Assignment #4

Artificial Intelligence - CSCE 523

Due: 8:00 AM, Monday March 11, 2019

Uncertainty and Machine Learning

Marvin Newlin

1. Bayes Rule: Seventy percent of the aircraft that disappear in flight through the Bermuda Triangle are recovered (P(r) = 0.7). Sixty percent of the recovered aircraft have an emergency locator (P(e|r) = 0.60). Unfortunately, 90% of the aircraft not recovered do not have such a locator. Suppose that an aircraft with a locator has disappeared. What is the probability that it will be recovered (P(r|e))?

**Sol’n:**

P(r) = 0.7, P(e|r) = 0.6, P(~e | ~r) = 0.9

P(~r) = 0.3, P(~e | r) = 1 – P(e | r) = 0.4, P(e | ~r) = 1 – P(~e | ~r) = 1 – 0.9 = 0.1

P(e,r) = P(e|r)\*P(r) (Product rule)

🡪 P(e,r) = 0.6 \* 0.7 = 0.42

P(~e,r) = P(~e|r)\*P(r) (Product Rule)

🡪 P(~e,r) = 0.3\*0.7 = 0.21

P(e,~r) = P(e|~r)\*P(~r) (Product Rule)

🡪 P(e,~r) = 0.1\*0.3 = 0.03

P(~e,~r) = P(~e|~r)\*P(~r) (Product Rule)

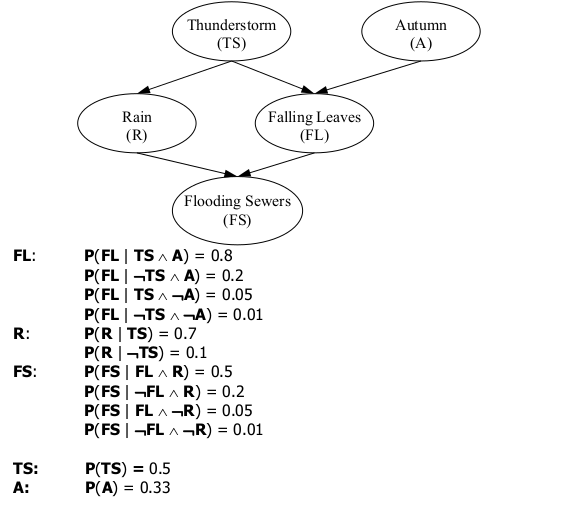
🡪 P(~e,~r) = 0.9\*0.3 = 0.27

P(e) = P(e,r) + P(e,~r) = 0.42 + 0.03 = 0.45

P(r|e) = (Bayes rule)

P(r|e) = 🡪 P(r|e) = 0.42/0.45 = 0.9333

2. Shown below is a Bayes network representing the risk of flooding sewers (**FS**) in a city as dependent on rainfall (**R**), falling leaves (**FL**), thunderstorm (**TS**), and autumn (**A**). Use the conditional probabilities below to determine the conditional probabilities of a thunderstorm for the observable scenarios  , , , and .



|  |  |
| --- | --- |
| P(R | TS) = 0.7 | P(~R | TS) = 0.3 |
| P(R | ~TS) = 0.1 | P(~R | ~TS) = 0.9 |

|  |  |
| --- | --- |
| P(FS | FL, R) = 0.5 | P(FS | FL, R) = 0.5 |
| P(FS | ~FL, R) = 0.2 | P(~FS | ~FL, R) = 0.8 |
| P(FS | FL, ~R) = 0.05 | P(~FS | FL, ~R) = 0.95 |
| P(FS | ~FL, ~R) = 0.01 | P(~FS | ~FL, ~R) = 0.99 |
| P(FL | TS, A) = 0.8 | P(~FL | TS, A) = 0.2 |
| P(FL | ~TS, A) = 0.2 | P(~FL | ~TS, A) = 0.8 |
| P(FL | TS, ~A) = 0.05 | P(~FL | TS, ~A) = 0.95 |
| P(FL | ~TS, ~A) = 0.01 | P(~FL | ~TS, ~A) = 0.99 |

**Sol’n:**

Scenario 1: FS, A

= 0.5\*0.33\*[0.1\*0.2\*0.5 + 0.1\*0.8\*0.2 + 0.9\*0.2\*0.05 + 0.9\*0.8\*0.01] = 0.5\*0.33\*0.0422 = 0.006963

0.052899 + 0.00693 = 0.059862

0.052899/0.059862 = 0.884 = P(ts | fs, a)

0.006963/0.059862 = 0.116 = P (~ts | fs, a)

P(TS | fs, a) = < 0.884, 0.116 >

Scenario 2: FS, ~A

= 0.5\*0.67\*[(0.7\*0.05\*0.5) + (0.7\*0.95\*0.2) + (0.3\*0.05\*0.05) + (0.3\*0.95\*0.01)]

= 0.5\*0.67\*0.1541 = 0.0516235

= 0.5\*0.67\*[(0.1\*0.01\*0.5) + (0.1\*0.99\*0.2) + (0.9\*0.01\*0.01) + (0.9\*0.99\*0.01)]

=0.5\*0.67\*0.0293 = 0.0098155

Scenario 3: ~FS, A

= 0.5\*0.33\*[(0.7\*0.8\*0.5) + (0.7\*0.2\*0.8) + (0.3\*0.8\*0.95) + (0.3\*0.2\*0.99)]

= 0.5\*0.33\*[0.6794] = 0.112101

= 0.5\*.33\*[(0.1\*0.2\*0.5) + (0.1\*0.8\*0.8) + (0.9\*0.2\*0.95) + (0.9\*0.8\*0.99)]

= 0.5\*0.33\*0.9578 = 0.158037

0.158037 = 0.270138

0.112101/0.270318 = 0.415 =

0.158037/0.270318 = 0.585 =

Scenario 4: ~FS, ~A

= 0.5\*0.67\*[(0.7\*0.05\*0.5) + (0.7\*0.95\*0.8) + (0.3\*0.05\*0.05) + (0.3\*0.95\*0.99)]

= 0.278854

= 0.5\*0.67\*[(0.1\*0.01\*0.5) + (0.1\*0.99\*0.8) + (0.9\*0.01\*0.95) + (0.9\*0.99\*0.99)]

= 0.5\*0.67\*0.97106 = 0.3253051

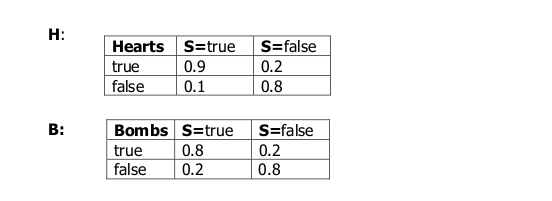
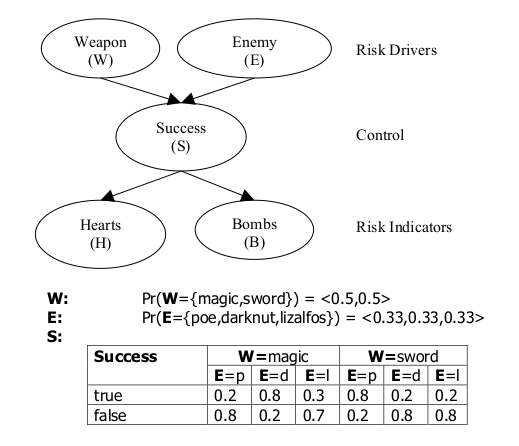
= P(ts | ~fs, ~a)

0.538 = P(~ts | ~fs, ~a)

3. Link sees Sheik on the horizon. Sheik is fighting with magic and has all her hearts. Link wants to determine Sheik’s potential for defeating the enemy and whether he should enter the fray.

Shown below is a risk analysis Bayesian network that Link plans to use. His risk drivers are the type of weapon in use and the enemy faced. Using certain weapons on specific enemies can improve effectiveness in battle. The effectiveness affects the number of hearts the individual has, as well

as the number of bombs they may have left. His question is what is the probability of success given magic and hearts: Pr(**S**|**W**=magic,**H**=true)?



**Sol’n:**

= \*0.5\*0.9\*[(0.33\*0.2) + (0.33\*0.8) + (0.33\*0.3) + (0.33\*0.2) + (0.33\*0.8) + (0.33\*0.3)]

=  \* 0.5\*0.9\*[0.858] = \* 0.3861

= \*0.5\*0.2\*[(0.33\*0.8) + (0.33\*0.2) + (0.33\*0.7) + (0.33\*0.8) + (0.33\*0.2) + (0.33\*0.7)]

= \* 0.5\*0.2\*1.122 = \* 0.1122

0.3861 + 0.1122 = 0.4983

0.3861/0.4983= 0.775

0.1122/0.4983= 0.225

P(S | W=magic, h=true) = < 0.775, 0.225 >

4. For this problem, you need to build a Bayes network in problem 2 in JavaBayes. Using the Bayes network, double check your solutions for problem 2. And change the table for Rain to:

|  |  |
| --- | --- |
| **R:** | **P(R | TS) = 0.9** |
|  | **P(R|~TS) = 0.3** |

What are the conditional probabilities of thunderstorm (**TS**) given the observable scenarios **FL**, and **~FL**. Turn-in your Bayes network file with your assignment.

**Sol’n:**

Problem 2 Scenarios

|  |  |  |
| --- | --- | --- |
|  | T | F |
| P(TS | FS, A) | 0.884 | 0.116 |
| P(TS | FS, ~A) | 0.839 | 0.161 |
| P(TS | ~FS, A) | 0.415 | 0.585 |
| P(TS | ~FS, ~A) | 0.466 | 0.534 |

JavaBayes filename: problem2.xml

Problem 4 Scenario

|  |  |  |
| --- | --- | --- |
|  | T | F |
| P(TS | FL) | 0.804 | 0.196 |
| P(TS | ~FL) | 0.431 | 0.569 |

JavaBayes filename: problem4.xml

5. Ahhh, life at AFIT- it's not just a job, it's an AI problem! You wake up to the sunrise after studying for a final all night. You find yourself amidst hundreds of cargo containers; your watch reads seven forty-five. Uh oh, you only have fifteen minutes to get to your exam. Unfortunately, you still must negotiate a maze of cargo between here and the classroom. Which brings us to the problem: how many steps does it take to reach your destination?

There are two challenges for you:

**Part I:**

This challenge requires you to calculate V(s) for a given map and output the MEU path. The calculation of the MEU should be conducted by performing value or policy iteration (your choice).

This is a gridworld in which a cell is either occupied or empty, and the agent may move, North, South, East, or West, 1 square at a time. The cost of moving is 1.0. When the agent moves, there is a probability of 0.90 that they will end in the state that they want to be in and a probability of 0.07 that they remain in the current state, and a probability of 0.03 that they go backwards a square (i.e. if they were headed West, they would instead go 1 square East). The world is fully-observable, the

agent knows where its location and the locations of the goal and the obstacles.

**Part II:**

For this challenge, solve the same problem using reinforcement learning. Use TD(0)/SARSA, and TD(λ). Solve the problem for V(s) not Q(s,a). The values should converge to those close to Part I.

Turn-in should include your code (no language stipulation), and a write-up which draws comparisons between the solutions. Include a results graph indicating the path length vs iteration (plot based on the same start location, and curves should go down). Testing should use world sizes of 25x25 and 50x50.

DataFile Format Example:

3 3

O O O

G V V

V V V

World x size, World y size

Each map location either O – obstacle, V – vacant, or G – goal

Have the agent start location be manually inserted or randomly generated at the users request.

Take a look at using the BURLAP (http://burlap.cs.brown.edu/) library as a starting point. You will need to install maven and hook it into your IDE.

Turn-ins: a write-up of your solution, with a graph comparing the convergence between approaches. Be sure to discuss your parameter setting, and how you identified the values you used (a parameter sweep is not unwarranted).

**Sol’n:**

**Part 1**

Code for part 1 is located in the Newlin\_VI.java file. To run in IntelliJ, click Newlin\_VI.java and click the “Run” button.

For part 1, I used a reward of 10-move cost of 1 for the goal node and a cost of 1 for all other cells. The initial V(s) array is printed before running the value iteration and again afterward. The provided output is from running the 10x10 maze2.txt file in the src folder. Note that the locations in the final V(s) array with zero values are the locations of the walls.

[0.0, 0.0, 0.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0]

[9.0, -1.0, 0.0, 0.0, 0.0, -1.0, -1.0, -1.0, -1.0, -1.0]

[-1.0, -1.0, -1.0, 0.0, -1.0, -1.0, -1.0, 0.0, -1.0, -1.0]

[-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, 0.0, -1.0, -1.0]

[-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, 0.0, -1.0, -1.0]

[-1.0, -1.0, -1.0, 0.0, 0.0, 0.0, -1.0, -1.0, -1.0, -1.0]

[-1.0, -1.0, -1.0, -1.0, -1.0, 0.0, -1.0, -1.0, -1.0, -1.0]

[-1.0, -1.0, -1.0, -1.0, -1.0, 0.0, -1.0, -1.0, -1.0, -1.0]

[-1.0, 0.0, 0.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0]

[-1.0, 0.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0]

Value Iteration Complete

Total Iterations: 246

[[0.00, 0.00, 0.00, 227.15, 235.97, 237.50, 236.25, 234.99, 233.49, 224.76]

[258.60, 249.08, 0.00, 0.00, 0.00, 246.66, 245.37, 244.07, 242.52, 233.49]

[257.33, 255.79, 246.35, 0.00, 241.67, 248.21, 244.08, 0.00, 241.23, 232.24]

[256.06, 254.79, 253.52, 252.24, 250.96, 249.50, 242.79, 0.00, 239.93, 230.98]

[254.79, 253.52, 252.24, 250.96, 249.68, 248.21, 241.49, 0.00, 238.63, 229.72]

[253.52, 252.24, 250.96, 0.00, 0.00, 0.00, 240.20, 238.89, 237.34, 228.47]

[252.24, 250.96, 249.68, 248.14, 238.94, 0.00, 239.07, 237.76, 236.21, 227.37]

[250.96, 249.67, 248.39, 247.01, 242.79, 0.00, 240.29, 238.98, 237.43, 228.55]

[249.41, 0.00, 0.00, 245.46, 244.17, 242.88, 241.58, 240.28, 238.72, 229.81]

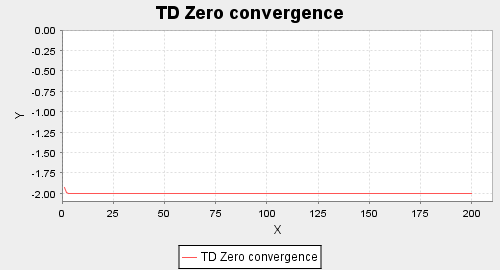
[240.17, 0.00, 227.52, 236.34, 235.09, 233.84, 232.58, 231.32, 229.81, 221.19]

As we can see, the goal node (element 1,0) has the highest value and all other values decrease from there. To get the MEU path, we simply follow the max value policy from wherever our start node is. To get to a stopping point on the Value Iteration, we use an epsilon to measure the sum difference of each respective element in the v(s) and v(s’) arrays. In our case, we used an epsilon of 5 which performed reasonably well.

**Part 2**

Code for part 2 is located in the Newlin\_TD.java file. To run in IntelliJ, Click the Newlin\_TD.java file and click the “Run” button.

I used a reward of 100-move cost, i.e. 99 for the goal node and a move\_cost of -1 for all other nodes except walls. These parameters seemed to work the best of the ones that I tried. I had issues getting TD0 to work correctly and all the values stayed nearly constant at around -2 as shown below. I ran the evaluation for 200 episodes as shown below. For both TD instances, I used a 25x25 maze with the filename maze3.txt located in the src folder.



When using TD lambda, the values continued to grow and did not converge. I used the same setting is TD0 and TD Lambda but still didn’t get any convergence.

