INF552: Programming Assignment 7 [HMM]

Yi-Li Chen

Part 1: Implementation

- All imported library

```python
import numpy as np
```

- Overview of the HMM algorithm

```python
class HMM:
    def __init__(self, ann, bnm, pi, O):
        self.A = np.array(ann, np.float)
        self.B = np.array(bnm, np.float)
        self.Pi = np.array(pi, np.float)
        self.O = np.array(O, np.float)
        self.N = self.A.shape[0]
        self.M = self.B.shape[1]
    def viterbi(self):
        T = len(self.O)
        I = np.zeros(T, np.float
        delta = np.zeros((T, self.N), np.float)
        psi = np.zeros((T, self.N), np.float)
        for i in range(self.N):
            delta[0, i] = self.Pi[i] * self.B[i, int(self.O[0])]
            psi[0, i] = 0
        for t in range(1, T):
            for i in range(self.N):
                delta[t, i] = self.B[i, int(self.O[t])] * np.array( [delta[t-1,j] * self.A[j,i]
                    for j in range(self.N)] ).max()
                psi[t,i] = np.array( [delta[t-1,j] * self.A[j,i]
                    for j in range(self.N)] ).argmax()
        P_T = delta[T-1, :].max()
        I[T-1] = delta[T-1, :].argmax()
        for t in range(T-2, -1, -1):
            print(I[t+1])
            I[t] = psi[t+1, int(I[t+1])]
        return I
```

- First initialize all needed parameters:

  - A: state transition matrix

  - B: emission probability matrix

  - Pi: initial state probability distribution

  - O: observation sequence

  - N: number of states

  - M: number of observables

```
def __init__(self, ann, bnm, pi, O):
    self.A = np.array(ann, np.float)
    self.B = np.array(bnm, np.float)
    self.Pi = np.array(pi, np.float)
    self.O = np.array(O, np.float)
    self.N = self.A.shape[0]
    self.M = self.B.shape[1]
```

- After initializing all parameters, I remove one observe node and remove one state node afterwards. I remove each of them recursively until it only remains the last node. I used delta to store the probabilities and based on them to get the most likely sequence of values and store in list.

```
def viterbi(self):
    T = len(self.O)
    I = np.zeros(T, np.float)
    delta = np.zeros((T, self.N), np.float)
    psi = np.zeros((T, self.N), np.float)
    for i in range(self.N):
        delta[0, i] = self.Pi[i] * self.B[i, int(self.O[0])]
        psi[0, i] = 0
    for t in range(1, T):
        for i in range(self.N):
            delta[t, i] = self.B[i, int(self.O[t])] * np.array([delta[t-1,j] * self.A[j,i]
                for j in range(self.N)] ).max()
            psi[t,i] = np.array( [delta[t-1,j] * self.A[j,i]
                for j in range(self.N)] ).argmax()
    P_T = delta[T-1, :].max()
    I[T-1] = delta[T-1, :].argmax()
    for t in range(T-2, -1, -1):
        I[t] = psi[t+1, int(I[t+1])]
        return I, P_T
```

- The result of the algorithm
  Sequence:
  ```
  array([7., 6., 5., 6., 5., 4., 5., 6., 7., 8.])
  ```
  Probability:
  ```
  3.3076343375840383e-09
  ```

At the beginning, it is really hard for me to figure it out how to apply Hidden Markov Model on this problem, since the notion is kind of abstract. But after I drew all the possibilities as tree by hand and tried to write the CPTs for the first observe and state. It seems that all of them are connected.