INF552: Programming Assignment 7 [Hidden Markov Models]

Project Report by Arpit Parwal & Yeon-Soo Park

1. Implementation Algorithms

a. Data structures

i. Lists

Python has a great built-in list type named "list". List literals are written within square brackets [ ].

ii. Dictionaries

We used “dictionaries” that contain lists to do sequence data manipulation and store the probability of them efficiently.

b. Explanation of Modules

- **Main Function**
  This main function acts as an umbrella for all the functions performed. It fetches the formatted data from the function load_data. It gets other values required for HMM and then iteratively calls it after which we implement the Viterbi algorithm to calculate the maximum probability of state in the last time-step, then backtrack the state to provide the condition for getting this probability, until the starting state is determined.
- **load_Data**
  The module gets data from the file and categories it into the following matrices**:**
  grid_non_obstacle_cells: The non-obstacle cells are represented as '1', otherwise, '0'.
  tower_loc: There are four towers, one in each of the four corners
  noisy_dist: Robot records a noisy measurement chosen uniformly at random from the set of numbers in the interval [0.7d, 1.3d] with one decimal place.
- **Tower_distance_noisy**
  This module generate distance matrix by using euclidean distance of cells location from each other
- **Get_next_steps_prob**
  The modules generate states and transition probability matrices that help in calculation of the most effective and probable path.
- **find_Moves**
  The modules calculate all possible moves from a cell. It eliminates any illegal move and checks for boundary conditions.

- **HMM**
  Use the function HMM based on observation_sequences (trans_matrix) which implements the ***Viterbi algorithm*** to calculate the maximum probability of state in the last time-step, then backtrack using the function '*backtracking*' to the state to provide the condition for getting this probability, until the starting state is determined.

c. Challenges

- **Reading and calculation of probability distribution table**
  Understanding how to calculate the probability distribution table in an efficient manner. There were many states and observations, which lead to a large data set and difficult to set up the matrix clearly.
- **Viterbi Algorithm Implementation**
  It's confusing to implement the Viterbi Algorithm under the given conditions, especially when distinguishing the matrices needed for the initial state and the following states.
- **Complex Calculations and optimization**
  Calculation is quite complex because of too many relations. It's not easy to debug and hard to check the correctness of every calculation result.

d. Code-level optimization

- Set up the necessary matrixes showing the state-and-state relation, state-and observation relation, matrix recording coordinates of point, and other lists to make calculation easier and the result traceable.

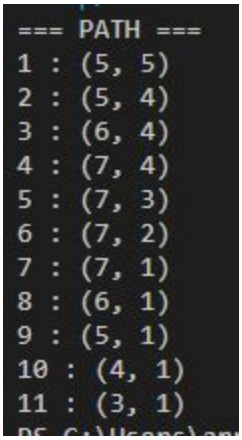
- We save the previous state list when we compute and select the maximum probability of current states. It save another loop to speed up the code

- Modularizing parts of code into methods, making code more readable.

- We used enumerate which allows us to loop over something and have an automatic counter

e. Output

i. Path



2. Software Familiarization

a. Existing Libraries

- **Scikit learn library**

  It's a machine learning library. It includes various machine learning algorithms. HMMlearn follows scikit-learn API as close as possible, but adapted to sequence data..

```
import numpy as np
from hmm import MultinomialHMM
hmm = MultinomialHMM(n_states=3, verbose=0, n_repeats=20)
X = ['a'] * 5 + ['b'] * 5 + ['c'] * 5
# build a simple sequence of characters
X = np.array(X) hmm.fit(X)
# learn the parameters of the model with EM
for i in range(3): print(''.join(hmm.generate(15)))
# generate a sequence of 15 characters from the model
```

- HMMlearn allows to implement other emission probability (e.g. Poisson), by implementing a new HMM class by inheriting the _BaseHMM and overriding the methods __init__, _compute_log_likelihood, _set and _get for additional parameters

b. Comparisons

- **Multiple algorithm options**
  Scikit's HMMlearn implements HMM with emission probabilities determined by multinomial distributions, Gaussian distributions and mixtures of Gaussian distributions. Our program deals with the given matrix and one type of emission. We use the Viterbi algorithm whereas HMMlearn offers three types of algorithm, namely – Viterbi, Forward-Backward algorithm and Baum-Welch algorithm.

- **Flexibility to include various kind of emission data**
  HMMLearn also has a base class called hmmlearn.base which allows for easy evaluation of, sampling from, and maximum a posteriori estimation of the parameters of a HMM. It allows us to deal with various other kinds of emission data unlike the single type of data that our algorithm deals.

c. Improvements

- **Using hash table (dict in python)**
  By using mapping we can make acquiring a certain observation easier. Using it in the implementation of the emission matrix in the current time-step. This modification improves the reliability of the result.

- **Using of Matplotlib to visualise our data**
  HMMLearn is also built with Matplotlib thus it helps us visualize the emissions that are generated

3. Applications

a. Speech Synthesis

The HMM not only does well in speech recognition but also is good at speech synthesis. Hidden Markov model (HMM)-based speech synthesis has recently been demonstrated to be very effective in synthesizing speech. It can change speaker identities, emotions, and speaking styles. Another idea is that an input utterance is recognized utilizing the hidden Markov model (HMM) for speech recognition, and the recognized phoneme sequences are used as labels for speech synthesis.

b. Gene discovery of a DNA

DNA sequences can be considered as texts in the alphabet of four letters that represent the nucleotides. The difference in stochastic properties of Markov chain models for coding and

non-coding regions of DNA can be used for gene finding. GeneMark is the first system that implemented this approach.

c. **Creating Multiple sequence alignment**:
HMMs can be used to automatically create a multiple alignment from a group of unaligned sequences. By taking a close look at the alignment, we can see the history of evolution. One great advantage of HMMs is that they can be estimated from sequences, without having to align the sequences first. The sequences used to estimate or train the model are called the training sequences, and any reserved sequences used to evaluate the models are called the test sequences. The model estimation is done with the forward backward algorithm, also known as the Baum-Welch algorithm. It is an iterative algorithm that maximizes the likelihood of the training sequences.

4. Individual Contribution

Student Name	**Programming Part**	**Documentation**
Arpit Parwal **aparwal@usc.edu**	• Loading data from file • Formatting for matrices • Finding moves	Comparison, improvements, Code-level optimization, applications for HMM
Yeon-Soo Park **yeonsoop@usc.edu**	• Calculation of transition matrices • Implementation of viterbi algorithm	Data structure, Modules, Challenges, improvements, Existing libraries