CSCI1410 Fall 2020
Assignment 3: Hidden Markov Models

Code Due Monday, November 2 at 11:59pm ET
Writeup Due Tuesday, November 3 at 11:59pm ET

1 Goals

In this assignment, you will use the forward algorithm to **filter** data in a hidden Markov model (HMM). To filter data in an HMM is to predict the hidden state from those noisy data. We provide a filtering library that runs on toy HMMs; your job will be to run it on a richer HMM that you build yourself. Specifically, the HMM that you will build in this assignment is intended to solve the **localization** problem, where the goal is to infer a location (the hidden state) from noisy sensor data (observations).

2 Silly Premise

George just bought the new iOS Atari game pack! Problem is, he spilled some of his favorite beverage, Dr. Pepper, soaking his entire desk including all his papers and his smartphone. He wants to play the game, but his phone's touchscreen is acting wonky. Your job is to help George filter his touchscreen data using hidden Markov models (HMMs) so that he can control the little Atari dude with his finger. Specifically, you will build an HMM, and then run filtering to predict the location of his finger, given noisy observations.

3 Hidden Markov Models

A **hidden Markov model** comprises a set of states and observations together with an initial probability distribution over those states, a transition model, which describes the probability of transitioning from one state to another, and a sensor model, which describes the probabilities of the various observations at the various states. The word "hidden" in the name describes the fact that the true state is hidden, and only noisy sensor data are observed. The model is Markov, because the transition model is assumed to satisfy the Markov property, namely: The future is independent of the past, given the present: i.e.,

$$\Pr(X_{t+1}|X_t, X_{t-1}, \ldots X_1) = \Pr(X_{t+1}|X_t) \ . \tag{1}$$

Furthermore, the sensor model is assumed to satisfy the following conditional independence assumption: The present observation is independent of past states and observations, given the present state.

$$\Pr(E_t|X_t, X_{t-1}, \ldots, X_1, E_{t-1}, E_{t-2}, \ldots, E_1) = \Pr(E_t|X_t) \ . \tag{2}$$

(Following the notation in the textbook, X_t is a random variable describing the state at time t, and E_t is a random variable describing the evidence observed at time t.)

4 Part 1: Filter (Read Only)

Your goal in past (and perhaps, future) versions of the first part of this assignment was to implement the forward algorithm to filter data in a toy localization problem (i.e., in a toy HMM). This year, your only goal is to be sure you understand the forward algorithm, as understanding it will be essential to successful completion of the second part of this assignment.

4.1 Algorithm

To filter data in an HMM is to predict the true (hidden) state, given noisy observations. Filtering can be accomplished using the **forward algorithm**, which is a dynamic program captured by the following recurrence relation: for all $t \geq 0$,

$$\Pr(X_{t+1}|E_1, E_2, \ldots, E_{t+1}) = \alpha \Pr(E_{t+1}|X_{t+1}) \sum_{x_t} \Pr(X_{t+1}|X_t = x_t) \Pr(X_t|E_1, E_2, \ldots, E_t) \tag{3}$$

This recurrence relation follows directly from the HMM assumptions described by Equations 1 and 2. A derivation appears in Section 15.2.1 of the textbook.

4.2 Data Structure

In a localization problem, states are locations. A natural way to represent locations is on a plane, say the Cartesian plane, using two dimensions, x and y. A natural way to represent locations in a computer is to discretize a plane into a grid. Such a grid can be coded as a matrix, indexed by rows and columns. We call such a grid a **frame**, and code it in Python using a two-dimensional `numpy` array. Here is an example of a frame, showing the object of interest to be located towards the bottom right of a 4x4 grid:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

While this frame representation may be inefficient to represent locations that are known with certainty (since they involve only one 1 in a sea of 0s), it is less inefficient when the location is uncertain. In particular, we can also use a frame to represent a probability distribution over locations in a 4x4 grid, as follows:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0.2 \\ 0 & 0 & 0.1 & 0.5 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}$$

Observations in a localization problem are measurements taken at the various locations, so they are the output of the sensor model. In this assignment, observations are locations, so they are also represented as frames. They are certain locations (i.e., as in the first example, not the second); however, they are not always correct. For example, at the location above, the sensor model might report a nearby location, such as:

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

4.3 Library

We are providing you with a filtering library which you can use to filter noisy finger tracking data in part two of this assignment. The most important function in this library is (drum roll!) `filter`. This filter function works on a "generic" HMM, in which states are specified as integers $(1, \ldots, n)$, and so are observations $(1, \ldots, m)$. Given such an HMM (i.e., an initial probability distribution, a transition model, and a sensor model), together with a sequence of observations, `filter` returns a probability distribution over states.

As this filter function is generic, to use it to solve a specific problem (such as localization), you will need to encode the states in your problem as integers. After doing so, you will likewise encode the three probability distributions that specify an HMM; then, after running filter, you will have to decode the result (back from integers to states). The library also provides example `encode` and `decode` functions, which transform frames into integers and back again. Note, however, that when you build your HMM in part two of this assignment, you are free to choose your state representation; you need not encode a single state as a single frame.

5 Part 2: Finger Tracking

In the first part of the assignment, you studied filtering, as an approach to solving the localization problem. In the second part, you will use our filtering library to solve George's localization problem; that is, to track his finger on the touchscreen of his smartphone, from only noisy observations.[1] To do so will involve you building an HMM (i.e., transition and sensor models), and then letting our filtering algorithm do its job.

5.1 Domain Knowledge

To build an HMM, you must use domain knowledge. Here is what you know about the present domain:

- In a single timestep, George's finger will always either stay in the same place or move to one of the 8 adjacent locations (including diagonally adjacent locations) on the touchscreen.
- George's finger is likely to continue moving in the same direction it moved in the last timestep than it is to move in any other direction. But when it does not, it moves in a uniformly random direction.
- If George's finger is about to move off the edge of the touchscreen, he pauses for one timestep. Then, at the next timestep, it is more likely to move in the opposite direction.

In addition to this high-level description of George's finger movements, we have provided a simulator of finger movements,[2] with the following functionality:

- **Simulator** The simulator simulates finger movements and reports the noisy readings of the touchscreen. These reports are often correct, but err occasionally.

  For example, a simulation of three finger readings on a 4x4 touchscreen might look like this:

$$\left[\begin{bmatrix}0&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}, \begin{bmatrix}0&0&0&0\\0&0&1&0\\0&0&0&0\\0&0&0&0\end{bmatrix}, \begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&1&0\\0&0&0&0\end{bmatrix}\right]$$

- **Visualizer** The visualizer is debugging and testing aid. It can either display the actual position of the finger, the noisy reading, or both simultaneously.

[1] since he spilled his Dr. Pepper on his desk

[2] not actually George's

- **Testing Manager** The testing manager contains a few handy features to help you debug and test your solution. For example, you can save simulations so that you can later test using the same data multiple times. Additionally, the actual path of the finger is provided. Indeed, building on the simulation visualizer, the testing manager can display your filtered probabilities alongside this path.

To get started, we recommend that you run the simulator a few times, and visualize those simulations. You might look for patterns in the errors that the touchscreen makes. How would you describe these errors? How might you model these errors in your HMM?

N.B. The simulator only runs on department machines (unfortunately).

5.2 Your Tasks

Your main task is to formulate George's predicament as an HMM, `touchscreenHMM`, in `touchscreen.py`. In particular, you must code up: states, observations, an initial probability distribution over states, a transition model (probabilities from one state to another), and a sensor model (probabilities of seeing the various observations in the various states). We have provided you with stencils for `prior_model`, `transition_model`, and `sensor_model`. You may change them in any way you like (including their signatures).

You are also required to write the `filter_noisy_data` function. To assist you with writing this function, we have provided a filtering library (see Section 4.3). Your task will be to filter a sequence of m (at most, 50) observations. **Important:** Assuming a touchscreen of dimension 20x20, building your HMM should not take more than 60 seconds; likewise, filtering all m observations should not take more than 60 seconds.

Additional Notes

- **Representation:** A state need not correspond to a single location: i.e., a single frame. On the contrary, you may be able to trade off run time for accuracy by grouping adjacent location together into a "mega-location." Concretely, a mega-location for the finger could be the four locations in the top-left corner of a frame. There are conceivably other sensible state representations as well.

- **Filtering:** While filtering generally operates on a sequence of observations, the `filter_noisy_data` function expects only a single observation (frame). Consequently, you should cache your dynamic programming (DP) table, and then whenever `filter_noisy_data` is called, you can assume as the prior your current posterior. An example of this behavior is provided in the filtering library.

- **Robustness:** Note that the filtering algorithm relies on an application of Bayes' rule. In particular, the probability of a state given an observation is calculated as the probability of the observation given the state, times the probability of the state, divided by the probability of the observation. Consequently, the filtering algorithm divides by a quantity, which, if you are not careful, could be zero.

  There are (at least) two ways to make your code robust to this possible division by zero. The first is to make sure that you observation model assigns some small positive probability to *all* observations.

  The second is something you should do whenever you invoke the division operator: check for a potential division by zero error, and then throw and catch exceptions accordingly. (We are just reiterating an obvious point here: No code you write should ever terminate ungracefully!)

5.3 Simulator

- `simulator.so`

  This file contains the finger simulator as a compiled module (meaning you cannot see the source code).[3] In it is the class `touchscreenSimulator`. To create an instance of a touchscreen simulator, use:

```
touchscreenSimulator(width=20, height=20, frames=50)
```

[3]Since the simulator is a compiled module, it only runs on department machines.

Here, `width` and `height` are the dimensions of the touchscreen, and `frames` is the number of frames in the simulation. The default, which we expect to use for grading, is a 20x20 touchscreen over 50 frames.

After creating an instance of the simulator, you should call `sim.run_simulation()` before anything else. This will generate the data—a sequence of actual and noisy frames—for the simulation instance.

Then, to access the noisy frames, you can call `get_frame()` on an instance of `touchscreenSimulator`. (You can make function calls to this class as if it were a `.py` file.) The `get_frame()` function also increments and then returns the next timestep.

Additionally, by calling `get_frame(actual_position=True)`, you can access the actual position of the finger for testing purposes. This call returns a tuple consisting of (`noisy_frame`, `actual_frame`).

To visualize a simulation, use:

```
sim.visualize_simulation(noisy=True, actual=True, frame_length=.1)
```

By changing the values of `noisy` and `actual`, you can plot the noisy data, finger's actual location, or both. If you plot both, orange represents the actual location; yellow, the noisy position; and purple, where they overlap.

- `run_visual_simulation.py`: Contains a script that runs all the visualization code on a new simulation. This is useful when you want to run a bunch of simulations just to observe their behavior. You can call this function by running the following command in shell:

  ```
  python run_visual_simulation.py
  ```

- `run_touchscreen_tests.py`: Contains a script with different examples of how to use the Testing Manager in `simulation_testing_manager.py`. This is useful when saving simulations, and for visualizing your solution alongside the simulation.

  The file also contains documentation about how to use the testing functions and the various flags. It also contains examples. You can run the samples tests by running the following command in shell:

  ```
  python run_touchscreen_tests.py
  ```

  The text files created by the testing manager are saved in the following format (see `simple_test.txt`):

  ```
  20 20 50
  8 10 8 9
  8 10 8 10
  8 11 8 11
  ...
  ```

  The first line contains three integers referring to width, height, and the number of frames of the simulation. Each subsequent line represents one frame. The four integers: `noisy_x`, `noisy_y`, `actual_x`, `actual_y`. For example, 8 10 8 9 denotes a noisy location of (8, 10) and an actual location of (8, 9).

- `noisy_algorithm.so` The *top-secret* algorithm that we used to add noise to the actual position of the finger. You will have to use the visualizer to try to get a handle on this function. You should never have to call this function directly.

- `constants.py` This file contains the constants we use to seed the noise model (i.e., our top-secret algorithm). We will be testing your touchscreen with the given constants, but feel free to experiment to see what changing them does!

- `generate_data.py` You may find it useful to generate statistics (or do other calculations) comparing the noisy data to the actual finger locations. **(Hint Hint!)**

  To make it easier for you to process the simulation data, the function `create_simulations(size, frames)` returns a list of all the frames in a simulation.

- `visualizer.py` Contains the visualizer for this project. It automatically progress through the frames, using `matplotlib` to plot the `numpy` arrays. You won't ever have to call this function directly.

6 Part 3: Written Questions

In addition to your code, you should also submit a PDF file in which you answer the following questions. While not required, we recommend that you use LaTeX to typeset your work.

1. How did you model George's predicament as an HMM? Specifically:

   (a) What did you choose as the hidden states and observations?

   (b) How did you decide upon transition and sensor models? How did the observations generated by the simulator influence your choices? Account for each choice that you made.

   (c) What assumptions do you make in your approach? Do they hold in reality? In particular, does your mental model of the problem violate either (or both) of the two key HMMs assumptions?

   (d) What other approaches did you consider? Why did they seem promising at first? Why didn't you end up using them?

   Be sure to describe your ideas clearly and completely. Doing so may easily take more than a page.

7 Part 4: Ethics Questions

HMMs can be applied in many fields where the goal is to recover information that is not immediately observable. One common application is in speech recognition assistants. Read this brief article, and this one as well, both about voice recognition assistants, and then answer the following questions.

- Surveys demonstrate that Americans have a preference for female sounding voice assistants. By defaulting to a feminine-sounding voice for speech assistants, who is harmed? Who benefits?

- If you ask Alexa, the speech-assistant software in Amazon Echo devices, if it's a feminist, it will respond in the affirmative. "I am a feminist. As is anyone who believes in bridging the inequality between men and women in society," it continues. In contrast, previous Apple guidelines for Siri led it to disengage when given such questions, "My name is Siri, and I was designed by Apple in California. That's all I'm prepared to say." What assumption does each design make about the role of politics in technology and what are the implications of each design?

- Research shows that current voice assistants suffer from both racial and gender bias. Current assistants are most accurate on male voices, with significant discrepancies occurring for women and racial minorities or immigrant communities that speak with regional accents. Acknowledging that supporting every dialect in a region may be unfeasible, at what point should we deem it acceptable to deploy a voice recognition assistant?

8 Grading

This assignment is a modeling exercise. As such, there is no right answer.[4] So the question arises, how are we ever to grade you on this assignment? Related, and arguably even more important, how is anyone to tell if they have produced a good solution? Obvious answer: there must be some kind of scoring system.

It turns out that an analog of this problem has been studied by forecasting experts for decades, dating back to the work of Brier, a meteorologist, in the 1950s.[5] Brier was concerned that meteorologists did not always put all their efforts into producing the best forecast. Why? Well, he observed that it was often easier for a meteorologist to predict how they would be *scored* on a forecast than to try to predict the weather!

And so, the theory of proper scoring rules was born. A **proper scoring rule** is one for which you maximize your score by producing the best forecast you can. Consequently, all your efforts should go into just that: producing the best forecast you can. You cannot do better by reporting anything else.

We will grade you on this assignment using a proper scoring rule; not Brier's original rule (the so-called Brier score), but an alternative: the spherical scoring rule. So you can be sure that whenever you improve your HMM, meaning filtering produces more reliable predictions, your score can only increase. In particular, you should never find yourself in the uncomfortable situation where you believe you have improved your model, but your score has decreased. If this ever happens, by the theory of proper scoring rules, you can logically conclude that it is time to debug. That is the beauty of grading using a proper scoring rule!

There is one final detail missing from this grading plan. We have not yet told you how we plan to convert your spherical score, a number between 0 and 1, into a grade on this assignment. If we simply multiply each of your scores by 100, we may end up with a distribution of grades with a mean of 60! (Many of you may find that undesirable.) Another alternative is to first normalize your score, by dividing by the best of the TA's scores. In that case, if your score is 0.7, and the best TA score is 0.6, your grade would be 117. As this is the first year that we are using a proper scoring rule to score this assignment, we don't quite know yet what the range of spherical scores will be, or how we will convert your score into a grade. But you should all feel free to post your scores on Piazza to give other students in the class an idea of what is possible.

Practicalities: So that you can get an idea of how your model performs, we have provided for you the file **test_model.py**. Executing this code on a department machine will run n simulations over m (likely, 50) observations, and report the average proper spherical score across the n simulations, as well as the total time taken to both instantiate your model and filter the m observations. We note once again that while it will inform your grade for this assignment, this average spherical score is *not* your grade for this assignment.

9 Virtual Environment and Requirements

As usual, you should be using our virtual environment to run your code:

> To activate: `$ source /course/cs1410/venv/bin/activate`
>
> Then simply run your files: `$ python <executable.py>`
>
> To deactivate: `$ deactivate`

Alternatively, if you choose to work on your own machine, you will need some additional packages for this project: `scipy`, `matplotlib`, and `numpy`. These are already installed in our virtual environment, but you will need to install them on your own machine.

One quick way to install the dependencies you don't have is to use pip. If you haven't used pip before, you can find instructions here: https://pip.pypa.io/en/stable/installing/.

[4] Well, that's not entirely true. There is a simulator, and it is driven by a stochastic process; so in principle it might be possible to reverse engineer that process from examples, and earn a perfect score. But barring that possibility ...

[5] Here is a link to Brier's article in the January, 1950 issue of the Monthly Weather Review.

10 Install and Handin Instructions

To install, run `cs1410_install HMM` in `~/course/cs1410`.

To hand in, run `cs1410_handin HMM` in `~/course/cs1410/HMM`, which should contain your `hmm.py` and `touchscreen.py`.

In addition, please submit the written portion of the assignment via Gradescope.

In accordance with the course grading policy, your written homework should not contain your name, Banner ID, CS login, or any other personally identifiable information.