Artificial Intelligence	**Assignment 5**
comp440/comp557 Fall 2016	**12 October 2016**

This assignment is due October 28 at 8 pm on Canvas. Download `assignment5.zip` *from Owlspace. There are four problems worth a total of 125 points. Problems 1 and 2 require written work only, Problem 3 requires Python code and a writeup. All written work should be placed in a file called* `writeup.pdf` *with problem numbers clearly identified. All code should be included in* `exactInference.py` *and* `particleFilter.py` *at the labeled points. For Problem 3, please run the autograder using the command line* `python grader.py` *and report the results in your writeup. For problem 4, which is extra credit for comp440 and required for comp557, you will need to have a writeup and an implementation of a Bayesian network. Put your written work in* `writeup.pdf` *and your election network code in a separate folder called* `BN`*. Zip up* `exactInference.py`*,* `particleFilter.py`*,* `writeup.pdf` *and the* `BN` *folder, and submit it as an attachment on Canvas by the due date/time.*

1 Bayesian networks for astronomy (25 points)

Two astronomers in two different parts of the world, make measurements M_1 and M_2 of the number of stars N in some small region of the sky, using their telescopes. Normally, there is a small possibility of error (e) of plus or minus one star in a measurement. Each telescope can be (with a much smaller probability f) be badly out of focus (events F_1 and F_2), in which case the scientist will undercount by three or more stars (or, if N is less than three, fail to detect any stars at all). Consider the three networks shown in Figure 1.

- (3 points) Which of these networks correctly (but not necessarily efficiently) represent the above information?
- (2 points) Which is the best network and why?
- (8 points) Construct the conditional probability distribution $P(M_1|N)$, for $N \in \{1, 2, 3\}$ and $M_1 \in \{0, 1, 2, 3, 4\}$. Each entry in the conditional distribution should be expressed as a function of the parameters e and/or f. Fill in your answer in the table below.

	$N = 1$	$N = 2$	$N = 3$
$M_1 = 0$			
$M_1 = 1$			
$M_1 = 2$			
$M_1 = 3$			
$M_1 = 4$			

- (7 points) Suppose $M_1 = 1$ and $M_2 = 3$. What are the possible numbers of stars N, if we assume no priors on the values of N?
- (5 points) What is the *most likely* number of stars, given these observations? Explain how to compute this, or, if it is not possible to compute, explain what additional information is needed and how it would affect the results.

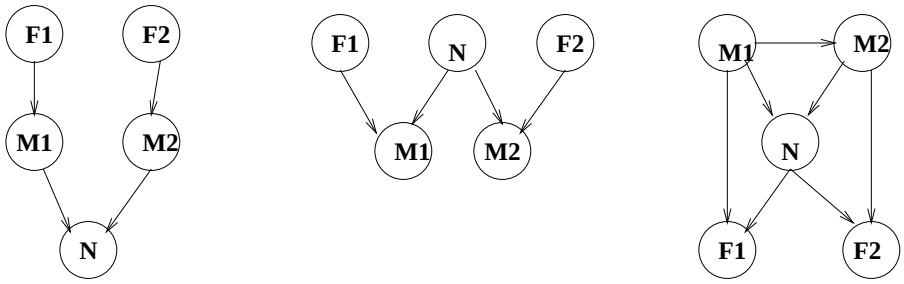

Figure 1: Three possible networks for the telescope problem.

2 Gibbs Sampling (20 points)

Consider the query $P(Rain|Sprinkler = true, WetGrass = true)$ in the Bayesian network shown in Figure 14.12 of your textbook. You will compute this probability exactly, and then estimate it using a Gibbs sampler.

- (5 points) Calculate $P(Rain|Sprinkler = true, WetGrass = true)$.
- (15 points) Now set up a Markov chain to approximate this probability.
  - (2 points) How many states does this Markov chain have?
  - (5 points) Compute the transition matrix Q for this Markov chain, containing $q(y, y')$ for all states y, y' .
  - (3 points) What is Q^n as $n \to \infty$? How does it relate to your answer in part (a)?
  - (5 points) Write a function in Python to simulate your Markov chain. Run it for N steps, to estimate the probability $P(Rain = true|Sprinkler = true, WetGrass = true)$. What is your estimate of this probability when N = 1000, 5000, and 10000. Make a guess for the burn-in period of this Markov chain (hint: for what n does Q^n stabilize?). stabilize)?

3 Tracking cars by particle filtering (50 points)

A study by the World Health Organization found that road accidents kill a shocking 1.24 million people a year worldwide. In response, there has been great interest in developing autonomous driving technology that can can drive with precision and reduce this death toll. Building an autonomous driving system is an incredibly complex endeavor. In this assignment, you will focus on the sensing system, which allows us to track other cars based on noisy sensor readings.

Getting started

Let us start by trying to drive manually:

You can steer by either using the arrow keys or 'w', 'a', and 'd'. The up key and 'w' accelerates your car forward, the left key and 'a' turns the steering wheel to the left, and the right key and 'd' turns the steering wheel to the right. Note that you cannot reverse the car or turn in place. Quit by pressing 'q'. Your goal is to drive from the start to finish (the green box) without getting in an accident. How well can you do on crooked Lombard street without knowing the location of other cars? Don't worry if you aren't very good; we were only able to get to the finish line 4/10 times. This 60% accident rate is pretty abysmal, which is why we're going to build an AI system to do this.

Flags for python drive.py:

- -a: Enable autonomous driving (as opposed to manual).
- -i Inference-method: Use `none`, `exactInference`, `particleFilter` to (approximately) compute the belief distributions.
- -l map: Use a map (e.g. `small` or `lombard`)
- -d: Debug by showing all the cars on the map.
- -p: All other cars remain parked (so that they don't move).

Modeling car locations

We assume that the world is a two-dimensional rectangular grid on which your car and other cars reside. At each time step t, your car gets a noisy estimate of the distance to each of the cars. As a simplifying assumption, we assume that each of the K other cars moves independently and that the sensor reading for each car is also independent. Therefore, in the following, we will reason about each car independently (notationally, we will assume there is just one other car).

At each time step t, let $c_t \in \Re^2$ be a pair of coordinates representing the actual location of the other car (which is unobserved). We assume there is a probability distribution $p(c_t|c_{t-1})$ which governs the other car's movement. Let $a_t \in \Re^2$ be your car's position, which you observe and also control. To minimize costs, we use a simple sensing system based on a microphone. Specifically, the microphone provides us with d_t, which is a Gaussian random variable with mean equal to the distance between your car and the other car and variance σ^2 (in the code, σ is `Const.SONAR_STD`, which is about two-thirds of the length of the car). That is

$$d_t \sim \mathcal{N}(||a_t - c_t||, \sigma^2)$$

For example, if your car is at $a_t = (1, 3)$ and the other car is at $(4, 7)$, then the actual distance is 5 and d_t might be or 4.6 or 5.2, etc. Use `util.pdf(mean, std, value)` to compute the probability density function (PDF) of a Gaussian with given `mean` and standard deviation `std`, evaluated at `value`.

Your job is to implement a car tracker that (approximately) computes the posterior distribution $p(c_t|d_1, \ldots, d_t)$ (your belief of where the other car is based on your measurements $d_1, \ldots, d_t$) and

update it for each $t = 1, 2, \ldots$. We will take care of using this information to actually drive the car (set a_t as to avoid collision with the other car), so you don't have to worry about this part.

To simplify things, we will discretize the world into tiles, each 30 pixels a side. Each tile is represented by a 2-tuple of integers (r, c) for integers $0 \leq r < numRows$, and $0 \leq c < numCols$, where $numRows$ and $numCols$ are the number of rows and columns in the discretized grid world. We store a probability for each tile (in the code, this is `self.belief.getProb((r, c))`. To convert from a tile to a location value, use `util.rowToY(r)` and `util.colToX(c)`. In Problems 3.1 and 3.2, you will implement `ExactInference`, which computes a full distribution over tiles (r, c). In Problem 3.3, you will implement `ParticleFilter`, which works with a particle-based representation of this distribution.

Problem 3.1: Incorporating observations (10 points)

In this part, we assume that the other car is stationary, i.e., $c_t = c_{t-1}$ for all time steps t. Implement the function `observe` that takes the current posterior probability $p(c_t|d_1, \ldots, d_{t-1})$ (`self.beliefs` in `ExactInference`), a new observation d_t, and sets `self.beliefs` to be $p(c_t|d_1, \ldots d_t)$ as follows:

$$p(c_t|d_1, \ldots, d_t) \propto p(d_t|c_t)p((c_t|d_1, \ldots, d_{t-1})$$

Fill in the `observe` method in the `ExactInference` class of `exactInference.py`. This method should update the posterior probability of each tile given the observed noisy distance. After you're done, you should be able to find the location of a stationary car by circling it (`-p` means cars don't move):

```
python drive.py -a -p -d -k 1 -i exactInference
```

You can turn off `-a` to manually drive. Run the autograder (`grader.py`) to check that your computations are correct.

Problem 3.2: Incorporating observations and transitions (10 points)

Now, let us consider the case where the other car is moving according to transition probabilities $p(c_t|c_{t-1})$. We have provided the transition probabilities for you in `self.transProb`. Specifically, `self.transProb[(oldTile, newTile)]` is the probability of the other car being in `newTile` at time step t given that it was in `oldTile` at time `t-1`.

In this part, you will implement a function `elapseTime` that takes the posterior probability of the previous time step $p(c_{t-1}|d_1, \ldots, d_{t-1})$ (`self.belief` in `ExactInference`) and sets `self.belief` to be

$$p(c_t|d_1, \ldots, d_t) \propto p(c_t|c_{t-1})p(c_{t-1}|d_1, \ldots, d_{t-1})$$

Finish `ExactInference` by implementing the `elapseTime` method in `exactInference.py`. When you are all done, you should be able to track a moving car well enough to drive autonomously:

```
python drive.py -a -d -k 1 -i exactInference
```

You can also drive autonomously in the presence of more than one car:

```
python drive.py -a -d -k 3 -i exactInference
```

and even down Lombard:

```
python drive.py -a -d -k 3 -i exactInference -l lombard
```

Run the autograder (`grader.py`) to check that your computations are correct.

Problem 3.3: Particle Filtering (30 points)

Though exact inference works well for the small maps, it wastes a lot of effort computing probabilities for cars being unlikely tiles. We can solve this problem using a particle filter which has complexity linear in the number of particles rather than linear in the number of tiles. Implement all necessary methods for the `ParticleFilter` class in `particleFilter.py`. When complete, you should be able to track cars nearly as effectively as with exact inference. Much of the code has been provided for you. Particles have been initialized randomly and the beliefs are automatically updated after you resample the particles. You just need to fill in the `observe`, `resample`, and `elapseTime` functions, which should modify `self.particles`, which is a map from tiles to the number of times that particle occurs.

You should use the same transition probabilities as in exact inference. The belief distribution generated by a particle filter is expected to look noisier compared to the one obtained exact inference.

```
python drive.py -a -i particleFilter -l lombard
```

To debug, you might want to start with the parked car flag `(-p)` and the display car flag `(-d)`.

Run the autograder (`grader.py`) to check that your computations are correct.

Acknowledgements

This exercise is due to Chris Peich at Stanford. We are allowed to use it under the same conditions as the pacman assignment from Berkeley. Please do not post solutions to this homework anywhere online and please limit your discussion of the problem to piazza.

4 Bayesian networks for predicting the 2016 presidential elections (extra credit for comp440/required for comp557) (30 points)

Kevin Murphy's Bayes Net toolbox is a Matlab toolbox for building your own Bayesian network. You will need to download it from `https://github.com/bayesnet/bnt`. There is a nice accompanying tutorial on that site. I was able to build and evaluate the Sprinkler network in minutes with this toolbox. There is also a Python package `https://github.com/eBay/bayesian-belief-networks` for building small to medium-scale Bayesian networks. It ran out of the box on my Mac and some

nice examples are already built in. You can use one of these two tools, or any other Python-based tool that you can find on the Web (please write down the URL where you found it and provide a brief review of the ease of use of the tool).

This problem is an exercise in knowledge engineering using Bayesian networks. You have been hired as a special consultant for the upcoming presidential elections. Your task is to build a predictive Bayesian network that will allow your team to figure out which candidate someone will vote for, given readily available information about the person. You can assume that information from public sources like voter registration records, as well as the results of polls conducted by reputed organizations, are available to you. Your network will be used to predict not just how a particular individual will vote, but how segments of the population with certain characteristics will vote. This prediction will be used by your team to make critical decisions about which issues to push on, what advertising strategy to use, whether to pump money for campaigning in specific states, which locations to include in your candidate's speeches, which special interest group endorsements to seek, etc..

In order to design a Bayesian network, you will need output variables and evidence variables. The output variable for this problem is whether an individual will vote for Trump, Clinton or a third-party candidate. The evidence variables include information about age, gender, race, education level and income. You can also assume that you know about some of the organizations a voter belongs to (e.g., NAACP, ACLU, NRA, AARP), whether the voter owns a home, whether the voter lives in a suburb or the inner city, whether the voter has ever been on welfare, whether the voter has children attending public schools, and how the voter voted two years ago in the previous presidential elections, their attitudes to current US foreign policy, energy policies, etc. Some of the evidence variables may influence other evidence variables. These evidence variables are suggestive, feel free to add more appropriate ones.

To build a network, you will first need to decide on the set of evidence variables, what their domains are, and discretizing the domains of continuous evidence variables. Then you may need to add intermediate nodes called hidden nodes, which can have labels like *Pro-Obamacare*, *Environmentalist*, or *Conservative*, and *Libertarian* that summarize a voter's position on issues that are relevant to the choice of president. You should add other hidden nodes that capture the central themes of this election. In general, if you find a node has too many parents and the conditional probability table for the node becomes too big, it is a hint that you need an intermediate hidden node. If you are not up on presidential politics, check out extensive analyses available on websites like Nate Silver's `fivethirtyeight.com`, *realclearpolitics.com*, as well as the candidates own websites.

Your network should have about 15 nodes and have a depth of 3 to 4. Build your network using one of the tools (Matlab or Python). Please explain what the nodes represent, and what values they take on. For each node, indicate whether it is always observable, sometimes observable, or never observable. You should have CPTs attached to all the nodes. You should explain your choice of edges. Use your network to make predictions in three test cases that you make up. For each case, pick a set of observations that you might get in real life, instantiate them as evidence, and have the toolbox compute the probabilities at the output node. Report on these two test cases in your write-up.

I will evaluate your network on the basis of whether it is a reasonable representation of the domain, e.g., whether your edges go in the right direction. I do not expect you to come up with completely realistic CPTs for your network. However, your probabilities should have the right order of mag-

nitude (e.g., the probability of being a woman in Houston shouldn't be 2%). Your probabilities should be consistent with one another (e.g., the probability of being a member of a very high income bracket shouldn't be higher than the probability of completing high school).

If the US presidential elections isn't your cup of tea, pick another election that you have knowledge of and write a short background description for me so I can evaluate your network relative to the facts that you provide.