Programming Assignment 7 Hidden Markov Models

Part 1: Group member and contribution

Bin Zhang (5660329599): implantation of HMM for the assignment and report of implantation

Yihang Chen (6338254416): hmmlearn implementation of HMM, and report of software familiarization

Hanzhi Zhang (4395561906): implantation of HMM for the assignment and report of application

Part 2: Implementation

Data loading:

In this project, we are asked to figure out the most likely trajectory of the robot in a given grid-world. First, we load the data from the text file. There are three objects from the data recording the grid-word cells, tower locations and robot noisy measurement. We use the nested list to store this information. For the girds the format is:

```
[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 0, 0, 0, 0, 0, 1, 1, 1],
[1, 1, 0, 1, 1, 1, 0, 1, 1, 1],
[1, 1, 0, 1, 1, 1, 0, 1, 1, 1],
[1, 1, 0, 1, 1, 1, 0, 1, 1, 1],
[1, 1, 0, 1, 1, 1, 0, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]
```

For the tower locations, the format is:

```
[[0, 0], [0, 9], [9, 0], [9, 9]]
```

For the noisy measurement, the format is:

```
[[6.3, 5.9, 5.5, 6.7],
[5.6, 7.2, 4.4, 6.8],
[7.6, 9.4, 4.3, 5.4],
[9.5, 10.0, 3.7, 6.6],
[6.0, 10.7, 2.8, 5.8],
[9.3, 10.2, 2.6, 5.4],
[8.0, 13.1, 1.9, 9.4],
[6.4, 8.2, 3.9, 8.8],
```

[5.0, 10.3, 3.6, 7.2],
[3.8, 9.8, 4.4, 8.8],
[3.3, 7.6, 4.3, 8.5]]

Initial state determining:

After data loading, we build the initial state for the robot, in the grid world cells, only 1 can be accessed by the robot. We use a dictionary to store the valid cell and store its coordinates. In the dictionary, the key refers to the valid state index, the value refers to its corresponding coordinates. There are 87 items in the dictionary. The data structure's format is:

{0: [0, 0],
1: [0, 1],
2: [0, 2],
3: [0, 3],
4: [0, 4],
5: [0, 5], xxxx}

Matrix building:

After initial state building, we build a transition matrix for the state transition. The state transformation is based on robot's current state and its neighbor, for example, for the [0,0], there are only 2 accessible cell that the robot can enter into. So each accessible point's probability is 1/2. In order to record this transition probability of each cell to other cells, we use a numpy's matrix to store it. The matrix's shape is (87*87) and the format is like:

```
[[0.         0.5        0.         ... 0.         0.         0.        ]
[0.33333333 0.         0.33333333 ... 0.         0.         0.        ]
[0.         0.33333333 0.         ... 0.         0.         0.        ]
                                     ...
[0.         0.         0.         ... 0.         0.33333333 0.        ]
[0.         0.         0.         ... 0.33333333 0.         0.33333333]
[0.         0.         0.         ... 0.         0.5        0.        ]]
```

M[i][j] refers to for robot is the ith state, the probability it will enter into jth state.

Then according to the robot's noisy measurements, we build a transmission matrix to record how evidence of each state influences its state. Here the evidence is the distance between robot's location and towers. According to the noisy measurements, the transmission matrix's shape is (11*87), M[i][j] is calculated based on two factors: first one is the noisy measurement recorded, second one is the derived probability from the error range of a state to towers' true distance. Then we can fill out the transmission matrix.

Take the first row as an example, it has 87 numbers which shows the possibility of the robot's location in the corresponding state location based on the observation data.

Viterbi algorithm design:

Once we have prepared the matrices and other data, we now start to design the Viterbi algorithm. Since it's a dynamic programming algorithm and the distinctiveness of HMM model, we stare from the first step. The main idea is that we first eliminate the e1 to add constraints for x1, and then eliminate the x1 to add constraint for x2. In this way, we remove the first step variables. The following step is the same as the first step. We record the generated state at each step for the final track back. From this algorithm, we can get the final state that has the most probability for the robot. And we use the final state and recoded state to find each step state for the robot. And we can use the state to check our defined dictionary to get the corresponding coordinates. Then we get the path of the robot, the most likely path is:

[5, 3]->[6, 3]->[7, 3]->[7, 2]->[7, 1]->[7, 2]->[7, 1]->[6, 1]->[5, 1]->[4, 1]->[3, 1]

Part 3: Software Familiarization

hmmlearn

For the implementation of hmm, we used library hmmlearn. Basically, it can build HMM and generate samples. The code shown below is a sample code from Hmmlearn to build HMM instance

```
>>> import numpy as np
>>> from hmmlearn import hmm
>>> np.random.seed(42)
>>>
>>> model = hmm.GaussianHMM(n_components=3, covariance_type="full")
>>> model.startprob_ = np.array([0.6, 0.3, 0.1])
>>> model.transmat_ = np.array([[0.7, 0.2, 0.1],
...                             [0.3, 0.5, 0.2],
...                             [0.3, 0.3, 0.4]])
>>> model.means_ = np.array([[0.0, 0.0], [3.0, -3.0], [5.0, 10.0]])
>>> model.covars_ = np.tile(np.identity(2), (3, 1, 1))
>>> X, Z = model.sample(100)
```

Here, there are 3 component and thus the transition matrix is 3x3.

Using “sample” function, it can generate samples using built HMM model. It also offers API to train parameters and infers the hidden states.

```
>>> remodel = hmm.GaussianHMM(n_components=3, covariance_type="full", n_iter=100)
>>> remodel.fit(X)
GaussianHMM(algorithm='viterbi', ...
>>> Z2 = remodel.predict(X)
```

Besides, this library can work with multiple sequences. That is to say, we can pass multiple sequences to fit and predict. First, we need to concatenate them into a single array and compute an array of sequence length:

```
>>> X = np.concatenate([X1, X2])
>>> lengths = [len(X1), len(X2)]
```

Then we do fitness and prediction.

```
>>> hmm.GaussianHMM(n_components=3).fit(X, lengths)
GaussianHMM(algorithm='viterbi', ...
```

Could be mentioned, we can choose algorithms to fit data. For now, Viterbi algorithm and maximum a posteriori estimation are supported.

For our problem, we only get the probability of position in each step. It’s quite different with the input in the hmmlearn library example codes. We can build HMM models after observation in each step, but we can hardly predict based on the probability input. Thus, for now we cannot solve the robot problem based on this library.

So we try to solve predict problem using hmmlearn library by building a HMM model. The hypothetical problem is described below:
Assume we have three box and the box has both red and white balls. We know the percentage of red balls to white balls in each box. Now we pick up one ball each time from one box and put it back. We record the sequence of the colors. We can use HMM models to predict the sequence of box we picked balls from. The implementation part is in the HMM_lib.py file.
The example is learned from:
https://www.cnblogs.com/pinard/p/7001397.html

Part 4: Applications

In this project, start matrix, transition matrix, and emission matrix are

already known. We are required to use these matrixes and observed data to predict the actual route. However, it is possible that we cannot have start matrix, transition matrix, and emission matrix. In this case, we can use the known hidden state and observed state to calculate start matrix, transition matrix, and emission matrix. Therefore, we can build a model to predict a new problem. For example, HMM has a wide application on speech recognition. Most of researches are focus on the human language. However, this algorithm can also be applied on the other sounds. For instance, HMM can be used to estimate ship's underwater noise. Use existing training datasets, model can use the obtained the original signal from ship's underwater noise to detect the abnormality of the ship. Furthermore, HMM is also applied on bioinformatics. The four types of DNA, A, G, C, T are the observed state. The meaning behind DNA sequencing is unknown, hidden state. Researchers use the existing training sets to build a HMM model, and use this model to help bioinformatics.