Assignment 4: EM and Decision Making Under Uncertainty

CS486/686 – Winter 2019

Out: March 18th, 2019
Due: April 5th, 2019 5pm

Submit your assignment via LEARN (CS486 site) in the Assignment 4 Dropbox folder.
No late assignments will be accepted

1. **[50 pts]** Expectation Maximization

   Olertawo is a university town in New Zealand in which there is a strange medical condition called *Dunetts Syndrome*. *Dunetts Syndrome* comes in two forms: mild and severe (as well as not being there at all). It is known that about half the population of Olertawo have *Dunetts*, and that about half of those with *Dunetts* have it in severe form. *Dunetts Syndrome* has three observable symptoms, *Sloepnea*, *Forienndit*is, and *Degar spots*, all of which may be present if the patient has *Dunetts*, but with varying frequencies. In particular, it is well known that *Forienndit*is is present much more often when the condition is in its mild form (it is not as common in severe cases), whereas *Degar spots* are present much more often when the condition is in its severe form (and not as common in the mild cases). *Sloepnea* is present in either form of *Dunetts Syndrome*. However, about 10% of the population have a gene (called *TRIMONO-HT/S*) that makes it so they hardly ever show symptom *Sloepnea* (whether they have *Dunetts Syndrome* or not), but does not affect the other two symptoms. Symptoms *Sloepnea*, *Forienndit*is, and *Degar spots* are sometimes present (but much less often) even if the person does not have *Dunetts Syndrome*.

   (a) construct a Bayesian network (BN) for the domain explained above. Assign priors to each CPT based on the prior information given above. You don't need to be precise - just make rough guesses as to what the CPTs may be based on the description above.

   (b) you are given a dataset from 2000 patients, giving the existence of the three symptoms *Sloepnea*, *Forienndit*is and *Degar spots*, and whether they have gene *TRIMONO-HT/S* or not. About 5% of the data also has a record of wether the patient actually had *Dunetts Syndrome* or not. Using the Expectation Maximization (EM) algorithm, learn the CPTs for your BN. Run EM until the likelihood of the complete data (the sum of all your "weights" over *Dunetts Syndrome*) only changes by 0.01 or less. Start EM from your prior model, but add a small amount δ of random white noise to each parameter. Do this by adding a different randomly generated number $r \in [0, \delta]$ to each probability in each CPT, and then renormalising. So if you have a distribution $[p, 1-p]$ you draw two random numbers δ_1 and δ_2 between $[0, \delta]$ for whatever δ you are using, and then you compute $\left[\frac{p+\delta_1}{1+\delta_1+\delta_2}, \frac{(1-p)+\delta_2}{1+\delta_1+\delta_2}\right]$ Repeat the EM learning for a range of settings of δ from ~ 0 to 4., and do 20 trials for each setting with different randomizations at the start of each trial (but not at the start of each EM iteration). Use at least 20 values of δ evenly spread in $[0, 4)$. The idea is to evaluate the sensitivity of EM to the initial guess. As δ gets bigger, your initial CPTs will become more and more random, and you should get increasing numbers of trials with low accuracy. If your initial guess is not very good, you may even find that adding a small amount of noise helps.

   (c) To validate your learned model, you are given 100 test instances in which it is known whether the patient had *Dunetts Syndrome*. Make a prediction of whether *Dunetts Syndrome* is present for each example in the test set based on your learned model using EM and compare with the actual values of *Dunetts Syndrome*.

   The datasets are available on the course webpage. `trainData.txt` is a file of 2000 training examples with five columns. The first three give the presence/absence of each symptom (in order *Sloepnea*,*Forienndit*is,*Degar*

spots,*TRIMONO-HT/S* with 0 indicating the symptom is not present, and 1 indicating it is). The fourth column gives the presence/abscence of gene *TRIMONO-HT/S* (with 0 indicating the gene is not present, and 1 indicating it is). The fifth column is -1 if there is no record of *Dunetts Syndrome*, and $0, 1, 2$ if *Dunetts Syndrome* is recorded as being not present, mild or severe, respectively. `testData.txt` is the 100 examples for testing, has an additional column giving the severity of *Dunetts Syndrome*(the last column: 0=none, 1=mild, 2=severe).

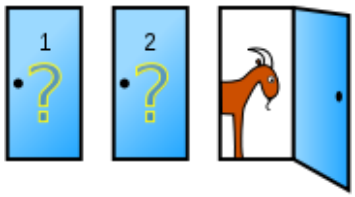
What to hand in:

- A drawing of your Bayesian network showing all CPTs
- A printout of your code for doing EM on this model. You don't have to write code to do EM in general, just for this specific example.
- A graph showing the prediction accuracy on the test set, for each value of δ, giving mean accuracy and standard deviation (error bars) over the 20 trials. Plot both the accuracy before and after running EM.

2. **[50 pts]** Decision Networks:

The *Monty Hall Problem* is stated as follows[1]

Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?" Is it to your advantage to switch your choice?



The host in this problem always opens a door with a goat behind it, but not the door you chose first, regardless of which door you chose first. You "reserve" the right to open that door, but can change to the remaining door after the host opens the door to reveal a goat. You get as a prize whatever is behind the final door you choose. You prefer cars over goats (cars are worth 1 and goats are worth 0). The car is behind doors 1 and 2 with probability p_1 and p_2, respectively (so is behind door 3 with probability $1 - p_1 - p_2$), and you know what these numbers are.

(a) Design a decision network to represent this problem. Your network should be as small as possible. For instance, the host's choice has only two possibilities, not three, and your second choice also has only two possibilities (stay or switch). Draw all decision nodes, random nodes, and utility nodes, as well as all information arcs. Clearly state your variables and their interpretation, and show all probability and utility tables explicitly.

(b) Compute the policy of action for the first and second choices by using the variable elimination algorithm for decision networks when $p_1 = p_2 = \frac{1}{3}$. Show all your work including all the intermediate factors created. Clearly indicate the optimal policy (decision function for the first and second choices), and give the expected value of your decisions (this is a single number).

(c) Now do the calculation again, but use $p_1 = 0.8$ and $p_2 = 0.1$. Show the final decision functions and give the expected value.

(d) Now, suppose you don't know what p_1 and p_2 are, but you can play the game repeatedly. You are given a function $simulator(s, a)$ which takes as inputs a list of two integers (pd, hc) representing the state (see below), each of which takes on values $\in \{0, 1, 2, 3\}$ and an action $a \in \{1, 2, 3\}$. The state variables have the following interpretation (doors are numbered 1,2,3):

- pd: the door you last chose (0 at the start, refers to the first door you pick after the first choice, and to the second door you pick after the second choice. pd may be the same after the first and second steps. Resets to 0 after every second action.)
- hc: the door the host chose (0 if no door chosen yet, and the chosen door by the host after your first choice of door each time). Will never be the same as your first choice, and will never be the door with the car. Resets to 0 after every second action as well.

[1] vos Savant, Marilyn (9 September 1990a). "Ask Marilyn". Parade Magazine: 16. `http://marilynvossavant.com/game-show-problem/`". Drawing by Cepheus - Own work, Public Domain, `https://commons.wikimedia.org/w/index.php?curid=1234194`.

Note that the state variables in this case are not the same as the ones you will use in Question 2(a). This is because in that case you used decision nodes explicitly representing your choices, whereas here, you need to "remember" these choices as part of the state (in variable pd). The function `simulator` returns a new state (pd', hc') and a reward r. The reward is non-zero only after every second call to this function (when the hc passed in is non-zero). The actual location of the car resets every second call to this function as well. Athough this is a global variable ($actual\_cd$) in the python code provided, you may not use this variable in your code. That is, you can write other functions but cannot access the variable $actual\_cd$ and can only use the provided code by calling the function $simulator$. Your Q-learning code must look like this:

```
...
s=(0,0)
while not done:
    ....
    a = get_best_action(...,s,...)  // your function
    (sp,r)=simulator(s,a)
    s=sp
    ....
```

Also provided is a short snippet of test code as the main function in `simulator.py` that you can look at which shows how the game is played with the simulator function. You can play the game yourself and see how it works, and then use that as a starting point for your Q-learning code. Here's an example of a simulated run:

	state		your	new state		reward	
$actual\_cd$	pd	hc	action	pd'	hc'	r	explanation
1	0	0	2	2	3	0	car is behind door 1, you chose door 2, host chose door 3
1	2	3	1	0	0	1	you switch to door 1, you win a car!, actual car door resets to 2
2	0	0	2	2	1	0	car is behind door 2, you chose door 2, host chose door 1
2	2	1	3	0	0	0	you switch to door 3, you win a goat!, actual car door resets to 2
2	.....						

Run Q-learning for until convergence with $p_1 = 0.8$ and $p_2 = 0.1$ (as it is in `simulator.py`). Use a fixed learning rate of 0.1, a discount factor 0.75, and select actions during learning using a greedy approach with random actions selected 10% of the time (and the best action according to the Q-function learned so far the rest of the time). Start your Q-function off with 0.0 as every value. Measure convergence as the average error per step over each batch of 1000 steps, and stop when this average error falls below 0.1. Measure the error in each step (s, a, s', r) as $|(r + \gamma max_{a'} Q(s', a')) - Q(s, a)|$.

(e) Test your final policy out by running it in simulation for 1000 games and computing the average return per game. Always choose actions according to your final learned Q-function.

(f) In the *confused host* version of the game, the host forgets which door the car is behind, and so takes a chance and opens a door randomly (and you know that his choice is random) but still reveals a goat (by chance)[2]. Will your decisions change? Why? Be brief.

What to hand in for Question 2

- a drawing of your decision network from part (a), showing all tables and variable interpretations
- the work done for your variable elimination computation in part (b), and the final policy and values
- the work done for your variable elimination computation in part (c), and the final policy and values
- your code for doing Q-learning in part (d)
- A printout of your final Q-function (part (d)) as a table showing the Q(s,a) for each s and a as :

[2]this is also known as the "earthquake" version where just before opening a door, an earthquake happens and a door swings open because its hinges break, and it just so happens to reveal a goat.

		action		
pd	hc	1	2	3
0	0	Q((0,0),1)	Q((0,0),2)	Q((0,0),3)
0	1	Q((0,1),1)	Q((0,1),2)	Q((0,1),3)
			...	
3	3	Q((3,3),1)	Q((3,3),2)	Q((3,3),3)

There are 16 total rows in this table, but you only need to report the rows with non-zero values. Highlight the Q value for the best action in each row so the greedy policy can be easily read off.

- The total reward gathered over 1000 games with your final Q-function in part (d)
- A brief answer to the *confused host* version question. You don't need to do calculations, just a few sentences explaning the optimal policy and giving a reason why.