CS 189 / 289 Introduction to Machine Learning
Fall 2023 Jennifer Listgarten, Jitendra Malik HW6

Due 11/17/23 11:59 pm PT

- Homework 6 consists of both written and coding questions.
- We prefer that you typeset your answers using LaTeX or other word processing software. If you haven't yet learned LaTeX, one of the crown jewels of computer science, now is a good time! Neatly handwritten and scanned solutions will also be accepted for the written questions.
- In all of the questions, **show your work**, not just the final answer.

Deliverables:

1. Submit a PDF of your homework to the Gradescope assignment entitled "HW 6 Write-Up". Submit your code to the Gradescope assignment titled "HW 6 Code". **Please start each question on a new page.** If there are graphs, include those graphs in the correct sections. **Do not** put them in an appendix. We need each solution to be self-contained on pages of its own.
    - In your write-up, please state with whom you worked on the homework. This should be on its own page and should be the first page that you submit.
    - In your write-up, please copy the following statement and sign your signature underneath. If you are using LaTeX, you can type your full name underneath instead. We want to make it *extra* clear so that no one inadvertently cheats.

      "I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted."
    - **Replicate all of your code in an appendix**. Begin code for each coding question on a fresh page. Do not put code from multiple questions in the same page. When you upload this PDF on Gradescope, *make sure* that you assign the relevant pages of your code from the appendix to correct questions.

1 Robot Localization with the Viterbi Algorithm

Consider the following semi-realistic model of a robot's motion: we say that the state variable X_t represents the robot location on a grid-like environment (like the one given below). Suppose the grid has r rows and c columns. Then, $X_t \in \{(i, j) \mid 0 \leq i < r,\ 0 \leq j < c\}$ is the domain of this variable, i.e., there are $N = r \cdot c$ total states. For the particular environment in the figure below, we have $r = 16$ and $c = 16$ so $N = 16 \cdot 16 = 256$.

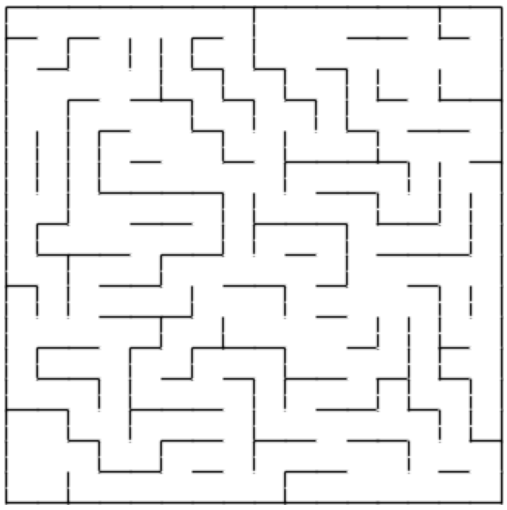

Figure 1: This is just one possible instance of the environment! The actual assignment uses a maze generation algorithm to create these environments from scratch non-deterministically.

This environment is enclosed by walls and has both horizontal and vertical walls scattered throughout between pairs of adjacent cells. Let NEIGHBORS(q) be the set of empty cells that are adjacent to state q and are reachable from it (i.e., there is no wall between q and its neighbor), and let $N(q)$ be the size of this set. Then, we model the robot motion in this environment as a random walk where it transitions to one of its neighboring cells uniformly at random, i.e.,

$$[A]_{qq'} = P(X_{t+1} = q' \mid X_t = q) = 1/N(q) \text{ if } q' \in \text{NEIGHBORS}(q) \text{ else } 0$$

We don't know where the robot starts, so we will assume that the initial probability distribution π over the states is uniform; that is $P(X_0 = (i, j)) = 1/N$.

The robot has 4 sensors mounted on it, one pointing in each of the 4 directions. Each sensor returns a single bit output, which is a 0 when a sensor reads a wall directly in front of it and 1 if it reads open space. The observation at a timestep t is a 4-bit sequence returning the sensor values from the left, right, up and down sensors respectively, in that particular order. For example, for the environment in the figure above, if the robot is present in the top left cell, it's sensors should read 1011 because there is a wall in the left, up and down directions. Similarly, if the robot is present in the top right cell, it's sensors should read 0110 since there are walls in the right and up directions. We note that the number of 1s that a sensor should read in state q is precisely $N(q)$.

However, the sensors are faulty! Each sensor, independent of the others and the timestep, can fail with probability ϵ and incorrectly flip its output. Therefore, there are 2^4 possible observations that

we can read off of the robot, each occurring with different probabilities:

- The probability of getting all 4 bits right is $(1-\epsilon)^4$
- The probability of getting all 4 bits wrong is ϵ^4
- Suppose there are d discrepancies between the true sensor reading and the returned sensor reading: the probability of this observation is $(1-\epsilon)^{4-d}\epsilon^d$

The true robot states are hidden from us, but we can observe the sensor readings every time the robot moves to a new state. As such, we can model the scenario above using a hidden markov model!

(a) Our goal is to predict the true robot hidden states from the observed sensor readings. Given a sequence of T observations, implement the Viterbi algorithm to decode and find the most likely sequence of hidden states. You have all the information you need to compute the transition probabilities $a_{qq'}$ and observation emission probabilities $b_q(o_t)$.

The starter code contains two files:

- `env.py`: This file encodes the dynamics of the environment. In particular, it defines the `Env` class which creates the underlying environment, tracks the true robot hidden state and emits sensor reading. You may use any functions that don't start with an underscore ("_") when implementing the Viterbi algorithm.
- `hmm.py`: This is where all of your code should go.

Since your goal is to return the best set of hidden states, don't forget to also backtrace your way through the most likely sequence. Feel free to reference the handout on Prof. Jurafsky's lecture notes on HMMs that is linked on the course website. **Include all of your code below and submit `hmm.py` to the Gradescope assignment.**

(b) Now, we will vary the error parameter ϵ to values like 0, 0.05, 0.1, 0.2, 0.25 and 0.5. For each value of epsilon, we will generate and decode 100 sequences of 100 observations each. Plot the cumulative accuracy (i.e., the accuracy of the first i predictions when compared to the first i hidden states, for each $i \in [1, 100]$, for a single trajectory), averaged over all 100 trajectories, for each ϵ. Describe any trends you see in the plot. Do these align with your intuition?

2 Markov Decision Processes and Value Computations

In this question, you'll be reasoning about maximizing reward when sequentially making decisions in a Markov Decision Process (MDP), as well as about the Bellman equation - the central equation to solving and understanding MDPs.

Consider the classic gridworld MDP, where an agent starts in cell (1, 1) and navigates around its environment:

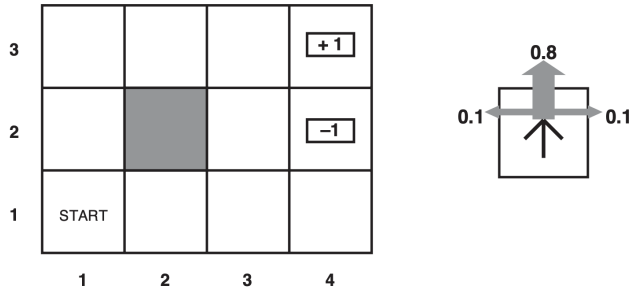

Figure 2: Gridworld MDP with stochastic transition probabilities.

In this world, the agent can take 4 actions in each cell: Up, Down, Left, or Right. The cells are indexed by (horizontal, vertical); that is, cell (4, 1) is in the bottom-right corner. The transition probabilities of the world work as follows: if the agent takes an action, it will move to the cell in the action's direction with probability 0.8, and it will slip to the action's relative right or left direction with probability 0.1 each. If the action (or slipping direction) is into a cell with no traversable tile (i.e., either a border or the wall in cell (2, 2)), that action keeps the agent at the cell it is currently at. For example, if the agent is in (3, 1), and it takes the action Up, it will land in cell (3, 2) with probability 0.8, in cell (2, 1) with probability 0.1, and (4, 1) with probability 0.1. If the agent is in cell (1, 3) and takes the action Right, it will land in cell (2, 3) with probability 0.8, in cell (1, 2) with probability 0.1, and (1, 3) with probability 0.1. When the agent reaches either of the defined reward states, at cells (4, 2) and (4, 3), the agent incurs the corresponding reward and the episode terminates.

Recall the Bellman equation for computing the *value*, $V(s)$ of each state in an MDP, where we have a set of actions A, a set of states S, a reward value for each state $R(s)$, the transition dynamics of our world $P(s'|s,a)$, and a discount factor γ:

$$V(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P(s'|s,a)V(s')$$

Lastly, we'll refer to policies as $\pi(s) = a$, where a policy π prescribes an action to take when in a given state.

(a) Consider an agent that starts in cell (1, 1) and takes the actions Up, Up in timesteps 1 and 2, respectively. Calculate which cells can be reached in each timestep from this action sequence and with what probabilities. How is this calculation similar to the prediction task for an HMM?

(b) Consider the reward function, $R(s)$, for all states that currently don't have a reward assigned to them (every cell except for (4, 2) and (4, 3).) Define what an optimal policy would be for an agent given the following reward values: (i.) $R(s) = 0$, (ii.) $R(s) = -2.0$, and (iii.) $R(s) = 1.0$. You may assume the discount factor to be a number arbitrarily close to 1, e.g. 0.9999. It may be helpful to draw out the gridworld and actions that should be taken at each state (remember that policies are defined over all states in an MDP!)

(c) Sometimes MDPs are formulated with a reward function $R(s, a)$ that depends on the action taken, or with a reward function $R(s, a, s')$ that also depends on the outcome state. Write out the Bellman equations for these formulations.

3 Jack's Car Rentals

This problem is inspired by an exercise from Sutton and Barto's Reinforcement Learning textbook.

Jack manages two locations for a nationwide car rental company. Each day, some number of customers arrive at each location to rent cars. If Jack has a car available, he rents it out and is credited \$10 by the national company. If he is out of cars at that location, then the business is lost. Cars become available for renting the day after they are returned. To help ensure that cars are available where they are needed, Jack can move them between the two locations overnight, at a cost of \$2 per car moved. We assume that the number of cars requested and returned at each day follow Poisson distributions. Suppose the rental requests and locations 1 and 2 follow Poisson distributions with rate parameters 3 and 4. Similarly, suppose the number of cars returned at locations 1 and 2 follow Poisson distributions with rate 3 and 2.

To simplify the problem slightly, we assume that there cannot be no more than 20 cars at each location (any additional cars are returned to the nationwide company and, thus, disappear completely from the problem), and a maximum of 5 cars can be moved from one location to the other in one night. We take the discount rate to be $\gamma = 0.9$ and formulate this as a finite MDP, where the time steps are days, the state is the number of cars at each location at the end of the day, and the actions are the net numbers of cars moved between the two locations overnight (a positive action indicates cars moving from location 1 to location 2 while a negative action indicates the reverse direction).

We define the policy $\pi(a \mid s)$ as the action Jack takes given the state s. In this problem, we will use Policy Iteration and Value Iteration to compute the optimal policy. **Add your code to the appendix of your writeup and also submit it to the Gradescope coding assignment.**

(a) Both algorithms depend on the Bellman equation: the Policy Iteration algorithm uses the Bellman expectation equation and the Value Iteration algorithm uses the Bellman optimality equation. In each case, we need to compute the sum below. Show that

$$\sum_{s',r} P(s', r \mid s, a)[r + \gamma V(s')] = \mathbb{E}[r \mid s, a] + \gamma\, \mathbb{E}[V(s') \mid s, a]$$

(b) Implement the Policy Iteration algorithm in the starter code, and use it to derive the optimal value function V^* and policy π^* that Jack should follow to maximize his returns.

Each location can have anywhere from 0 to 20 cars so the state space can be represented as $\mathcal{S} = [0, \ldots, 20] \times [0, \ldots, 20]$. As such, the policy $\pi : \mathcal{S} \to [-5, \ldots, 5]$ can be represented by a 21×21 2D matrix where π_{ij} is the action taken at state (i, j). The same goes for the value function $V : \mathcal{S} \to \mathbb{R}$. Plot the π and V matrices returned by the Policy Iteration algorithm after it converges. Describe what these plots represent. Do these align with your intuition?

(c) Implement the Value Iteration algorithm in the starter code, and use it to derive the optimal value function V^* and policy π^* that Jack should follow to maximize his returns.

Plot the π and V matrices returned by the Value Iteration algorithm. Does it return the same π and V as the Policy Iteration algorithm?

Now, suppose that one of Jack's employees at the first location rides a bus home each night and lives near the second location. They are happy to shuttle one car to the second location for free. Each additional car from location 1 to location 2 will still cost \$2, as do all the cars moved in the other direction. In addition, Jack has limited parking space at each location. If more than 10 cars are kept overnight at a location (after any moving of cars), then an additional cost of \$4 must be incurred to use a second parking lot (independent of how many cars are parked there) at said location.

These sorts of non-linearities and arbitrary dynamics often occur in real problems and cannot easily be handled by optimization methods other than dynamic programming.

(d) Repeat part (b) with the added non-linearities. How does the new policy compare against the previous policy? Does it align with your intuition?

(e) Repeat part (c) with the added non-linearities. Does Value Iteration still return the same policy as Policy Iteration?