CS440 - Spring 2021 - Project 3

Gal Zandani (gz113) and Vaishnavi Manthena (vm504)

April 10, 2021

Problem 1

Given observations up to time t ($Observations_t$), and a failure searching $Cell_j(Observations_{t+1} = Observations_t \wedge Failure\, in\, Cell_j)$, how can Bayes' theorem be used to efficiently update the belief state? i.e., compute:

$$P\ (Target\ in\ Cell_i \mid Observations_t\ \wedge\ Failure\ in\ Cell_j).$$

The probability that a target is in cell i given the observations up to time 't' is represented by:
$P\ (Target\ in\ Cell_i \mid Observations_t)$.
This represents the prior probability before moving into step t+1. This probability was achieved by updating the initial probability (1/2500) after every step for 't' steps. This updated probability includes all the data from 't' observations. This question asks us to get a posterior probability after a failure in cell j at time t+1. This posterior probability can be derived directly by using the prior probability $P(TargetinCell_i|Observations_t)$. The derivation of the solution is given below:

$$P\ (Target\ in\ Cell_i \mid Observations_t\ \wedge\ Failure\ in\ Cell_j)$$
$$= \frac{P(Target\ in\ Cell_i \wedge Observations_t \wedge Failure\ in\ Cell_j)}{P(Observations_t \wedge Failure\ in\ Cell_j)}$$
$$= \frac{P(Observations_t)*P(Target\ in\ Cell_i \mid Observations_t)*P(Failure\ in\ Cell_j \mid Target\ in\ Cell_i \wedge Observations_t)}{P(Observations_t)*P(Failure\ in\ Cell_j \mid Observations_t)}$$
$$= \frac{P(Target\ in\ Cell_i \mid Observations_t)*P(Failure\ in\ Cell_j \mid Target\ in\ Cell_i \wedge Observations_t)}{P(Failure\ in\ Cell_j \mid Observations_t)}$$

Given that the target is in a Cell, the Failure in $Cell_j$ only depends on this fact and does not depend on all the observations anymore. So the formula can be updated as: $\frac{P(Target\ in\ Cell_i \mid Observations_t)*P(Failure\ in\ Cell_j \mid Target\ in\ Cell_i)}{P(Failure\ in\ Cell_j \mid Observations_t)}$
Calculating $P(FailureinCell_j|Observations_t)$:
There are 2 options for a $Cell_j$. Either the target is in that cell or the target is not in that cell. Marginalizing over these two possible outcomes and using prior probabilities we get: $P(Failure\ in\ Cell_j \mid Observations_t) =$

$P(Target\,in\,Cell_j \mid Observations_t) * P(Failure\,in\,Cell_j \mid Target\,in\,Cell_j)+$
$P(TargetnotinCell_j|Observations_t)*P(FailureinCell_j|TargetnotinCell_j)$

Note: False positive probability is 0.

$P(Failure\ in\ Cell_j \mid Observations_t) =$
$P(Target\,in\,Cell_j \mid Observations_t) * P(Failure\,in\,Cell_j \mid Target\,in\,Cell_j)+$
$P(Target\ not\ in\ Cell_j \mid Observations_t) * 1$

Final Formula: $P\ (Target\ in\ Cell_i \mid Observations_t\ \wedge\ Failure\ in\ Cell_j)$

$= \frac{P(Target\ in\ Cell_i \mid Observations_t) * P(Failure\ in\ Cell_j \mid Target\ in\ Cell_i)}{P(Target\ in\ Cell_j \mid Observations_t) * P(Failure\ in\ Cell_j \mid Target\ in\ Cell_j) + P(Target\ not\ in\ Cell_j \mid Observations_t)}$

So, we can solve the equation by using a simple Bayesian formula with the prior probabilities. Note:

$P(Target\ in\ Cell_i \mid Observations_t)$ = prior probability = current belief we have in $cell_i$

$P(TargetnotinCell_j|Observations_t) = 1 - P(TargetinCell_j|Observations_t)$

The $P(FailureinCell_j|TargetinCell_i) =$ is the false negative probability given to us in the case where i and j are equal. When i and j are not equal this probability is 1 because the probability of false positive is 0.

In conclusion, if we searched $cell_x$ and did not find the target, the belief of the cells is updated according to the following equation:
Type of $cell_x$ = P(Target not found in $cell_x$| Target is in $Cell_x$)
So, for $cell_x$:
New Belief $Cell_x = \frac{Type\ of\ cell_x * Old\ belief\ in\ cell_x}{Type\ of\ cell_x * Old\ belief\ in\ cell_x + 1 - Old\ belief\ in\ cell_x}$

And for any other cell, represented by $cell_y$:
Scale Factor $= \frac{1}{Type\ of\ cell_x * Old\ belief\ in\ cell_x + 1 - Old\ belief\ in\ cell_x}$
New Belief in $Cell_y$ = Old Belief in $cell_y$ * scale factor

Problem 2

Given the observations up to time t, the belief state captures the current probability the target is in a given cell. What is the probability that the target will be found in Cell i if it is searched:

$$P\,(Target\,found\,in\,Cell_i \mid Observations_t)?$$

Marginalizing over whether or not the target is in the cell and using prior probabilities, we can derive the equation as below:

$$P\,(Target\,found\,in\,Cell_i \mid Observations_t)$$

$= P(target\,is\,in\,Cell_i \mid Observations_t) * P(target\,is\,found \mid target\,in\,Cell_i)$
$+P(targetisnotinCell_i|Observations_t)*P(targetisfound|targetnotinCell_i)$
$= P(target\ is\ in\ Cell_i \mid Observations_t) * P(target\ is\ found \mid target\ in\ Cell_i)$
$+P(targetisnotinCell_i|Observations_t)*0 = P(targetisinCell_i|Observation_t)*$
$P(target\ is\ found \mid target\ in\ Cell_i)$
$= Current\,belief\ in\,Cell_i\ *(1 - false\,negative\,probability\ for\ that\ cell\ )$

Problem 3

Consider the following situation. Your agent is dropped into the map at a random location and wants to find the target as quickly as possible. At every time step, the agent can either a) search the current cell the agent is in, or b) move to one of the immediate neighbors (up/down/left/right). We can consider the following basic agents:
– Basic Agent 1: Iteratively travel to the cell with the highest probability of containing the target, search that cell. Repeat until target is found.
– Basic Agent 2: Iteratively travel to the cell with the highest probability of finding the target within that cell, search that cell. Repeat until the target is found.
For both agents, ties in probability between cells should be broken based on shortest distance (minimal Manhattan distance), and broken at random between cells with equal probability and equal shortest distance. The final performance of an agent is taken to be 'total distance traveled' + 'number of searches', and we want this number to be as small as possible. Generate 10 maps, and play through each map (with random target location and initial agent location each time) 10 times with each agent. Which agent is better, on average?

We wrote a method to construct both of the agents specified, basic_agent()-the logic of the agents, and a method that runs both of these basic agents using different environments, targets, and initial points as described above, avg_agent()-generating environments, random targets, and random initial points.
After running avg_agent(), we arrived at the following performances:

Run Number	Basic Agent 1	Basic Agent 2	Advanced Agent
1	15105.6	10341.4	2024.8
2	4839.0	3203.0	1903.0
3	11138	8008.7	7251.3
4	9366.4	12595.5	8041.4
5	22076.0	17110.0	5581.0
6	23664.6	20784.7	17308.1
7	28106.0	12884.2	11521.6
8	22128.5	8612	5136.5
9	23535.4	19053.2	17633.8
10	22974.7	10884.7	7829.7
Average of all runs	**18293.42**	**12347.71**	**8423.12**

Therefore, we can conclude that basic agent 2 works better on average. This is because basic agent 1 might waste a lot of travel/search effort in moving to and searching cells in which the probability of finding the target is low.
Note about the results: the data in each row or run is the average score (distance + number of searches) for each agent for 10 games on a given environment with different initial and target positions each time.

We added the performance of our advanced agent in here to compare it to the other two basic agents in the same runs (with the same environment, target, and initial point). These results will be used in the analysis of the advanced agent in problem 4.

Problem 4

Design and implement an improved agent and show that it beats both basic agents. Describe your algorithm, and why it is more effective than the other two. Given world enough, and time, how would you make your agent even better?

The improved agent we designed follows this logic: Our agent 3 build on to agent 2. After finding the cell with highest probability of finding the target, we create a list of locations in one Manhattan distance path to that cell. The next cell to search is modified to be the first cell in this path whose probability of finding the target is only slightly less than (a difference of 0.01 in their probabilities) that of the cell with the current highest probability of being found. In this case, the agent could just stop at this cell and search it, taking the Manhattan distance to get there into account. The main idea is that when an agent is travelling to a position with highest probability, it would not incur any extra cost in terms of travel to search a cell along the path which also has a good probability. This would just incur one extra (search) cost and we want to do these searches efficiently. So, our improved agent stops along the way to its current destination if it sees a cell that seems good to search.

As we can see in the table above where we ran the two basic agents and the improved agent for 10 different environments and changing the target and initial point to random points in it 10 times. On average, agent 3 does better because it kind of takes into consideration the next step into the future. This is because it stops at a cell with a probability that is close to the current highest probability cell. So, this cell the agent 3 stops at may likely be visited in future. So, its good to just visit it now when it is along the path to our current destination cell.

We can even make the advanced agent perform better by using the Markov Decision process to accurately approach this problem given unlimited resources.

States: current belief array and current position.
Actions at a state: Choosing to explore a cell.
Rewards for taking an action: A function of probability of target being found in that cell (The higher this probability is, the greater the reward) and the Manhattan distance to the cell (The lower this value is, the higher should be the reward)

An action at a given state could lead to two future states: 1)The target is found and we automatically get the reward of winning the game (highest possible reward) 2) The target is not found in which case belief array is updated and we reach a new state with new actions to take.

With unlimited computational resources, we can construct the Bellman equations for all of these numerous potential states and actions. Solving the equations can give us the action (cell to explore) with highest utility. By considering all of these states we are looking into the future to maximum extent and hence making more accurate decisions. This method would give us more accurate results of the beliefs, using the utility function, in each state of the environment and can direct us faster with less amount of work done to the target.

Bonus

Clever Acronym

The clever acronym we thought of for our advanced agent is pull over agent. Our logic works that we find the highest belief probability and on the way of the Manhattan distance we search at close probabilities to the highest one (in a difference of 0.01 or less). The algorithm stops on the way to the point with the highest probability to search those points, like pulling over on the way to a certain location your heading towards.

Summary

We both worked on the logic and code for the problems in this project, we brain stormed ideas, and agreed on the solutions we got it. We met on Zoom to write together the code for this project to generate the environment, the two agents algorithms, and the advanced agent. We collaboratively found solutions to issues and bugs we faced while testing and running the code. We both were running test cases on the code outside of our Zoom meetings (generating performances, running different environments with different target and initial point). In this report, we wrote the basic agents and advanced agent sections together. Mostly it was completely collaborative but the below are some individual things we did:

Gal: Got the latex document ready with the relevant sections, debugged the code and found solutions for the issues, and offered ideas or more efficient ways of implementations while working on the code together. I wrote the agents 1 and 2 and added on to the probabilities sections.

Vaishnavi:

Coded for generating environment, basic agent 1, and the helper methods needed for those. Tried to brainstorm equations for questions 1 and 2 in a word doc. Ran half of the performance results. Brainstormed what to do with unlimited resources for improved agent along with Gal and inputted ideas into that part of question 4.

We both wrote some methods of the code, debugged separately, and contributed to the answers on this report. This write up was written by both Gal and Vaishnavi, each one of us was responsible for some questions and we added on to each other's answers.

I read and abided by the rules laid out, I have not used anyone else's work for our project, my work is only my own and my partner's.

Gal

Vaishnavi