Minh Tran – HW7: Hidden Markov Model

1. Part 1: Implementation:

a. Libraries:

Pycharm IDE is used with Python 3.6 as the programming language. Basic libraries such as math (for square root operation), collections (for dictionary usage) are used to facilitate the program.

b. HMM details:

✓ Here the hidden markov chain is the actual positions of the robot at each of the 11 timesteps. At each position, the robot measures its distance to the four corners and emit a noisy signal to record its location.

✓ My task will be to reveal the most likely path of

✓ To do that, several steps are taken:

- Extract input text file to get the tower location, the free positions in the grid and the recorded distance at each time step
- Establish the actual distance of each free position to the four towers, introduce noise to constrain the recorded distance of the robot.
- Based on the recorded distance, I find out at each time steps, what are the probable positions of the robot. Then with a simple back calculation, I find out at each of those positions, what are the probable time steps corresponding to that position.
- Calculate transitional probabilities between 2 probable positions with a constraint that those 2 positions have to be physical neighbors (so that the robot can actually move between the two) and they belong to probable states of two consecutive timesteps (based on the recorded position of the robot).
- Perform Viterbi algorithm to find out which chain of positions has the highest probability of happening to emit the recorded distance.

c. Data structure:

✓ The given txt files are read line by line to obtain:

- Free positions' locations: list of lists: [[i,j],...]
- Tower locations: list of lists: [[i,j],...]
- Recorded distance: list of lists: [[d1,d2,d3,d4],…]

✓ Then the following are evaluated:

- For each free pos, calculate distances to each tower: list of lists: [[[$0.7d_1$, $1.3d_1$],[$0.7d_2$, $1.3d_2$],[$0.7d_3$, $1.3d_3$],],[$0.7d_4$, $1.3d_4$]],…]

- Find probable states of each time step: dictionary: {timestep0: [[i1,j1], …],timestep1:…}
- Find probable timesteps each pos might belong: dictionary: {[i1,j1]: [timestep1, ...],...
- Find neighbors: dictionary {(3, 4): [(4, 4), (3, 5), (2, 4), (3, 3)],…}
- Find transitional probability: dictionary {(4, 3): {(5, 3): 1.0}, (4, 4): {(5, 4): 0.5, (4, 3): 0.5},…}

d. Files included:

✓ hmm-data.txt: storing the input data

✓ hmm_MT.py - Contains the implementation of HMM from scratch

e. Code-level optimization:

✓ Dictionary is used extensively to update the transitional probability using key properties

✓ Viterbi algorithm is used to keep the problem tractable.

f. Challenges:

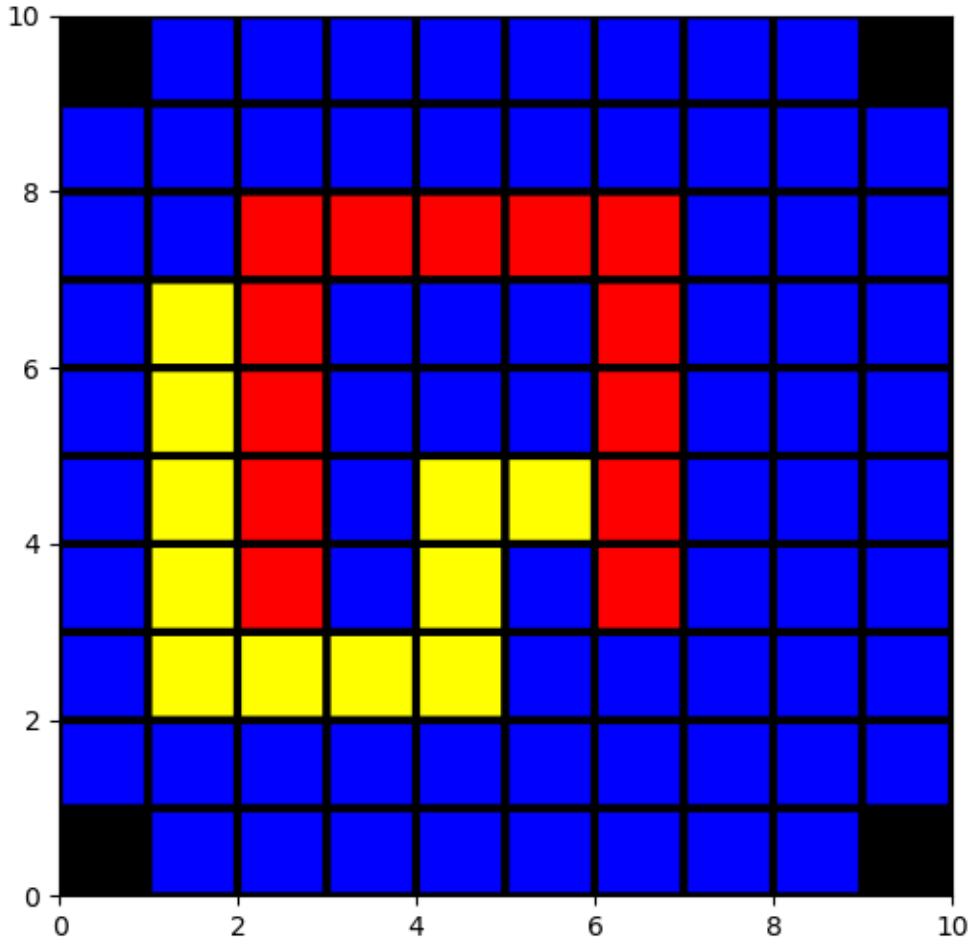
✓ The algorithm is currently fast but scaling up the grid or expand the upper and lower margins of the recorded distance may slow down the script.

g. Results:

✓ The final path showing the most probable position of the robot is:

[(5, 5), (5, 4), (6, 4), (7, 4), (7, 3), (7, 2), (7, 1), (6, 1), (5, 1), (4, 1), (3, 1)]

Below is the illustration: blue cells are free cells to move, red cells are obstacles, black cells are location of towers and yellow cell is the path.


10
8
6
4
2
0
0
2
4
6
8
10

2. Part 2: Software familiarization:

a. Sklearn for HMM (hmmlearn):

Features	My Algorithm	HMMLearn
API	No API provided	Emulates scikit-learn API but adapted to sequence data.
Data type	Only deal with the given matrix and can only work with one sequence	Deal with different types of emission data including Gaussian, Gaussian Mixture and Multinomial. Work with multiple sequences
Algorithm	Viterbi Algorithm	3 types: Viterbi, Forward-Backward algorithm and Baum-Welch algorithm.
Plot	No Plotting provided	Built with matplotlib to visualize the emission if needed
Coding	Use multiple function in the script	Build a base class called hmmlearn.base which facilitates easy evaluation, convenient sampling, and MAP estimation of HMM parameters

3. Applications of HMM

- ✓ HMM has applications in several industries: medical (DNA sequencing, bioinformatics), logistics (queuing network design), linguistics (stochastic language modeling, speech recognition), natural language processing (pos tagging)
- ✓ Stochastic language modeling: It was shown that alphabet level Markov chains of order 5 and 6 can be used for recognition texts in different languages. The word level Markov chains of order 1 and 2 are used for language modeling for speech recognition HMM models are widely used in speech recognition, for translating a time series of spoken words into text.