Assignment 7 Group Report - HMM-Viterbi

Group Members:
Che-Pai Kung 5999-9612-95
Chenqi Liu 2082-6026-02
Mengyu Zhang 3364-2309-80

Part 1: Implementation

Data structure

Three essential elements are needed to run the Viterbi algorithm. These are the initial probability of each state, transition matrix and emission probability.

First, We transformed the coordinates value (i, j) of each cell into a single value by the formula: $10 \times i + j$ so that it becomes easier to fetch the value from those three matrices. For example, coordinates (0,1) is converted into 1 and (2,1) is converted into 21. There are 100 cells in total, meaning 100 hidden states.

As the robot starts at any free cell equally, the initial probability is 1/87 (87 free cells) for free cells and 0 for obstacles. The "initial_prob" is the matrix whose shape is 100 times 1.

At each free cell, the robot moves to its adjoining free cells uniformly. We assume that the neighboring cells include only vertical and horizontal cells. Initially, the matrix "transition_matrix" is created as the numpy array whose dimension is 100 times 100. If cell i is a neighbor of cell j, then transition_matrix[i][j] and transition_matrix[j][i] are set equal to one. Afterwards, the probability is calculated by the value of each cell dividing the sum of its corresponding row.

Lastly, the observations are the distances from the current cell to the four towers. As the true distance (d) is needed to recognize if the observations are within [0.7d, 1.3d], instead of storing the emission probability, we store the distances in the matrix "distance" and later compute the probability when needed. The dimension of the matrix "distance" is 100 times 4.

Once three matrices are created, the Viterbi algorithm can be executed. Several variables are created to store the results of each timestamp. "state_t" is a dictionary storing the route with maximum log probability of its key value at timestamp t and "state_t_1" is the same structure but at timestamp t-1. The maximum log probability mentioned above is saved in the numpy array "logprob_t" and "logprob_t_1". "possible_state_t" and "possible_state_t_1" are built to record the possible state at time t and t-1. The hidden states are selected if all the distances to four towers are within the specified range.

Code-level optimizations

The program was organized into a more structured order. Three main matrices were moved to the top of the code and the function "viterbi" was created to increase readiness and to avoid many variables being created after executing the program. In addition, the figure plotting the most likely route is created to check whether the route is reasonable or not.

We use the array/list "possible_state_t" and "possible_state_t_1" to store the possible states at time t and time t-1. In this case, we do not need to calculate the

probability of each hidden state and then derive the answer containing many '-inf'. Moreover, the calculation stops once there is no connection between two states. Therefore, redundant computations are reduced.

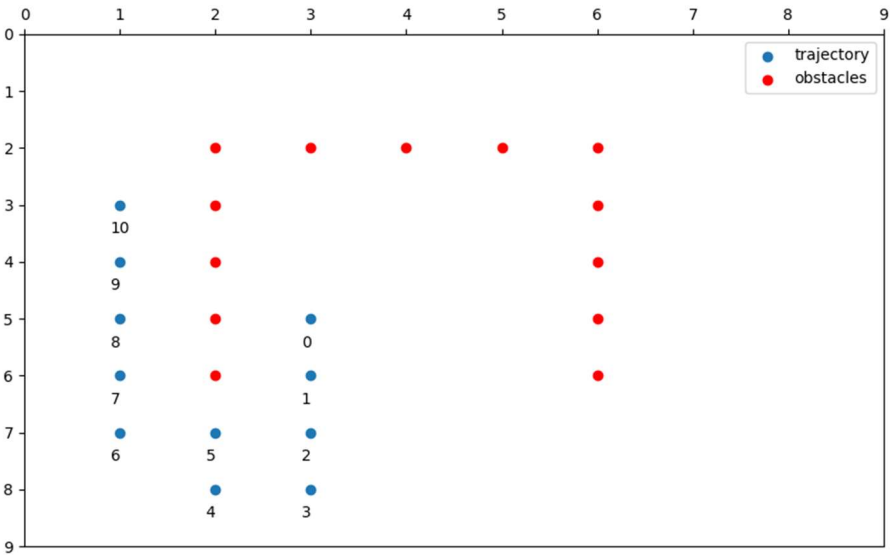
Challenge

At first, it was hard to understand the connection between the problem, variable-elimination algorithm and Hidden Markov Model. We figured out the concepts of the Viterbi algorithm after searching more information about it. Viterbi algorithm is implemented by recording the route with the maximum probability of each hidden state at each timestamp. With such technique, only the results in time t-1 are needed to calculate outcome in time t. This also matches the concept of variable-elimination algorithm as HMM's variable-interaction (VI) graph becomes smaller after a timestamp is finished computing.

Another difficulty was about memorizing the results at each time-step. It would be helpful if only the possible states and their routes with maximum probability are stored so it is not required to iterate all the hidden states at the next time-step. Then, we figured out that we can use the dictionary and the list to record only the possible states and their routes.

Results

The most likely trajectory of the robot for 11 time-steps:
[(5, 3), (6, 3), (7, 3), (8, 3), (8, 2), (7, 2), (7, 1), (6, 1), (5, 1), (4, 1), (3, 1)]



Part 2: Software Familiarization

Implementation

The library that is relevant to Hidden Markov Models is “hmmlearn”.

The library offers three base models with different kinds of emission probability, Gaussian, Gaussian Mixture and multinomial (discrete) emission. In this assignment, the emission probability is calculated uniformly according to the distance between four towers, which seems not applicable to either of three models. Then, we came up with the way to transform the problem into discrete emission. Suppose that there are only 11 possible observations, which are the observation provided, then we computed the probability of each hidden state (position) to generate those observations. The “emissionprob” is the matrix to save the above results and its dimension is 100 times 11. Afterwards, we can apply “hmm.MultinomialHMM” on the assignment using the same “transition_matrix” and “initial_prob” as we used in our implementation and the “emissionprob” we created specifically for this group of observation.

The model requires to be fitted before decoding the most likely hidden state sequence. Therefore, it is necessary that the matrices being fixed, thus the parameter “params” is set to ‘’, meaning no updates on the matrices in the training process. Simultaneously, the matrices are assigned to its correspondent parameters. Though the error appears when we assigned the initial probability to parameter “startprob_”, the start probability is uniform among all the states, which is 0.01, and it does not affect the answer as the robot cannot move to the next cell if it chooses the obstacle cell at first. The trajectory generated by “hmm.MultinomialHMM” is the same as our implementation.

Comparison and Inspiration

Other than decoding the most likely state sequence, the “hmmlearn” library also helps finding the parameters for the model given the observations. For example, instead of giving the obstacle cells, we are provided with multiple observations of robot routes. We can train the Hidden Markov model based on those observations and we might be able to find where the obstacle cells are based on the value of parameters (start probability, transition matrix and emission matrix). In addition, we found that the training process utilizes an EM algorithm, which is related to one of the previous assignments. After combining two assignments, we might be able to implement the model that can generate a HMM based on visible results.

Part 3: Applications

Since the Hidden Markov Model(HMM) did a great job when there is an unknown parameter involved in the data with its ability of processing sequential data with timestamps, it becomes a solid technique in the trend of popularizing Machine Learning techniques. HMM has done an excellent job in processing speech recognition.

Noticed that there is similarity between speech signals and earthquake signals, Beyreuther and his team apply HMM in the task of detecting earthquakes. In the article *Constructing a Hidden Markov Model based earthquake detector: application to induced seismicity* they published in 2012, Beyreuther and his team did a

comprehensive research on borrowing HMM in studying the earthquake signals. As a result, HMM provided a promising and valuable alternative technique for certain cases of earthquake signals.
Because of the great job HMM did in pattern recognition, Monaco and Tappert did a research in 2018 on applying HMM in studying the typist's behavior by the keystroke dynamic. In the article *The Partially Observable Hidden Markov Model and its Application to Keystroke Dynamics* they published, they concluded that using POHMM, an extension of HMM, applied to keystroke dynamics demonstrates superiority over leading alternative models on a variety of tasks, including identification, verification, and continuous verification.
Also, considering the characteristics of HMM, Harrison and his team developed a storage workload model from HMM to the application of flash memory. In the paper *Storage workload modelling by hidden Markov models: Application to Flash memory* they published in 2011, they stated that the model they developed could have better performance because correlation within and between the read and write streams is perhaps more significant than randomness.

Part 4: Individual Contributions

- Model discussion: Che-Pai Kung, Chenqi Liu, Mengyu Zhang
- Model implementation: Che-Pai Kung
- Model optimization: Che-Pai Kung
- Software Familiarization: Chenqi Liu, Mengyu Zhang
- Applications: Chenqi Liu