hand2,tie) :- \+better(Hand1,Hand2), \+better(Hand2,Hand1).

best_hand(Cards,Hand) :-
    hand(Cards,Hand),
    \+ (hand(Cards,Hand2), better(Hand2,Hand)).
```

- What's the probability that `player2` draws the hand [`ace`, `king`, `queen`, `ace`].
- Given that `player2` draws the hand [`ace`, `king`, `queen`, `ace`], and that the coin comes up tails, what is the posterior belief that your opponent is cheating?
- What is the prior probability that player 1 wins?[1] Why does this query take so long to answer? What is the probability that player 1 wins, given that you know that player 2 is a cheater?

[1] This query might time-out on the online editor.