COMS W4701: Artificial Intelligence, Spring 2025

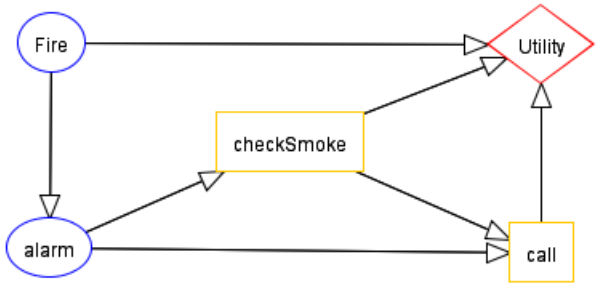
Homework 5

Instructions: Compile all written solutions for this assignment in a single, typed PDF file. Coding solutions may be directly implemented in the provided Python file(s). **Do not modify any filenames or code outside of the indicated sections.** Submit all files on Gradescope in the appropriate assignment bins, and make sure to **tag all pages for written problems**. Please be mindful of the deadline and late policy, as well as our policies on citations and academic honesty.

For Problems 1-3, you should perform all computations yourself, either by hand or using Python (please attach all notebooks/code). Print both intermediate and final results, including distributions, factors, or estimated parameters. You do not have to separately copy over the final answers to your writeup as long as you clearly indicate them in the notebooks.

Problem 1: Fire Alarm Decision Network (20 points)

The following Bayes net is a modified version of the "Fire Alarm Decision Problem" from the Sample Problems of the Belief and Decision Networks tool on AIspace. All variables are binary, and the chance node CPTs as well as utility values are shown below.



Fire	CheckSmoke	Call	Utility
True	True	True	−550
True	True	False	−850
True	False	True	−500
True	False	False	−800
False	True	True	−220
False	True	False	−20
False	False	True	−200
False	False	False	0

Fire	Pr(Fire)
True	0.01
False	0.99

Fire	Alarm	Pr(Alarm | Fire)
True	True	0.95
True	False	0.05
False	True	0.01
False	False	0.99

(a) (4 pts) Compute and show the joint factor over all variables and the utility. Then eliminate any variable that is not a parent of a decision node, and show the resulting factor.

(b) (4 pts) Determine the optimal decision functions for Call and CheckSmoke. Compute the maximum expected utility of the resulting policy.

(c) (4 pts) Does the decision function for Call actually depend on both of its parent nodes? Explain whether the resulting policy would differ if you eliminated the decision variables in a different order than what you used in (b).

(d) (4 pts) Now suppose that the Call decision also depends on Fire; in other words, we get to observe Fire prior to making the decision. Determine the optimal decision functions for Call and CheckSmoke. Compute the maximum expected utility of the resulting policy.

(e) (4 pts) Compute the VPI of Fire for the Call decision and provide an interpretation for this quantity. Why might it be beneficial to be able to observe Fire directly, in addition to (or instead of) relying on the "after-effects" of a Fire event?

Problem 2: Wandering Robot (20 points)

A robot is wandering around a room with some obstacles, labeled as # in the grid below. It can occupy any of the free cells labeled with a letter, but we are uncertain about its true location and so we keep a belief distribution over its current location. At each timestep it moves from its current cell to a neighboring free cell in one of the four cardinal directions with uniform probability; it cannot stay in the same cell. For example, from A the robot can move to either B or D with probability $\frac{1}{2}$, while from E it can move to B, D, or F, each with probability $\frac{1}{3}$.

A	B	C
D	E	#
#	F	G

The robot may also make an observation after each transition, returning what it sees in a random cardinal direction. Possibilities include observing #, "wall", or "empty" (for a free cell). For example, in D the robot observes "wall", # (each with probability $\frac{1}{4}$), or "empty" (probability $\frac{1}{2}$).

(a) (4 pts) Suppose the robot wanders around for a long time without making any observations. What is the stationary distribution π over the robot's predicted location?[1]

(b) (4 pts) The robot's sensors just started working. Take the stationary distribution that you found above to be $\Pr(X_0)$. Starting from this belief state, the robot makes one transition and observation e_1 = "wall", followed by a second transition and observation e_2 = #. What are the belief distributions $\Pr(X_1 \mid e_1)$ and $\Pr(X_2 \mid e_1, e_2)$?

(c) (4 pts) Compute the joint distribution $\Pr(X_1, X_2 \mid e_1, e_2)$. You can either write the result as a matrix A, where $A_{ij} = \Pr(X_1 = i, X_2 = j \mid e_1, e_2)$, or just list the joint probability values that are nonzero. What is the most likely *sequence*(s) of states over the two timesteps?

(d) (4 pts) Now suppose we perform localization using a particle filter instead. We initialize the filter with seven particles, one in each free cell, and perform the transition step. Identify

[1] `numpy.linalg.eig` in Python returns a 1D array of eigenvalues and a 2D array where each column is a corresponding eigenvector. Remember that eigenvectors may not sum to 1 by default.

a resultant particle distribution in which a *maximal* number of cells are now-particle free. Identify a different distribution in which a *minimal* number of cells are now particle-free.

(e) (4 pts) Suppose we have two particles in state A, three in B, one in E, and one in F. For the observation of "wall", compute the set of weights that would be applied to a particle in these states. Also compute the *distribution* from which we would resample the particles.

Problem 3: Naïve Bayes (20 points)

Consider the following data set. There are two features x_1 and x_2 with possible values in $\{0, 1, 2\}$, and a class variable y with values 0 and 1.

Sample	x_1	x_2	y
1	2	0	0
2	2	2	0
3	1	0	0
4	1	2	0
5	2	2	1
6	1	1	1
7	1	2	1
8	0	2	1

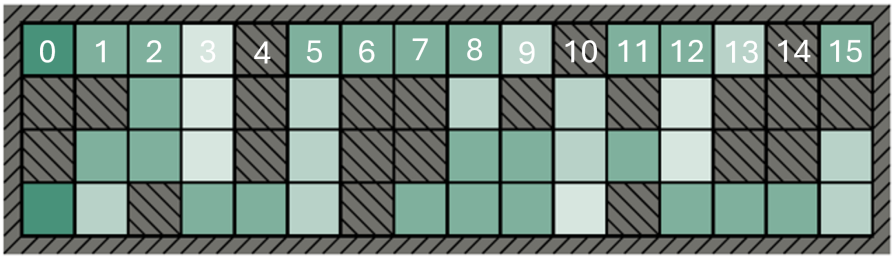
(a) (4 pts) The maximum likelihood class CPT is $\Pr(Y) = (0.5, 0.5)$. Estimate the feature CPTs $\Pr(X_1|Y)$ and $\Pr(X_2|Y)$ with Laplace smoothing of $\alpha = 1$.

(b) (4 pts) Using the learned parameters, predict the value of y for each of the eight samples. What is the accuracy rate?

(c) (6 pts) Suppose that we find out that the class values y in the data set may not be correct. We can use the estimated probabilities in (a) as a starting point for expectation-maximization. Compute the expected counts $\Pr(Y|x_1, x_2)$ for each sample (x_1, x_2) in the data set.

(d) (6 pts) Perform the maximization step and find the new CPTs $\Pr(Y)$, $\Pr(X_1|Y)$, and $\Pr(X_2|Y)$. Do not use Laplace smoothing for this part.

Problem 4: Grid World Localization (40 points)



You will be implementing grid world localization (see Section 14.3.2 in AIMA for a similar example). An agent is moving around and gathering observations on a grid, where each cell may be either passable or not passable (e.g., a wall). We do not know exactly where the agent is, so we will have a belief state over each cell of the grid. For simplicity, the belief state will also include

the blocked cells, which will always have probability 0. You will be completing the `Gridworld_HMM` class to implement the environment model, as well as the forward and particle filter algorithms.

We may test your code on different grid worlds. Please do not hardcode any information about the specific example shown above, including sizes and wall locations.

4.1: Transition Probabilities (8 points)

The transition model is such that the agent may either stay in the current cell or move to an adjacent free cell, all with uniform probability. Thus, if the set S includes all adjacent free neighbors of state x in addition to x itself, then $\Pr(x'|x) = \frac{1}{|S|}$ for all $x' \in S$.

Write the function `initT()`, which should return a $N \times N$ array T, where N is equal to `grid.size` and $T_{ij} = \Pr(x_j|x_i)$. We recommend that you populate T one row (state) at a time, following the cell order enumerated in the figure. You may use the `neighbors()` helper function, which returns a list of all adjacent free cells and the given cell. Be sure to check that the rows of T sum to 1.

4.2: Observation Probabilities (8 points)

From a state x, the agent can make an observation e, which is an integer value between 0 and 15 (inclusive). The 4-bit representation of e indicates whether each of the four adjacent neighbors is a blocked (1) or free (0) cell. The order of the bits is north, east, south, west (NESW). For example, the correct observation for the lower-right corner cell in the figure would be the value 6, with bit representation 0110 indicating free, blocked, blocked, free cells in NESW order.

However, the observations are also *noisy*; there is a ϵ probability that each bit is independently wrong. Let e^* be the correct observation for state x, e be the actual observation, and d be the number of mismatched bits between e and e^*. Then $\Pr(e|x) = (1-\epsilon)^{4-d}\epsilon^d$. Following the example above, $\Pr(e = 6|x) = (1-\epsilon)^4$, while $\Pr(e = 0|x) = \Pr(e = 15|x) = (1-\epsilon)^2\epsilon^2$.

Write the function `initO()`, which should return a $16 \times N$ array O where $O_{ij} = \Pr(e = i|x_j)$. Again, it may be easiest to populate the array one row (state) at a time. First find the "correct" observation value for each state. Then for each possible observation value between 0 and 15, compute the discrepancy[2] and insert the observation probability into the array.

4.3: Forward Algorithm (6 points)

Now that we have the model defined, we can perform inference. `forward()` is given a list of `observations` and the initial belief state `init`. It should return a $T \times N$ NumPy array, where T is the number of observations and N is equal to `grid.size`, containing the *normalized* $\Pr(X_t|e_{1:t})$ belief states in each row.

4.4: Particle Filter (12 points)

Alternatively, we can use a particle filter to perform approximate inference. We will store the set of particles in an array `self.particles`, such that `self.particles[i] = s` indicates that particle i is in state s. We will also have an identically sized array storing weight values of each particle.

[2] You can do so in Python using `bin(x^y).count('1')`.

The filter will be implemented via three helper and one main function. The `transition()` function should sample a successor state for each particle according to the transition matrix, and update `self.particles` in place[3]. The `observe()` function should compute the weights for each particle given an observation value, and update `self.weights` in place. Finally, the `resample()` function should generate a new set of particles using the current set of weights. This should update `self.particles` in place, and also reset all `self.weights` to 1.

You will put then put everything together in `particle_filter()`. Like `forward()`, it will perform inference by executing the three procedures above for each individual observation. It should also return a $T \times N$ array, in which each row contains the *counts* of particles in each state (think of this as an approximation of the belief distribution).

4.5: Analysis (6 points)

When you are finished, you can test your implementation by invoking `python main.py`. It can run under four "modes", specified by the argument `-m`, as described below:

- Mode 0 (default): Run the forward algorithm for `-t` timesteps (default 50), `-n` episodes (default 100), and a set of ϵ values. Five different values of ϵ $(0.4, 0.2, 0.1, 0.05, 0.02)$ are tested. When finished, a plot showing the average localization error[4] will be shown.
- Mode 1: Run the particle filter for `-t` timesteps, `-n` episodes, a set of ϵ values, and `-p` (default 30) particles. When finished, a plot showing the average localization error will be shown.
- Mode 2: Run the forward algorithm for `-t` timesteps with ϵ equal to `-e` (default 0.02). An animation will pop up showing the agent's true location and the estimated belief distribution colored on the grid over time. Bright yellow corresponds to higher probabilities.
- Mode 3: Run the particle filter for `-t` timesteps with ϵ equal to `-e` and `-p` particles. An animation similar to that in Mode 2 will pop up here. There will also be values showing the number of particles on certain cells (cells without values have 0 particles).

After verifying that your implementation is working, briefly address the following questions.

1. Show the error plot for Mode 0 in your writeup. How does the value of ϵ affect the performance of the forward algorithm?
2. Show the error plot for Mode 1. How does the performance of the particle filter generally compare to exact inference using the forward algorithm?
3. Run Mode 3 a few times—you should see both successes and failures when using the particle filter. Briefly explain why tracking failures occur even when the particles are relatively "confident" about the agent's location.

Coding Submission

Please submit the completed `gridworld_hmm.py` file under the HW5 Coding bin on Gradescope.

[3] `np.random.choice()` should be helpful here.

[4] This is computed as the total Manhattan distance between the agent's belief distribution and its true location (a distribution with a value of 1 in its true location and 0 elsewhere).