Thomas Huzij
tph2109@columbia.edu

Homework 4

14.14) Consider the Bayes net shown in Figure 14.23.

a. Which of the following are asserted by the network structure?

i) P(B, I, M) = P(B)P(I)P(M)

This is not asserted by the network structure. It is clear that P(I) depends on P(B) and P(M) and thus the equation is too simplistic and assumes independence.

ii) P(J | G) = P(J | G, I)

This is asserted by the network structure. Looking at P(G), it is obvious that P(G) is always equal to 0 if I is false. Therefore G cannot exist without I and thus the statement is asserted.

iii) P(M | G, B, I) = P(M | G, B, I, J)

This is asserted by the network structure. Any node can at most be dependent on its parents, its children, and the parents of its children. This is known as the Markov blanket. Thus adding J, which is a child of M’s children and therefore independent of M, creates an equivalent statement.

b. Calculate the value of P(b, i, ¬m, g, j).

P(b, i, ¬m, g, j) = P(b | i, ¬m, g, j)P(i | ¬m, g, j)P(¬m | g, j)P(g | j)P(j) = P(b)P(i | b, ¬m)P(¬m)P(g | b, i, ¬m)P(j | g) = .9 * .5 * .9 * .8 * .9 = 0.2916

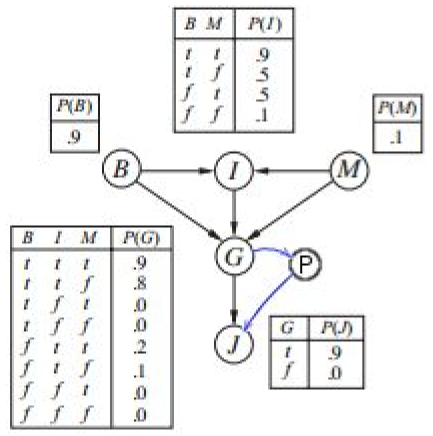
c. Calculate the probability that someone goes to jail given that they broke the law, have been indicted, and face a politically motivated prosecutor.

P(j | b, i, m) = P(j | g)P(g | b, i, m) = .9 * .9 = 0.81

d. A context specific independence allows a variable to be independent of some of its parents given certain values of others. In addition to the usual conditional independences given by the graph structure, what context-specific independences exist in the Bayes net in Figure 14.23?

It is clear from the network that when I is false, the values of B and M do not affect the probability of G. Therefore G is context-specific independent of B and M given I is false.

e. Suppose we want to add the variable P = PresidentialPardon to the network; draw the new network and briefly explain any links you add.



P(B)
.9

B	M	P(I)
t	t	.9
t	f	.5
f	t	.5
f	f	.1

P(M)
.1

B	I	M	P(G)
t	t	t	.9
t	t	f	.8
t	f	t	.0
t	f	f	.0
f	t	t	.2
f	t	f	.1
f	f	t	.0
f	f	f	.0

G	P(J)
t	.9
f	.0

For the sake of brevity, let us assume that the entire network in Figure 14.23 is redrawn but with a new node P being pointed to by node G (i.e., (G) -> (P)), because it only makes sense to pardon someone who is found guilty, and P pointing to J (i.e. (P) -> (J)), because whether or not someone has been pardoned will affect whether or not they go to jail. P(P | G) could be equal to 0.01 and P(P | ¬G) would obviously be 0. The probabilities for J would also need to be updated so that J is context-specific independent of B, I, and M if P is true (i.e., once a pardon has been given, the probability of going to jail would be 0, regardless of the values of B, I, and M).

17.10)

a. What can be determined qualitatively about the optimal policy in states 1 and 2?

To achieve the highest reward, terminal state 3 must be reached as soon as possible, because each time states 1 or 2 are entered a negative reward is added to the total. So the simplest optimal policy would be to always attempt to transition to state 3. However, if the current state is 2, it would make more sense to move to state 1 first before attempting to reach state 3. This is because the probability of entering state 3 is very low and each failed attempt from any given state will once again add that state's negative reward to the total. Thus, the optimal policy is to enter state 1 and repeatedly try to reach state 3 until this goal has been achieved.

b. Apply policy iteration, showing each step in full, to determine the optimal policy and the values of states 1 and 2. Assume that the initial policy has action b in both states.

U(state1) = -1 + 0.1 * U(state1) + 0.9 * U(state3), U(state1) = -10
U(state2) = -2 + 0.1 * U(state2) + 0.9 * U(state3), U(state2) = -20
U(state3) = 0

For each possible action from any given state, sum the utilities.

For state 1,
sum of U(destination-state$_i$) given action a = 0.8 * -20 + 0.2 * -10 = -18
sum of U(destination-state$_i$) given action b = 0.9 * -10 + 0.1 * 0 = -9
The sum of U(destination-state$_i$) for action b is greater, so action b remains in the policy for state 1.

For state 2,
sum of U(destination-state$_i$) given action a = 0.8 * -10 + 0.2 * -20 = -12
sum of U(destination-state$_i$) given action b = 0.9 * -20 + 0.1 * 0 = -18
The sum of U(destination-state$_i$) for action a is greater, so action a replaces the old value for state 2 in the policy.

Given that an action was updated, must iterate again.

U(state1) = -1 + 0.1 * U(state1) + 0.9 * U(state3), U(state1) = -10
U(state2) = -2 + 0.8 * U(state1) + 0.2 * U(state2), U(state2) = -12.5
U(state3) = 0

For state 1,
sum of U(destination-state$_i$) given action a = 0.8 * -15 + 0.2 * -10 = -12
sum of U(destination-state$_i$) given action b = 0.9 * -10 + 0.1 * 0 = -9
The sum of U(destination-state$_i$) for action b is greater, so action b remains in the policy for state 1.

For state 2,
sum of U(destination-state$_i$) given action a = 0.8 * -10 + 0.2 * 12.5 = -10.5
sum of U(destination-state$_i$) given action b = 0.9 * -12.5 + 0.1 * 0 = -11.25
The sum of U(destination-state$_i$) for action a is still greater, so action a remains in the policy for state 1. Since the policy remained unchanged, the iteration terminates and we have the final policy of <state1, b>, <state2, a>.

c. What happens to policy iteration if the initial policy has action a in both states? Does discounting help? Does the optimal policy depend on the discount factor?

If the initial policy has action a in both states, then the problem becomes unsolvable. The utility functions would be as follows:
U(state1) = -1 + 0.8 * U(state2) + 0.2 * U(state1)
U(state2) = -2 + 0.8 * U(state1) + 0.2 * U(state2)
U(state3) = 0

Since U(state1) depends on U(state2) and vice versa, it is clear that the equations are inconsistent and the policy iteration will never terminate (the values will slowly march towards negative infinity). Yes, discounting does help because it will limit the number of iterations and set a limit on the decreasing reward to produce a final terminating result. Instead of iterating indefinitely, the final utilities will be calculated because each new iteration will have a smaller, decrease in reward. Finally, the discount factor does have an effect on the optimal policy. Depending on how quickly the utilities are being discounted, certain actions which may not make any sense on their own can actually be optimizing the final reward when compared against the utility of those actions after many iterations. Previously we wanted to leave state 2 before attempting to reach the terminal state because we wanted to minimize the

18.11) Suppose you are running a learning experiment on a new algorithm for Boolean classification. You have a data set consisting of 100 positive and 100 negative examples. You plan to use leave-one-out cross-validation and compare your algorithm to a baseline function, a simple majority classifier. (A majority classifier is given a set of training data and then always outputs the class that is in the majority in the training set, regardless of the input.) You expect the majority classifier to score about 50% on leave-one-out cross-validation, but to your surprise, it scores zero every time. Can you explain why?

There are only two classes to choose from for the boolean classifier and there are originally an equivalent amount of each class. If you leave-one-out an example of one of the classes, then the opposite class immediately becomes the majority class and thus the chosen example's class will never be the majority. Therefore it will always be wrong and the score will be zero every time.