Assignment 2

Intro to AI CS1571

Due Monday, Nov. 2 at 11:59PM

Total: 100 points (+20 bonus points)

Goals

The goals of this assignment are to practice some of the techniques we have discussed in class with respect to the use of decision trees, Markov Decision Processes, and logical inferences. Most of this assignment will involve problem-solving and pseudocode, with a small portion involving the use to the textbook code to adapt it to a new application.

Logistics

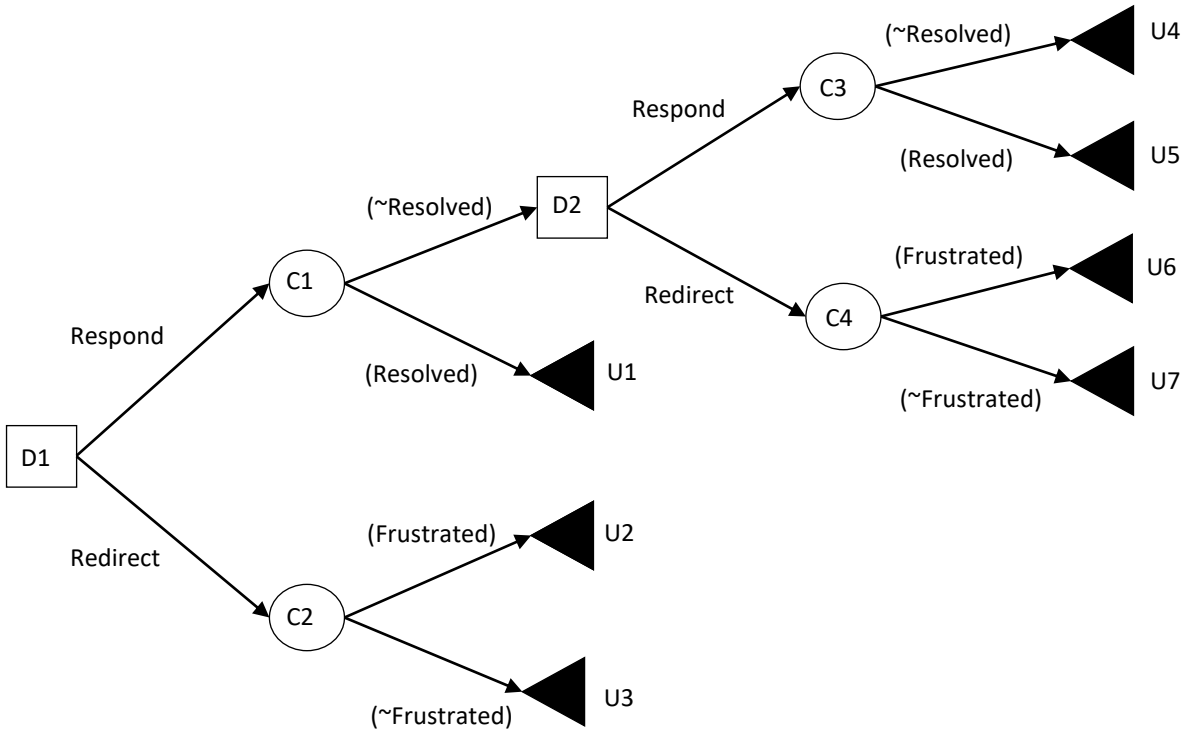
- This is an individual assignment. You may talk about it in general terms with your classmates, but your work should represent your individual work.
- Submit your written work as a single PDF on Gradescope. Indicate how your work maps to the different assignment questions during your submission.
- Name your code as follows: <lastname>_a2.py . You can submit all your code as part of this file.

Installation

- For this assignment, you will use logic4e.py from the AIMA code.
- Download the a2_release.py file provided as part of this assignment. You can use it to create your submission file.

Part A – Decision Trees (30 points)

1. Say you are building a customer service spoken dialogue chatbot. Each time the chatbot responds to the user is denoted by a *conversational turn.* In each turn, the chatbot has two simple choices: respond to the user's query (**Respond**), or direct the user to a human customer service representative (**Redirect**). Consider the following decision tree:



You'll notice there are two different types of chance nodes. Assume the number of turns taken so far is represented by *k.* At the first decision made by the chatbot, *k*=1.

- *Resolved* represents the probability that a problem will be resolved after a given turn. It increases logarithmically with the turn number. P(Resolved) = min(.10 + .3log*k*, 1)
- *Frustrated* represents the probability that the user was frustrated with the dialogue system on the previous turn. It increases exponentially with the number of terms. P(Frustrated) = min(.1 + .005*k^2, 1)

Assume a utility function *U.* The utility value for redirecting a user to a human if they are NOT frustrated is -25. The utility value for redirecting a user to a human if they are frustrated is 10. The utility value for resolving the user's problem is max(100 – 5*k*, 5). If your tree has a set number of turns (above the maximum number of turns is 2), assume at the terminal level that the utility value of continuing to respond without resolving the problem or redirecting the user is 0. Note, if the problem is resolved or the user is redirected to a human, no further turns are taken by the chatbot.

1. (5 pts) Given the decision tree above, fill out the expected value of each chance node in the below table, the utility value for each of the outcome nodes. Then, describe the best course of action for the chatbot on the first turn.

Node	Value
U1	95
U2	10
U3	-25
U4	0
U5	90
U6	10
U7	-25
C1	.1*95 + .9*17.128 = 24.962
C2	.105*10 + .895*-25 = -21.325
C3	.1903*90 + .8097*0 = 17.128
C4	.12*10 + .88*-25 = -20.8
Best Course of Action	Respond

2. (15 pts) Implement an expectimaxResponseTree function that takes the number of levels in the tree above as an input. It should return a list of actions to take, with each action formatted as follows: "<<level>> - <<action>> - <<utilityValue>>". There is an example in a2_release.py. You can test your function for k=2 to make sure it is functioning correctly (your results should match your answer to 1).

3. (5 pts) Assume the tree is cut off at 20 levels rather than 2. Given the current utility function and probabilities, at which turn is it first better to stop the conversation and redirect the user to a human? You can show your work by calling your function and pasting your output below, or showing the mathematical reasoning you used.

   ['1 - Respond - 78.32475183514752', '2 - Respond - 76.47194648349725', '3 - Respond - 73.29232572083757', '4 - Respond - 69.53132913995634', '5 - Respond - 65.44768867974983', '6 - Respond - 61.162267421907615', '7 - Respond - 56.74117281892929', '8 - Respond - 52.22474411917817', '9 - Respond - 47.640137420317686', '10 - Respond - 43.00792597421503', '11 - Respond - 38.346543290358376', '12 - Respond - 33.676551514906194', '13 - Respond - 29.026470314490716', '14 - Respond - 24.442647100383564', '15 - Respond - 20.007665026551443', '16 - Respond - 15.876124334189', '17 - Respond - 12.34567338206741', '18 - Respond - 10.0', '19 - Redirect - 10']
   It is first better to redirect the user to a human at turn 19.

4. (5 pts) Assume the chatbot always has the option to ask a person if they'd like to be directed to a human prior to deciding to redirect them. You can assume if people say yes there is a higher probability they are frustrated, and if they say no, they are likely not frustrated – but sometimes people will say yes even if they are not frustrated or no if they are frustrated. Name two different changes you would make to the structure of the decision tree, utility function, or probabilities at chance nodes, and justify why your changes make sense based on how these interactions are likely to unfold in the real world.

A third option would be added to the tree. The decisions that could be made would be respond redirect, and ask. If the user is asked, it would increase the probability that they become frustrated. Asking this

would waste time when the user just wants to solve their problem which raises the chance of the user becoming frustrated with the bot.

Part B –Markov Decision Processes and Reinforcement Learning (30 pts)

5. (10 pts) Explain how you would implement the problem of when to redirect a user to a human operator as a Markov Decision Process by outlining **states**, **actions**, **transition probabilities**, and **rewards**. Assume the Markov Property **holds**, that is each state *s'* only depends on the agent's previous state *s*. For this reason, you should use the probabilities and utilities from the above scenario where $k = 1$. Assume P(Resolved) = .1, P(Frustrated) = .105, and U(Resolved) = 95. All other utilities are the same as in Question 1.

   Submit:
   - a diagram of your MDP with states represented by nodes and potential transitions between states as directed edges.
   - a transition probability table, with state-action pairs represented in the rows, and the next state represented by the columns
   - a reward table showing the mapping between states and rewards

   Transition Probability Table

Respond → S1	.9
Respond → S2	.1
Redirect → S3	.105
Redirect → S4	.895

   Rewards Table

Respond → S1	95
Redirect → S3	10
Redirect → S4	-25

   These tables can be formatted in the same way as in the Week 7 worksheet.

6. (10 pts) Walk through two iterations of value iteration on your MDP, using $\gamma=0.8$. As in the Week 7 worksheet, you should submit four different tables, two for $Q_1(s,a)$ and $Q_2(s,a)$, and two for $V_1(s)$ and $V_2(s)$.\

*- terminal state

$Q_1(s,a) = \sum_{s'} Pr(s'|s,a) [R(s')+\gamma V_0(s')]$

s	1	2	3	4

a	**Respond**	Redirect	**Respond**	Redirect	**Respond**	Redirect	**Respond**	Redirect
Q(s,a)	9.5	-21.325	0*	0*	0*	0*	0*	0*

$V_1(s) = \max (Q_1(s, \text{left}), Q_1(s, \text{right}))$

s	1	2	3	4
$V_1(s)$	9.5(Respond)	0*	0*	0*

$Q_2(s,a) = \sum_{s'} Pr(s'|s,a) [R(s')+\gamma V_1(s')]$

s	1		2		3		4	
a	**Respond**	Redirect	**Respond**	Redirect	**Respond**	Redirect	**Respond**	Redirect
$Q_2(s,a)$	17.1	-13.725	0*	0*	0*	0*	0*	0*

7.

8. $V_2(s) = \max (Q_2(s, \text{left}), Q_2(s, \text{right})$

s	1	2	3	4
$V_2(s)$	17.1(Respond)	0*	0*	0*

9. (10 pts) Consider the article assigned on reinforcement learning in the class, which contains python code for Q-learning (https://blog.floydhub.com/an-introduction-to-q-learning-reinforcement-learning/). Answer the following two questions based on the code and the lecture on Q-learning delivered in class.
    a. What line of code selects action that the agent chooses to try during Q-learning training? Write a few lines of your own code that would follows the epsilon-greedy method to select the action, which was described in class on 10-5. Paste your code below.

```
2. current_state = np.random.randint(0,9)
```

epsilon = some value 0 < e < 1

if epsilon > random.random():

current_state = np.random.randint(0,9)

else:

current_state = np.argmax(Q[current_state,])

a. Describe (no need to write code) how you would modify the code in the blog post so that it would work with the MDP you outlined in Question 5.

The value of TD would need to be changed. Instead of just using the reward value of traveling to the new state, it would need to use the probabilities of the possible states. Using the summation of the probability multiplied by the reward function, the value of TD can be calculauted.

Part C – Logic (30 pts)

One final option for approaching this problem is to use first-order logic to make the decision of when to respond to the user and when to redirect the user to a customer service agent.

10. Create a first-order logic knowledge base that represents when to respond to the user, when to redirect the user to a human, when the user's problem will be resolved, and when the user might become frustrated. Write an English language description of your knowledge base below, and then your knowledge base in a logical formalism. Your logic base should consist of **at least** five sentences, and represent all the above possibilities (respond, redirect, problem resolved, and user frustrated; 10 pts).

    If the user sends longer messages, then the user is frustrated. If a user is frustrated, the bot should redirect the user once 10 responses have occurred. If the bot responds and the user is thankful, the problem is resolved. If the bot responds more than 20 times, the user is redirected. If the bot doesn't redirect the user, it responds.

    LongMessages(user) → Frustrated(User)

    Frustrated(user) ^ TenResponses(Bot) → Redirect(Bot)

    Respond(Bot) ^ Thankful(User) → ProblemResolved(User)

    TwentyResponses(Bot) → Redirect(Bot)

    ~Redirect(Bot) → Respond(Bot)

11. Write a fol_resolution function that accepts as parameters a FolKB_a2 object representing a first-order logic knowledge base in conjunctive normal form and a first-order logic sentence alpha, that you can format using the expr method. There is an example in a2_release.py. It returns **True** if the knowledge base entails alpha and **False** if it does not. You can assume there will be no functions as part of the KB or alpha. The method stub is given in a2_release.py, along with a sample sentence in conjuctive normal form. You may want to consider the following functions from the aima code in your implementation: *pl_resolution*, *pl_resolve*, *unify*, and *subst.* If you use logic4e.py, you will need to subclass FOL_KB to remove the requirement that it consist only of

definite clauses, and we have done this for you in a2_release.py. You may also need to call *move_not_inwards* in order to get rid of a double negation ("~~") in your expressions (20 pts).

Part D Reflection (10 pts)

Of the above four approaches (expectimax, value iteration, q-learning, theorem proving with logic), which do you think is most appropriate for this problem and why?

Expectimax is the best approach for this problem. Value iteration and q-learning aren't very appropriate because there is only one state that isn't a terminal state. Also, there is no reward for moving to a nonterminal state which makes these algorithms unnecessarily inefficient. Because so much of the decision making in this tree is probability based, theorem proving with logic is not the best approach. This leaves expectimax as the best approach to the problem.

Part E Bonus (up to 20 points)

Implement the reinforcement learning algorithm in Part B so that it works with your MDP. Submit your code and instructions for testing it.