A6: HMM and First-Order Logic

Kevin King

Dartmouth COSC 76, Fall 2023

Implementation

This HMM (Hidden Markov Model) implementation is designed for a robot navigating a maze. The model is used to estimate the state of the robot over time, based on observations and a series of movements. The constructor instantiates the following parameters:

- `maze` : An instance of the `Maze` class that provides the layout of the problem
- `moves` : Sequence of moves (N, S, E, or W) that the robot will make within the maze
- `sensor_input` : Sequence of sensor readings corresponding to the colors observed by the robot at each step
- `start_state` : Probability distribution vector representing initial belief about robot's starting location in the maze, assumed to be uniform across all possible floor cells
- `state_indices` and `indices_states` : Dictionaries for mapping between maze coordinates and state indices
- `color_mapping` : Dictionary mapping colors to indices, used to interpret sensor readings
- `obs_matrix` : Observation probability matrix, reflecting probability of observing each color from each state
- `forward_probs` , `backward_probs` , `smoothed_probs` : Lists to store probabilities computed during forward pass, backward pass, and smoothing process, respectively.

Methods:

- `initialize_start_state` : Sets up the initial state distribution and creates two dictionaries for mapping between the indices of the state and their corresponding coordinates in the maze. The initial probabilities are uniform.
- `build_transition_matrix` : Creates the transition matrix for the HMM based on possible moves from each state (location) in the maze. Probabilities are assigned such that the robot has equal chances of moving to any of its accessible neighboring cells.
- `build_observation_matrix` : Sets up the observation matrix that contains the probabilities of observing each color from each state, taking into account the sensor's accuracy and potential for error.
- `filtering` : Implements the filtering process, updating the belief state based on the sequence of moves and sensor readings. It calculates the forward probabilities using the observation and transition matrices.
- `normalize` : A utility function that normalizes a probability distribution vector so that its elements sum up to 1.
- `print_results` : This method prints the results of the HMM operations. It shows the filtering and smoothing results at each time step, and if the smooth parameter is True, it also prints the smoothing results.
- `smoothing` (extension): This method combines information from both the forward pass (filtering) and a backward pass to refine the probability distributions of the robot's past states given all evidence up to the current time. See below for in-depth description
- `viterbi` (extension): Implements the Viterbi algorithm to find the most probable sequence of states given the sequence of observations. This algorithm uses dynamic programming to keep track of the best path so far. See

below for in-depth description

Additions to `Maze` Class

- `get_neighbors` : Obtains the neighboring coordinates of a given position, helping to define the transition probabilities, where transitions are only possible to neighboring cells
- `get_color` : Obtains the color at a given coordinate and is used to define the observation probabilities by relating the actual sensor readings to the expected readings in each cell
- `get_robot_location` and `update_robot_location` : Functions used to keep track and update the robot's state (location) as it moves through the maze

Testing

```
MAZE 1:
----------------------

Maze + Actual Robot (X) Location:
rgyr
ygrg
bybb
gXbg

TIME: 0
Filtering Results:
[[0.0625 0.0625 0.0625 0.0625]
 [0.0625 0.0625 0.0625 0.0625]
 [0.0625 0.0625 0.0625 0.0625]
 [0.0625 0.0625 0.0625 0.0625]]

Smoothing Results:
[[0.4279 0.2237 0.0012 0.0119]
 [0.005  0.0069 0.026  0.054 ]
 [0.0042 0.0031 0.003  0.0056]
 [0.0924 0.0881 0.0021 0.045 ]]
----------------------

Maze + Actual Robot (X) Location:
rgyr
ygrg
bybb
Xrbg

TIME: 1
Filtering Results:
[[0.2109 0.0064 0.0064 0.0096]
 [0.0096 0.0064 0.1406 0.2109]
 [0.0096 0.0064 0.0064 0.0096]
 [0.2109 0.0064 0.1406 0.0096]]

Smoothing Results:
[[6.415e-01 4.000e-04 1.980e-02 3.000e-03]
 [5.000e-03 3.200e-03 1.040e-02 7.330e-02]
 [4.900e-03 1.800e-03 2.900e-03 6.200e-03]
 [1.375e-01 1.500e-03 8.450e-02 4.000e-03]]
----------------------

Maze + Actual Robot (X) Location:
```

```
rgyr
ygrg
Xybb
grbg

TIME: 2
Filtering Results:
[[0.0276 0.3085 0.001  0.0016]
 [0.0016 0.0097 0.014  0.0363]
 [0.0363 0.0227 0.001  0.0016]
 [0.0276 0.499  0.001  0.0103]]

Smoothing Results:
[[1.760e-02 6.430e-01 4.000e-04 3.700e-03]
 [6.400e-03 3.500e-03 5.340e-02 2.860e-02]
 [4.100e-03 3.100e-03 2.000e-03 6.600e-03]
 [4.140e-02 1.612e-01 2.100e-03 2.280e-02]]
----------------------

Maze + Actual Robot (X) Location:
rgyr
Xgrg
bybb
grbg

TIME: 3
Filtering Results:
[[4.010e-02 3.200e-03 7.517e-01 5.000e-04]
 [1.400e-03 3.800e-02 5.100e-03 9.500e-03]
 [1.050e-02 4.100e-03 2.700e-03 1.050e-02]
 [6.110e-02 3.200e-03 5.610e-02 2.400e-03]]

Smoothing Results:
[[8.700e-02 4.600e-03 5.723e-01 9.000e-04]
 [2.600e-03 5.480e-02 7.300e-03 2.720e-02]
 [4.900e-03 3.000e-04 2.000e-03 8.500e-03]
 [1.326e-01 2.930e-02 4.270e-02 2.300e-02]]
----------------------

Maze + Actual Robot (X) Location:
rgyr
yXrg
bybb
grbg

TIME: 4
Filtering Results:
[[0.2252 0.0973 0.0004 0.0925]
 [0.005  0.0008 0.1285 0.0653]
 [0.0031 0.0016 0.0018 0.0029]
 [0.3387 0.0144 0.015  0.0075]]

Smoothing Results:
[[8.910e-02 2.570e-02 1.400e-03 5.487e-01]
 [2.980e-02 2.400e-03 3.390e-02 2.580e-02]
 [1.200e-03 4.900e-03 5.000e-04 9.200e-03]
 [1.340e-01 3.800e-03 4.540e-02 4.440e-02]]
----------------------

Maze + Actual Robot (X) Location:
rgyr
```

```
ygXg
bybb
grbg

TIME: 5
Filtering Results:
[[0.0722 0.0297 0.025  0.5378]
 [0.0314 0.0176 0.0087 0.0342]
 [0.001  0.0006 0.0131 0.001 ]
 [0.0912 0.0466 0.0029 0.0869]]

Smoothing Results:
[[0.0722 0.0297 0.025  0.5378]
 [0.0314 0.0176 0.0087 0.0342]
 [0.001  0.0006 0.0131 0.001 ]
 [0.0912 0.0466 0.0029 0.0869]]

Observed Colors: ['g', 'b', 'y', 'g', 'r']
Viterbi Best Path for Maze 1: [(0, 0), (0, 0), (0, 1), (0, 2), (0, 3), (0, 3)]
-------------

MAZE 2:
----------------------

Maze + Actual Robot (X) Location:
gbyr
#gyb
brgg
Xybr

TIME: 0
Filtering Results:
[[0.0667 0.0667 0.     0.0667]
 [0.0667 0.0667 0.0667 0.0667]
 [0.0667 0.0667 0.0667 0.0667]
 [0.0667 0.0667 0.0667 0.0667]]

Smoothing Results:
[[0.0257 0.0257 0.     0.0049]
 [0.0151 0.0153 0.0184 0.0446]
 [0.036  0.0177 0.0011 0.0007]
 [0.3001 0.2533 0.0771 0.1642]]
----------------------

Maze + Actual Robot (X) Location:
gbyr
#gyb
Xrgg
bybr

TIME: 1
Filtering Results:
[[0.1976 0.1976 0.     0.012 ]
 [0.009  0.006  0.006  0.1976]
 [0.1976 0.006  0.006  0.009 ]
 [0.009  0.006  0.1317 0.009 ]]

Smoothing Results:
[[2.570e-02 2.570e-02 0.000e+00 4.900e-03]
 [2.250e-02 2.000e-04 1.570e-02 5.510e-02]
 [5.300e-02 1.400e-03 1.000e-04 1.000e-03]
```

```
 [3.568e-01 1.245e-01 2.687e-01 4.470e-02]]
----------------------

Maze + Actual Robot (X) Location:
gbyr
#gyb
bXgg
bybr

TIME: 2
Filtering Results:
[[0.087  0.087  0.     0.007 ]
 [0.0035 0.0484 0.0299 0.0589]
 [0.0589 0.0299 0.0022 0.0035]
 [0.0774 0.0207 0.0022 0.4835]]

Smoothing Results:
[[2.570e-02 2.570e-02 0.000e+00 4.900e-03]
 [3.000e-04 3.720e-02 3.000e-03 5.300e-02]
 [5.170e-02 2.100e-03 1.600e-03 0.000e+00]
 [4.756e-01 1.900e-03 1.390e-02 3.033e-01]]
----------------------

Maze + Actual Robot (X) Location:
gbyr
#gyb
brXg
bybr

TIME: 3
Filtering Results:
[[0.0301 0.0301 0.     0.0714]
 [0.0064 0.0039 0.2721 0.017 ]
 [0.017  0.155  0.0039 0.0011]
 [0.0202 0.2019 0.0581 0.1118]]

Smoothing Results:
[[2.570e-02 2.570e-02 0.000e+00 4.900e-03]
 [1.400e-03 8.000e-04 7.990e-02 1.140e-02]
 [9.900e-03 4.550e-02 1.000e-04 0.000e+00]
 [9.900e-03 4.801e-01 1.710e-02 2.876e-01]]
----------------------

Maze + Actual Robot (X) Location:
gbyr
#gyb
brgg
byXr

TIME: 4
Filtering Results:
[[0.0858 0.0858 0.     0.0123]
 [0.0007 0.012  0.0009 0.2909]
 [0.1796 0.0009 0.0067 0.0003]
 [0.0105 0.0034 0.298  0.0122]]

Smoothing Results:
[[2.570e-02 2.570e-02 0.000e+00 4.900e-03]
 [1.700e-03 2.400e-03 2.100e-03 8.720e-02]
 [5.390e-02 2.000e-04 1.300e-03 1.000e-04]
 [4.710e-02 7.800e-03 6.851e-01 5.470e-02]]
```

```
----------------------

Maze + Actual Robot (X) Location:
gbyr
#gyb
brgg
bybX

TIME: 5
Filtering Results:
[[2.570e-02 2.570e-02 0.000e+00 4.900e-03]
 [1.300e-03 3.600e-03 3.030e-02 5.820e-02]
 [3.600e-02 1.860e-02 1.000e-04 7.000e-04]
 [5.350e-02 3.080e-02 1.600e-03 7.088e-01]]

Smoothing Results:
[[2.570e-02 2.570e-02 0.000e+00 4.900e-03]
 [1.300e-03 3.600e-03 3.030e-02 5.820e-02]
 [3.600e-02 1.860e-02 1.000e-04 7.000e-04]
 [5.350e-02 3.080e-02 1.600e-03 7.088e-01]]

Observed Colors: ['b', 'r', 'g', 'b', 'r']
Viterbi Best Path for Maze 2: [(3, 0), (3, 0), (3, 0), (3, 1), (3, 2), (3, 3)]
-------------

MAZE 3:
----------------------

Maze + Actual Robot (X) Location:
ybrg
bg#b
Xybg
#gry

TIME: 0
Filtering Results:
[[0.     0.0714 0.0714 0.0714]
 [0.0714 0.0714 0.0714 0.0714]
 [0.0714 0.0714 0.     0.0714]
 [0.0714 0.0714 0.0714 0.0714]]

Smoothing Results:
[[0.000e+00 1.000e-04 2.500e-03 5.000e-03]
 [2.738e-01 1.130e-02 2.571e-01 1.170e-02]
 [1.000e-04 1.000e-04 0.000e+00 0.000e+00]
 [2.774e-01 1.453e-01 3.300e-03 1.230e-02]]
----------------------

Maze + Actual Robot (X) Location:
ybrg
bg#b
rXbg
#gry

TIME: 1
Filtering Results:
[[0.     0.0146 0.0097 0.3204]
 [0.0146 0.2136 0.0097 0.0146]
 [0.0146 0.0146 0.     0.0194]
 [0.3204 0.0097 0.0097 0.0146]]
```

```
Smoothing Results:
[[0.000e+00 1.000e-04 1.000e-04 7.400e-03]
 [3.380e-02 5.026e-01 1.000e-04 1.750e-02]
 [1.000e-04 1.000e-04 0.000e+00 0.000e+00]
 [4.154e-01 9.000e-04 1.360e-02 8.400e-03]]
----------------------

Maze + Actual Robot (X) Location:
ybrg
bg#b
rybg
#Xry

TIME: 2
Filtering Results:
[[0.     0.0019 0.0164 0.0319]
 [0.2617 0.0012 0.246  0.0019]
 [0.0021 0.0021 0.     0.0038]
 [0.0319 0.3559 0.0012 0.0419]]

Smoothing Results:
[[0.000e+00 0.000e+00 1.600e-03 6.000e-03]
 [2.819e-01 0.000e+00 2.718e-01 2.000e-04]
 [1.000e-04 1.000e-04 0.000e+00 0.000e+00]
 [6.020e-02 3.669e-01 1.000e-04 1.110e-02]]
----------------------

Maze + Actual Robot (X) Location:
ybrg
bg#b
rXbg
#gry

TIME: 3
Filtering Results:
[[0.000e+00 9.000e-04 1.400e-03 7.490e-02]
 [2.230e-02 4.745e-01 1.000e-04 1.060e-02]
 [3.000e-04 3.000e-04 0.000e+00 6.000e-04]
 [3.922e-01 1.400e-03 1.690e-02 3.600e-03]]

Smoothing Results:
[[0.000e+00 1.000e-04 1.000e-04 7.400e-03]
 [5.900e-03 5.356e-01 0.000e+00 1.250e-02]
 [1.000e-04 1.000e-04 0.000e+00 0.000e+00]
 [3.963e-01 0.000e+00 3.450e-02 7.500e-03]]
----------------------

Maze + Actual Robot (X) Location:
ybrg
bX#b
rybg
#gry

TIME: 4
Filtering Results:
[[0.000e+00 1.000e-04 2.300e-03 4.600e-03]
 [3.493e-01 7.000e-04 3.264e-01 7.000e-04]
 [0.000e+00 0.000e+00 0.000e+00 1.000e-04]
 [2.400e-02 2.753e-01 2.000e-04 1.620e-02]]

Smoothing Results:
```

```
[[0.000e+00 1.000e-04 3.000e-04 7.200e-03]
 [6.760e-02 1.000e-04 4.844e-01 1.900e-03]
 [0.000e+00 1.000e-04 0.000e+00 0.000e+00]
 [4.700e-03 4.085e-01 0.000e+00 2.510e-02]]
----------------------

Maze + Actual Robot (X) Location:
ybrg
Xg#b
rybg
#gry

TIME: 5
Filtering Results:
[[0.000e+00 2.000e-04 6.700e-03 7.000e-04]
 [4.510e-02 4.360e-02 1.000e-04 4.652e-01]
 [0.000e+00 1.000e-04 0.000e+00 0.000e+00]
 [2.090e-02 1.600e-03 4.138e-01 2.100e-03]]

Smoothing Results:
[[0.000e+00 2.000e-04 6.700e-03 7.000e-04]
 [4.510e-02 4.360e-02 1.000e-04 4.652e-01]
 [0.000e+00 1.000e-04 0.000e+00 0.000e+00]
 [2.090e-02 1.600e-03 4.138e-01 2.100e-03]]

Observed Colors: ['y', 'g', 'y', 'g', 'b']
Viterbi Best Path for Maze 3: [(3, 0), (3, 0), (3, 1), (3, 0), (3, 1), (3, 2)]
```

At Time: 0, both filtering and smoothing start with uniform distributions since there is no prior information about the robot's location. As time progresses, the smoothing probabilities tend to have higher values at the true location of the robot, showing that it utilizes the future observations effectively.

At the last time step (Time: 5), the smoothing results are identical to the filtering results because there are no future observations to consider; thus, smoothing has no additional information to incorporate.

The observed colors sequence and the Viterbi best path indicate how the robot is moving through the maze and how it corresponds to the actual locations of the robot marked by 'X'.

The forward-backward smoothing results are generally more refined than the raw filtering results, showing higher probabilities at the robot's actual location. The Viterbi path provides a definite trajectory of the robot's movement, which can be cross-referenced with the actual path taken. The discrepancies between the robot's actual location and the high probability locations in the filtering results highlight the imperfect nature of sensor observations and the model's attempt to reconcile these uncertainties.

Extension

1. Forward-Backward Smoothing

- Iterates through the sensor inputs in reverse order.
- For each sensor reading, it gets the corresponding observation probability vector from the `obs_matrix` using the `color_mapping` dictionary that translates sensor inputs to indices.
- Calculates the backward probabilities (`backward_probs`) by matrix multiplying the transpose of `trans_matrix` with the element-wise product of the observation vector and the last backward probability in the list

- Normalizes the backward probabilities using the `normalize` function.
- These probabilities are then appended to the `backward_probs` list.

Reversing the Backward Probabilities:

- After the backward pass, the `backward_probs` list is reversed to align it with the forward probabilities.

Combining Forward and Backward Probabilities:

- The method then iterates over pairs of forward and backward probabilities.
- It combines them by taking the element-wise product of the two, yielding the smoothed probabilities for each state at each time step.
- These are normalized and appended to the `smoothed_probs` list.
- The normalize function ensures that the probability distributions sum up to 1 by dividing each element of the vector by the total sum of the vector, provided that the total is greater than zero.

2. Viterbi Algorithm

- Sets up a Viterbi matrix `V` and a path matrix path with dimensions `num_states` x T+1, where `T` is the length of the observed sequence.
- Initializes the first column of `V` with the `start_state` probabilities, representing the initial distribution over the states.

Body:

- For each time step t from 1 to T, the algorithm iterates over each possible state `s`.
- For each state, it calculates the product of the observation probability at time t (from `obs_matrix`), the transition probabilities from all states to state s (from trans_matrix), and the probabilities from the previous time step in `V`.
- Determines the maximum of these products (`max_trans_prob`), which represents the most likely probability of ending up in state `s` at time `t`.
- The algorithm updates the Viterbi matrix `V` with these maxima and records the path taken to reach these maxima in the path matrix.

Backtracking:

- After filling in the Viterbi matrix, the algorithm finds the most probable last state (`last_state`) by selecting the state with the highest probability in the last column of `V`. It then constructs the most probable path (`best_path`) by backtracking through the path matrix, from the last timestep to the first, following the indices of the maximum probabilities.
- The backtracked path is reversed to get the correct order from the first to the last timestep.
- The algorithm converts the state indices in `best_path` to coordinates using `indices_states` that maps state indices to their coordinates.

Extension - Summary of Article: "Extrapolation Methods for Accelerating PageRank Computations"

The research presents a significant enhancement to the PageRank algorithm, crucial for ranking web pages in search engine results. By introducing the Quadratic Extrapolation method, the study addresses the inefficiencies of the traditional Power Method, which relies on computing the principal eigenvector of the web's hyperlink matrix. This new

approach, using the Aitken and Epsilon Extrapolation techniques, assumes an iterate can be represented as a linear combination of the matrix's first two eigenvectors, enabling the subtraction of nonprincipal eigenvector estimates to accelerate convergence. Specifically, Quadratic Extrapolation utilizes the first three eigenvectors for approximation, significantly improving convergence speed—up to threefold in tests — especially when the damping factor is near 1. The methods were tested using the STANFORD.EDU and LARGEWEB datasets, demonstrating a 25-300% efficiency increase and a 69% reduction in computation time in some cases, making them valuable for large-scale web graphs where existing algorithms tend to under-perform. Furthermore, these techniques introduce minimal computational overhead, require no major algorithmic infrastructure changes, and are general-purpose - suitable for any large, sparse Markov matrix.