INF 552 Programming Assignment 7

Hidden Markov Model

4472-6221-33 Yo Shuan Liu

Introduction

I complete this homework in Python3. My code can be executed from terminal, and there is no absolute path for input files. User only needs to follow below instructions to execute this script:

1) Put the input .txt file (hmm-data.txt) and this code **under the same directory**.
2) Change directory to this folder and execute below line in terminal:
   python HMM.py
3) After printing out all the output, a message will show up in the terminal and ask user to **press [enter]** in order to end the execution and close all the output graph windows.

In this report, I will introduce my implementations of the HMM Viterbi algorithm to output the coordinates of the most likely trajectory of the robot in a grid world. I will cover following sessions:

I. Data Structure
II. Implementation, Result, Challenges I Face and Optimization
III. Software Familiarization
IV. Applications

I. Data Structure

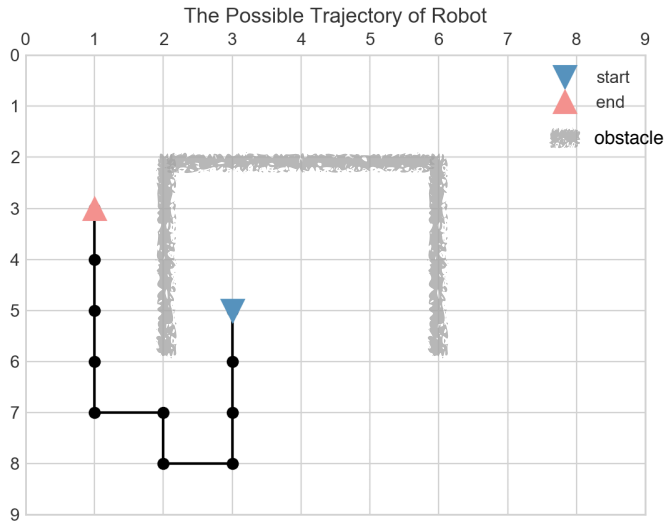
I store all the input information in a dictionary, with keys: grid_world, tower_loc and footprint. Using dictionary allows me to read input file more intelligent (without specifying the actual number of line to read for three different information).

Since the most likely trajectory of the robot is obtained by retracing from the last step, I need to store the optimal path for every step. Instead of storing the whole path from the first step, I store only the previous position for each step, and call it as the “back-pointer”. In other words, for each step, I use a 2D numpy array to represent the grid world, and for each grid, I store from which grid the robot came from from previous step, that is, which neighboring grid along with this grid forms the highest probability to produce provided observation.

II. Implementation, Result, Challenges I Face and Optimization

Result

The most possible trajectory of Robot for 11 time-steps is as follow:
[(5, 3), (6, 3), (7, 3), (8, 3), (8, 2), (7, 2), (7, 1), (6, 1), (5, 1), (4, 1), (3, 1)]



Implementation
Hidden state: the real position of the robot in the grid world
Observation: noisy distances to towers 1, 2, 3 and 4 respectively for 11 time-steps

I calculate the probability of the robot stepping on each grid base on the probability of its free neighboring grid's on the previous step, the count of free neighbors of this neighboring grid, and the current step's input distance record. Since each grid has four neighbors, I choose the one which produce the largest probability and set a back-pointer pointing back to this neighboring grid. After 11 time-steps, I find out the grid with the highest probability for the last step, and mark it as the final grid the robot step on. And then backtrack the robot's trajectory base on the back-pointer I stored for each step.

Challenges I Face: the space complexity
Beside the input data, there are some data generated through the process. At first, I store the probability for each grid in each step. This forms a 11x10x10 matrix after all calculations. I also store the "back-pointer" for each grid in each step. This forms another 11x10x10 matrix after all calculations. Although these doesn't exceed my computer's capacity, I believe I could further improve the performance of my implementation by reducing the space complexity.

Code Level Optimization
I found out that storing the probability for each grid in each step is unnecessary. After all, we only need the probabilities of standing on each grid from the last step (11th time-step), and find out which grid in the last step has the largest probability. Therefore, I store only the probabilities of each grid for current and previous step only.

Moreover, I write my code in an object oriented manner. I also put parameters such as the noise interval coefficient at the very front of my class to avoid any magic numbers in the code.

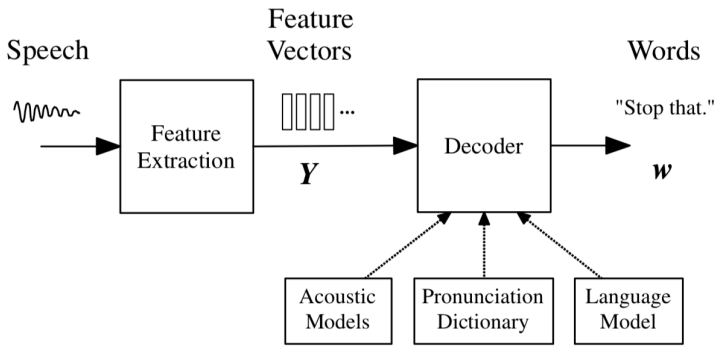
III. Software Familiarization

from hmm learn import **hmm**
This is not a library from sklearn, and before we use it, we have to install it via pip install --upgrade --user hmmlearn. It takes initial population probability, the transition matrix, and the means and covariance of each component as input. We can train an HMM by calling the fit method. The inferred optimal hidden states can be obtained by calling predict method. Currently the Viterbi algorithm, and maximum a posteriori estimation are supported.

Reference:
https://github.com/hmmlearn/hmmlearn
https://hmmlearn.readthedocs.io/en/latest/tutorial.html#

IV. Applications

Hidden Markov Model can be applied to speech recognition. We give machine acoustic data as input and we want to output a word sequence with highest probability, and HMM provides a natural framework for



constructing such models. Both discrete density HMMs and continuous density HMMs can be used as acoustic model. Forward–backward algorithm is commonly used, which is an example of expectation maximization (EM).

Automatic continuous speech recognition (CSR) has many potential applications including command and control, dictation, transcription of recorded speech, searching audio documents and interactive spoken dialogues. And HMM helps to construct the statistical models behind CSR.

Reference:
https://mi.eng.cam.ac.uk/~mjfg/mjfg_NOW.pdf