CS76 PA6, Kunal Jha, Fall 2021

Description

Filtering uses both a transition model and a sensor model. The transition model is to update the state, while the sensor model is used to update the probability distribution of likely states. The transition model is represented as a 16 by 16 matrix, and is multiplied by a flattened, 1 by 16 vector representing the probabilities of a location at a certain timestep to determine the location probabilities at the next timestep. Several matrix values are 0 because there is no possible way of arriving at that square (it is a wall). The sensor model represents the probability of being at a location in the maze given an observation by the sensor. Red readings have the chance of being at a square as 0.88/# of red squares. Any other color has a chance of being at that square as 0.04/# of that color's square. There are 4 matrices for any maze (one of each color).

At each timestep, the transition matrix is multiplied by the previous distribution and then further multiplied by the distribution generated by the sensor model to create a new probability distribution. The first timestep has a uniform distribution.

Smoothing was done using forward-backward algorithm. Forward-backward checking accounts for all of the sensor readings. The distribSmoothing function has two passes from the time 0 to the current time and from the current time back to time 0, with both passes starting with a uniform distribution. These passes are combined and form a smooth sequence.

For the actual methods, my computeDistrib method initialized the state and added it to the sequence. It then iterated through the sensorReadings, reshaped the state, and multiplied it by the transition matrix and color model. It then normalized the state and added it to the sequence again. Once all of the readings had been iterated through, the sequence was returned. DistribSmoothing was the forward-backward algorithm. It initialzed the forward and backward states, and executed a similar process as computeDistrib. The only difference was that distribSmoothing, iterated through the sensorReadings backwards as well. It then made a final iteration through the forward sequence, multiplied it by the corresponding timestep in the backwards sequence, normalized it, then added it to a new sequence. It then returned this new sequence. InitializeState assigned each square in a 4 by 4 matrix a uniform probability of 1/(number of sequence colors). Get moves iterated through all possible moves (left, right, up, down) and returned a list of the ones that were still on the floor. OneDimConvert multiplied the y-value of a square by the maze width and added the x value to that, and this function was used for building the transition matrix. Lastly, normalize divided all values in the state by the sum of all values in the state.

Evaluation

My program worked as expected, and in the testing phase it adjusted probabilities properly. Smoothing had more accurate predictions. Below is the output in the last few timesteps I had for *test_maze.maz* As you will see, smoothing assigns a higher probability to the square that the robot is actually in.

```
time 7 -----------
square color: y
sensor color: y

robot location:
#.#.
....
.#.A
.###

distribution:
[[0.         0.48835934 0.         0.00070441]
 [0.01224508 0.01579047 0.14853309 0.00830282]
 [0.00303387 0.         0.2142829  0.10547136]
 [0.00327667 0.         0.         0.        ]]

smoothed distribution:
[[0.00000000e+00 1.49169402e-01 0.00000000e+00 1.65288177e-04]
 [3.14594093e-03 7.69665561e-04 4.88324960e-01 2.59973439e-03]
 [7.77019800e-04 0.00000000e+00 2.10437846e-02 3.31005068e-01]
 [2.99913671e-03 0.00000000e+00 0.00000000e+00 0.00000000e+00]]
```

```
time 8 -----------
square color: g
sensor color: g

robot location:
#.#.
...A
.#..
.###

distribution:
[[0.         0.09127133 0.         0.00085597]
 [0.01067852 0.05464258 0.02384663 0.71325627]
 [0.00177418 0.         0.0420693  0.02671994]
 [0.03488527 0.         0.         0.        ]]

smoothed distribution:
[[0.00000000e+00 2.65848763e-03 0.00000000e+00 9.25316370e-05]
 [2.56525008e-03 2.31901933e-03 7.16070145e-04 9.42381627e-01]
 [1.37805914e-04 0.00000000e+00 1.51591561e-04 1.20352544e-04]
 [4.88572644e-02 0.00000000e+00 0.00000000e+00 0.00000000e+00]]

time 9 -----------
square color: r
sensor color: r

robot location:
#.#A
....
.#..
.###

distribution:
[[0.         0.01029965 0.         0.65843416]
 [0.00975524 0.16597266 0.02614654 0.04795718]
 [0.04517467 0.         0.00422405 0.02536105]
 [0.00667481 0.         0.         0.        ]]

smoothed distribution:
[[0.00000000e+00 4.00421246e-04 0.00000000e+00 7.50876237e-01]
 [1.51702432e-03 1.89274700e-01 1.01650320e-03 3.72887828e-03]
 [5.15170561e-02 0.00000000e+00 1.64218999e-04 9.85965564e-04]
 [5.18994997e-04 0.00000000e+00 0.00000000e+00 0.00000000e+00]]

---------Time Analysis---------
Total time for filtering only: 0.0002639293670654297
Total time for forward-backward smoothing: 0.0003979206085205078
```

On other 4x4 mazes, I had a similar output as well. This is to be expected since smoothing doesn't assign a uniform distribution to the starting state. Also, what you notice is that forward-backward smoothing runs slower than filtering. This makes sense as well, since forward-backward smoothing has to make two passes through the sensor readings and not just one like filtering. As such, if time is the major constraint, I would prefer standard filtering, however, if I wanted to maximize accuracy I would prefer forward-backward smoothing. As a whole, I would recommend using the forward-backward smoothing because the increased time is miniscule compared to the increase in prediction accuracy. For maze2, however, the performance time for smoothing was better than that of pure filtering.

```
time 6 -----------
square color: y
sensor color: y

robot location:
...A
....
....
....

distribution:
[[4.51616981e-03 1.23216864e-01 6.76691952e-03 1.91481269e-01]
```

```
 [2.35390594e-02 3.67585700e-03 2.84801781e-01 7.30412200e-03]
 [2.45983280e-03 1.27627201e-01 1.73459716e-04 1.93729007e-01]
 [5.94991631e-03 2.96721456e-03 1.76368246e-02 4.15450344e-03]]

smoothed distribution:
[[3.47180797e-04 1.30244167e-01 5.20207302e-04 2.02401826e-01]
 [9.04783227e-03 3.88549844e-03 3.01044593e-01 5.61504772e-04]
 [1.89099779e-04 1.34906034e-01 1.33347250e-05 2.04777758e-01]
 [1.14350034e-03 3.13644071e-03 3.38957960e-03 4.39144304e-03]]

---------Time Analysis---------
Total time for filtering only: 0.0007579326629638672
Total time for forward-backward smoothing: 0.0004277229309082031
```

This makes me recommend smoothing even more than before.