HMM Lab

Introduction

In this lab, students will work with two Python files: `HMM.py` and `test.py`. The objective is to fill in specific functions within the `HMM.py` file using dynamic programming concepts. The `test.py` file will vary from set to set and contains various test cases to evaluate the correctness of the students' implementations.

Files Provided

1. HMM.py

This file contains the implementation of the Hidden Markov Model (HMM) class. Within this class, students are required to fill in two key functions:

- **Viterbi Algorithm**: This function determines the most likely sequence of hidden states given a sequence of observations.
- **Likelihood Calculation**: This function computes the likelihood of a given observation sequence.

2. test.py

This file includes test cases designed to validate the functionality of the methods implemented in `HMM.py`. Each set of tests will evaluate different aspects of the students' code.

Function Explanation

The primary functions that students need to implement are as follows:

1. Viterbi Algorithm

- **Purpose**: To find the most likely sequence of hidden states given an observation sequence.
- **Dynamic Programming Approach**:
    - The Viterbi algorithm uses a dynamic programming approach to efficiently compute the most probable state sequence. It does this by breaking down the problem into smaller subproblems, storing intermediate results, and using them to build the final solution.

- Specifically, the algorithm maintains a matrix (often called the trellis) where each entry represents the maximum probability of reaching a specific state at a given time step. This matrix is filled iteratively based on previously computed probabilities, effectively avoiding redundant calculations.
- At each time step, the algorithm considers all possible transitions from the previous states, calculates the probabilities of these transitions, and stores the best one. Finally, it backtracks from the last time step to determine the most likely sequence of hidden states.

- **Output**: A list of predicted hidden states that corresponds to the input observation sequence.

2. Likelihood Calculation

- **Purpose**: To compute the likelihood of a specific observation sequence based on the HMM parameters.
- **Dynamic Programming Approach**:
  - The likelihood calculation uses a dynamic programming method known as the forward algorithm. This algorithm calculates the probability of observing a sequence of events by summing over all possible hidden state sequences that could generate the observed data.
  - Similar to the Viterbi algorithm, the forward algorithm uses a matrix to store intermediate likelihood values. For each observation in the sequence, it calculates the likelihood of being in each possible hidden state by considering all possible paths leading to that state from the previous observations.
  - This process involves recursively combining the likelihoods of the previous states with the emission probabilities of the current observation, ensuring that all potential state transitions are accounted for in a computationally efficient manner.
- **Output**: A floating-point number representing the likelihood of the observation sequence.

Note - In order to run the file, rename your solution file to CAMPUS_SECTION_SRN_Lab6.py and run the command python3 test.py --ID CAMPUS_SECTION_SRN_Lab6

```
(MI) gayathrinettem@Gayathris-MacBook-Air Instructor Copy % python3 test.py --ID RR_G_PES1UG21CS378_Lab6
Test Case 1 for the Viterbi algorithm PASSED
Test Case 2 for the likelihood PASSED
Test Case 3 for the likelihood PASSED
```

Variable Naming Convention

To encourage independent problem-solving and discourage direct solutions from AI, the variables within the HMM class have been given unconventional names. Below is a legend for faculty reference:

- **avocado**: State transition matrix
- **bubblegum**: Initial state distribution (priors)
- **mushroom**: List of hidden states

- **cheese**: Number of hidden states
- **kangaroo**: Emission probabilities
- **spaceship**: List of observations
- **jellybean**: Number of observations
- **pancake**: Likelihood value for the first observation sequence
- **lemon**: List of observations for the second likelihood calculation
- **jam**: Likelihood value for the second observation sequence
- **rat**: Output of the Viterbi algorithm
- **mouse**: Expected output of the Viterbi algorithm

Test Cases

The `test.py` file includes three primary test cases to validate the functionality of the implemented methods:

1. **Test Case for the Viterbi Algorithm**: This test checks whether the predicted sequence of hidden states using the Viterbi algorithm matches the expected output.
2. **Test Case for Likelihood Calculation**: This test verifies that the computed likelihood value is within a specified range, ensuring the accuracy of the likelihood calculation.
3. **Test Case for Likelihood with Time Limit**: This test checks the likelihood value for a different set of observations and ensures that the computation completes within a specified time limit.

Hidden Test Cases

In addition to the primary test cases, hidden test cases have been prepared to further evaluate the students' implementations. These hidden cases will help ensure that students' solutions can handle a variety of scenarios not covered by the primary tests. For each test case given, a corresponding hidden test case has also been prepared.

In case of any clarifications, please feel free to contact:

1. Nettem Gayathri - +91 90198 64716
2. Arvin Nooli - +91 96068 36869
3. Saarth Vardhan - +91 82967 64552