CSCI-561 - Fall 2023 - Foundations of Artificial Intelligence

Homework 3

Due Time: Monday, November 20th, 2023, 23:59:59 PST

Overview

This homework explores the applications of Temporal Reasoning in Artificial Intelligence. In general, the solution for a temporal reasoning task involves taking a sequence of actions/observations on an Partially Observable Markov Decision Process (POMDP Environment) , applying a temporal-reasoning algorithm that you learned from this class, and returning the most probable sequence of the hidden states that the POMDP most-likely went through when experiencing the given sequence of actions/observations.

More specifically, this assignment provides you with two versions of temporal data: a base version involving the "Little Prince" Environment and an advanced version that revolves around speech recognition and text prediction.

Scenario 1 : Little Prince Model

The setup in this version is very similar to the "Little Prince" environment shown in Figure 1, presented in the lecture notes and in the optional reading textbook (ALFE).

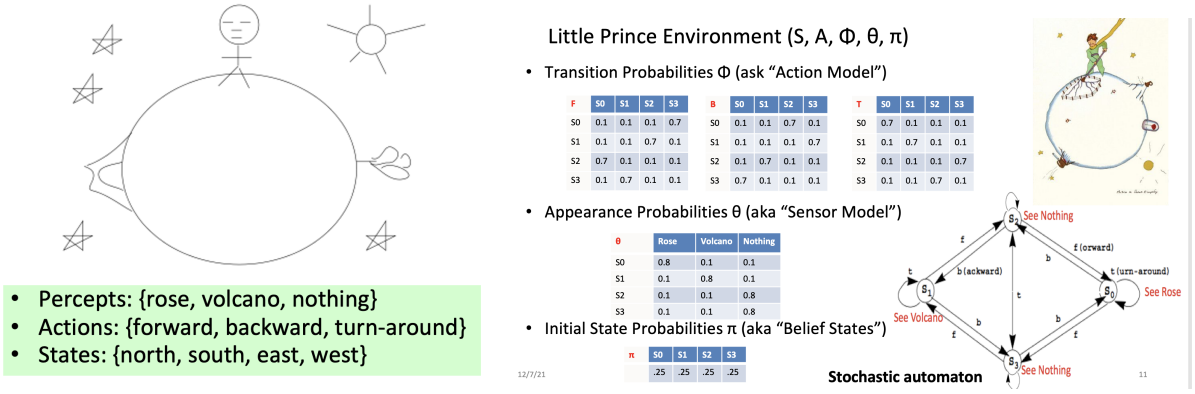

Figure 1: The Little Prince POMDP (lecture 08-09 and 22).

You will be given a list of available percepts, actions and states and the corresponding initial state weightages, transition and observation weight values in that environment (More on the input structure will be covered in the sections below) . Your task is to design and implement a temporal-reasoning algorithm, that will take a sequence of actions/observations and determine the most-likely sequence of states that this POMDP has gone through, as shown in Figure 2. For example, if the Little Prince's experience is given as

<rose, forward, none, …, turn, rose, backward, volcano, …>

Then, your program should return a sequence of hidden-states that this POMDP is most-likely going through (The following sequence is an example of the final state sequence that one might encounter):

<s3, s4, s2, s3, …>

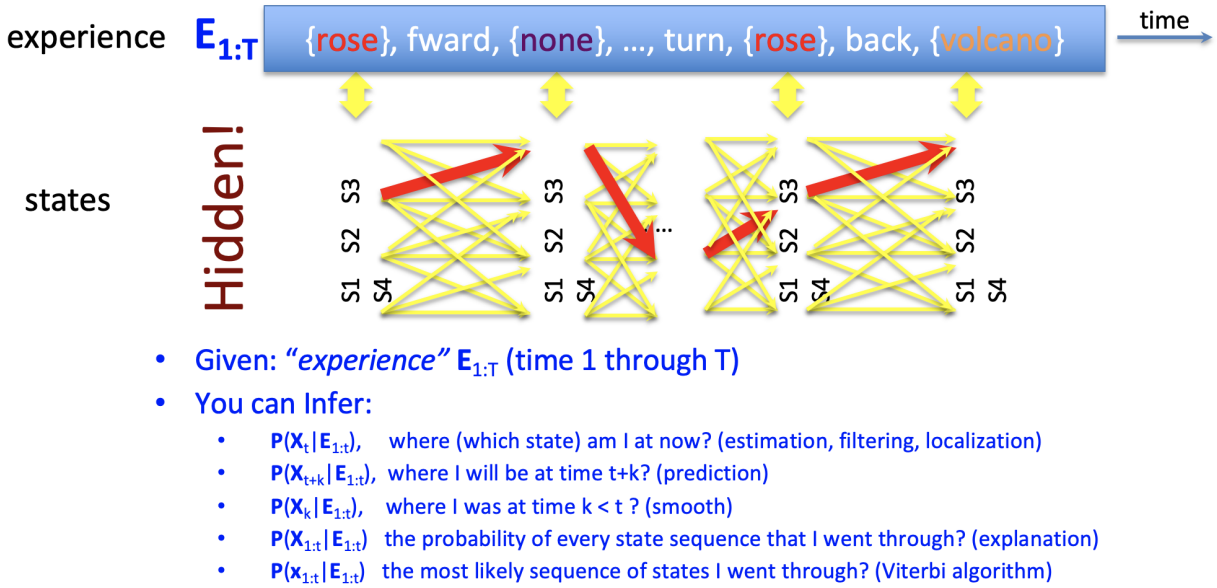

Figure 2: The inputs and outputs of a temporal-reasoning task (lecture 22).

More Details on solving this state sequence prediction problem can be found in the lecture slides. The input file format for this environment will be the same as the Speech Recognition environment. You will be given weight values instead of probabilities and the process of converting weights to probabilities is explained in detail in the following sections.

Scenario 2 : Simplified Speech Recognition

This scenario deals with a more sophisticated environment of speech recognition - without the hassle of going through audio signal processing. This model will primarily focus on resolving the ambiguity introduced by multiple plausible texts that could correspond to a single spoken utterance. For every word pronunciation, instead of dealing with audio signals, you will be given a set of phonemes which phonetically represent the word, and will produce a series of text fragments the same length as the sequence of phonemes. The following table provides some sample words and their phoneme / fragment mapping for better understanding.

Example Words	Phoneme Mapping	Fragment Mapping
water	W | AO1 | T | ER0	w | a | t | er
human	Y | UW1 | M | AH0 | N	h | u | m | a | n
ocean	OW1 | SH | AH0 | N	o | c | ea | n

As evident in the table, there is a 1:1 correspondence with phoneme and fragment for a given word. For the word water, fragment “A” corresponds to the phoneme “AO1” and fragment “er” corresponds to “ER0”.
Because this project builds off prior work from the CMU Pronouncing Dictionary, we use a dialect of English known as North American English, where e.g. human is pronounced with a “Y” sound at the start.

The ultimate goal of this environment is to find the best sequence of text fragments for a given sequence of phonemes. Formally, we will represent the process as a POMDP, with the text fragments corresponding to states and the phonemes corresponding to observations. For convenience, we will use a single null action “N”. This will ensure that we are using a Partially Observable Markov **Decision** Process instead of just a Partially Observable Markov Process. This will also allow you to re-use parsing code between the Little Prince environment and the Speech Recognition environment.

As part of the input, you will be provided with a dataset containing a list of fragment to phoneme pairs, along with a weight value for each pair. You will also be given fragment-to-fragment transition pairs with their weight values. The following example explains the procedure to compute the probability tables in more detail. These weights correspond to un-normalized total probabilities P(o, s) and P(s, s’) (using the null action “N”), which were computed by counting (observation, state) and (state, state) pairs from a set of approximately 300k Wikipedia articles. Note that you will need to normalize these weights into the appropriate probability distributions P(o | s) and P(s’ | s, N).

Consider the following dataset:

Fragment to Phoneme Mapping

Fragment	Phoneme	Weight
s	S	100
s	Z	50
er	ER0	10
o	AH0	10
e	AH0	20

Fragment to Fragment Transition Mapping

Fragment	Fragment	Weight
s	er	80
s	o	10
s	e	10
er	o	5
o	e	8

Construction of Probability Tables :

- **Initial State Probability**: Computing the initial state / prior state distribution only involves dividing each weight in the state probability table by the total weight in the table.
  Given a state table as follows:

State	s	er	o	e
Weight	1	1	1	1

The initial probability P(s) is:

State	s	er	o	e
Prob	0.25	0.25	0.25	0.25

- **State Transition Probability**: This can be determined by looking at the weights of the transitions from one fragment to another and calculating the probability through normalization for each (state, action) pair. This produces P(s|s,a). In this example, there is only a single valid action, so we don't show it in the table.
  The state transition probability for the sample dataset would be:

States	s	er	o	e
s	0.0	0.8	0.1	0.1
er	0	0	1	0
o	0	0	0	1
e	0	0	0	0

- **Appearance Probabilities:** This can be inferred from the fragment-phoneme pairs through normalization over all weights for a given state, producing the conditional distribution P(o|s). The appearance probability for the above example would be as follows:

	S	Z	ER0	AH0
s	0.667	0.333	0	0
er	0	0	1	0
o	0	0	0	1
e	0	0	0	1

  Note how the row for state "s" sums to 1 over the different possible observations / phonemes "S" and "Z".

Your algorithm will be tested on a list of phonemes for which it should provide the most probable sequence of fragments.

Input / Output Format

As mentioned in the problem statement above, there will be two types of input to indicate two modes / stages : Little Prince environment and Simplified Speech Recognition.

In both cases, you will be given a set of input files, containing the table of weights described above. You will need to parse and normalize these tables into the appropriate conditional probabilities.
These input files all contain a similar format, that begins with a one line file type, then a header that describes the number of entries, as well as a default weight. Then, each file contains a sequence of entries, where states, observations, and actions are wrapped in double quotes, and the weights are specified as integers. **At the end of the file, there will be one final newline.**

There are three weight table files:

The first of these files contains weights for every state, and describes the prior probability of each state P(s). (The default weight value is present in this file just to ensure format consistency across all weight files - this will not be required to be used, as all states will be present in the state weights file)

`state_weights.txt:`

```
state_weights
<number of states> <default weight>
"state1" <weight of state1>
"state2" <weight of state2>
etc…
```

The second of these files contains weights for (state, action, state) triples, and describes the probability of state transitions P(s, a, s). Triples not specified in the table should be given the weight default weight specified on the second line. Note that you will need to normalize these weights into the appropriate probability distribution P(s' | a, s).

`state_action_state_weights.txt:`

```
state_action_state_weights
<number of triples in file> <number of unique states> <number of unique
actions> <default weight>
"state1" "action1" "next state1" <weight of (state1, action1, next state1)>
"state2" "action2" "next state2" <weight of (state2, action2, next state2)>
etc…
```

The third of these files contains weights for every (state, observation) pair, and describes the probability of each state observation pair P(s, o). Pairs not specified in the table should be given the default weight specified on the second line. Note that you will need to normalize these weights into the appropriate probability distribution P(o | s).

`state_observation_weights.txt:`

```
state_observation_weights
<number of pairs in file> <number of unique states> <number of unique
observations> <default weight>
"state1" "observation1" <weight of (state1, observation1)>
"state2" "observation2" <weight of (state1, observation1)>
etc…
```

Lastly, you will receive a file containing the sequence of (observation, action) pairs, on which you should run the Viterbi algorithm.

observation_actions.txt:

```
observation_actions
<number of pairs in file>
“observation1” “action1”
“observation2” “action2”
etc…
```

Your code will be expected to produce a file containing the predicted state sequence.

states.txt:

```
states
<length of state sequence>
“state1”
“state2”
etc…
```

Sample Test Case

The following sample test case corresponds to the Little Prince Environment (The format matches that used for Simplified Speech Recognition Environment)

Little Prince Environment Test case:

state_weights.txt

```
state_weights
3 0
"S0" 2
"S1" 5
"S2" 5
```

state_observation_weights.txt

```
state_observation_weights
9 3 3 0
"S0" "Volcano" 3
"S0" "Grass" 3
"S0" "Apple" 2
"S1" "Volcano" 5
"S1" "Grass" 5
"S1" "Apple" 2
"S2" "Volcano" 3
"S2" "Grass" 5
"S2" "Apple" 2
```

state_action_state_weights.txt

```
state_action_state_weights
27 3 3 0
"S0" "Forward" "S0" 3
"S0" "Forward" "S1" 3
"S0" "Forward" "S2" 2
"S1" "Forward" "S0" 4
"S1" "Forward" "S1" 5
"S1" "Forward" "S2" 1
"S2" "Forward" "S0" 1
"S2" "Forward" "S1" 5
"S2" "Forward" "S2" 4
"S0" "Backward" "S0" 5
"S0" "Backward" "S1" 5
"S0" "Backward" "S2" 5
"S1" "Backward" "S0" 3
"S1" "Backward" "S1" 4
"S1" "Backward" "S2" 1
"S2" "Backward" "S0" 2
"S2" "Backward" "S1" 3
"S2" "Backward" "S2" 3
"S0" "Turnaround" "S0" 5
"S0" "Turnaround" "S1" 3
"S0" "Turnaround" "S2" 5
"S1" "Turnaround" "S0" 3
"S1" "Turnaround" "S1" 4
"S1" "Turnaround" "S2" 2
"S2" "Turnaround" "S0" 1
"S2" "Turnaround" "S1" 2
"S2" "Turnaround" "S2" 2
```

observation_actions.txt

```
observation_actions
4
"Apple" "Turnaround"
"Apple" "Backward"
"Apple" "Forward"
"Volcano"
```

Output:

states.txt

```
states
4
"S2"
"S2"
"S2"
"S1"
```

Input Constraints & Time Limits

Little Prince Environment:

Maximum Number of States in the Environment	10 states
Maximum Number of Percepts in the Environment	10 percepts
Number of Actions Possible in the Environment (Fixed)	3 (“Forward”, “Backward”, “Turnaround”)
Maximum Length of Observation Action Sequence	20
Time Limit allowed per test case	1 second
Number of test cases	20 (5 Preliminary and 15 hidden)

Simplified Speech Recognition Environment:

Maximum Number of States in the Environment	682
Maximum Number of Observations in the Environment	69
Number of Actions Possible in the Environment (Fixed)	“N” (Indicates Null Action, Refer Scenario Description)
Maximum Length of Observation Action Sequence	100 steps
Time Limit allowed per test case	1 minute
Number of test cases	30 (5 Preliminary and 25 hidden)

Grading Criteria

There will be 50 test cases in total : 20 cases with the Little Prince environment and 30 cases with Speech Recognition. Each test case is worth 2 points. On clicking the submit button in Vocareum, Your solution will be run on preliminary test cases. Your code will be evaluated on the hidden test cases after the assignment deadline.

The performance of your program will be computed automatically by comparing your output sequence with the most-likely known sequence, and the matching percentage will determine the grade of your submissions.More specifically, the probability of the hidden state sequence that your solution has generated (p1), will be compared with the probability of the hidden state sequence proposed by TA agent (p2) and p1/ p2 will be used as the final score for each test case.

Score for a single test case = 2 * $\frac{\text{Probability of student's state sequence}}{\text{Probability of TA agent's state sequence}}$

Final Score = Sum of scores across all test cases

NOTE : The Max Score for a single test case will be capped at 2 points.

The probability of the state sequence will be calculated based on the joint probability of the state sequence, along with the action sequence and the observation sequence.

In the following example [used for calculation purposes only], the score will be calculated as follows:

States List = [S0, S1, S2], Actions Set= [N], Observations set= [A, B]

Initial State Probability

S0	S1	S2
0.5	0.25	0.25

Appearance Probability

	A	B
S0	0.9	0.1
S1	0.2	0.8
S2	0.5	0.5

Transition Probability

	S0	S1	S2
S0	0.5	0.3	0.2
S1	0.9	0.1	0
S2	0.2	0.8	0

Observation Action Sequence = [A, <N>, B,<N>, B]

If the student's predicted state sequence is [S0, S1, S1], then
Student's Probability = $\pi(S0)$*P(A|S0)*P(S1|S0,<N>)*P(B|S1)*P(S1|S1,<N>)*P(B|S1)
= 0.5*0.9*0.3*0.8*0.1*0.8
= 0.00864

If the TA's predicted state sequence is [S0, S1, S0], then
TA's Probability = $\pi(S0)$*P(A|S0)*P(S1|S0,<N>)*P(B|S1)*P(S0|S1,<N>)*P(B|S0)
= 0.5 * 0.9 * 0.3 *0.8 * 0.9 * 0.1
= 0.00972

Your Score for this test case = 2 * 0.00864 / 0.00972 = 1.778

Notes

- Please name your program "**my_solution.xxx**" where 'xxx' is the extension for the programming language you choose ("py" for python, "cpp" for C++, and "java" for Java). If you are using C++11, then the name of your file should be "my_solution11.cpp" and if you are using python3 then the name of your file should be "my_solution3.py". Please use only the programming languages mentioned above for this homework. Please Note the highest version of Python that is offered is Python 3.7.5, hence the walrus operator and other features of higher version Python are not supported.

- The time limit is the total combined CPU time as measured by the Unix **time** command. This command measures pure computation time used by your program, and discards time taken by the operating system, disk I/O, program loading, etc. Beware that it cumulates time spent in any threads spawned by your agent (so if you run 4 threads and use 400% CPU for 10 seconds, this will count as using 40 seconds of allocated time). Your local machine may be more powerful, and thus faster than Vocareum.

- **Please don't copy code from any website as our plagiarism agent will certainly flag that. This includes Wikipedia.**

- There may be a lot of Q&A on Piazza. **Please always search for relevant questions before posting a new one. Duplicate questions make everyone's lives harder.**

- **Only submit the source code files (in .java, .py or .cpp). All other files should be excluded.**

- Please submit your homework code through Vocareum (https://labs.vocareum.com/) under the assignment HW3. Your username is your email address. Click "forgot password" for the first time login. You should have been enrolled in this course on Vocareum. If not, please post a private question with your email address and USC ID on Piazza so that we can invite you again.

- You can submit your homework code (by clicking the "submit" button on Vocareum) as many times as you want. Only the latest submission will be considered for grading. After the initial deadline, the submission window will still be open for 5 days. However, a late penalty will be applied as 20% per day if your latest submission is later than the initial deadline.

- Every time you click the "submit" button, you can view your submission report to see if your code works. The grading report will be released after the due date of the project.

- You don't have to keep the page open on Vocareum while the scripts are running.

- Be careful and avoid multiple submissions of large files to Vocareum. Vocareum does not allow students to delete old submissions, and in the past, students have run out of space and been unable to use Vocareum until we got in touch with support and asked them to delete files.