MI LAB 7

NAGAVENI L G

PES2UG21CS315

5F

EC CAMPUS

CODE:

import torch

class HMM:

    """

    HMM model class

    Args:

        A: State transition matrix

        states: list of states

        emissions: list of observations

        B: Emmision probabilites

    """

    def \_\_init\_\_(self, A, states, emissions, pi, B):

        self.A = A

        self.B = B

        self.states = states

        self.emissions = emissions

        self.pi = pi

        self.N = len(states)

        self.M = len(emissions)

        self.make\_states\_dict()

    def make\_states\_dict(self):

        """

        Make dictionary mapping between states and indexes

        """

        self.states\_dict = {state: i for i, state in enumerate(self.states)}

        self.emissions\_dict = {emission: i for i, emission in enumerate(self.emissions)}

    def viterbi\_algorithm(self, seq):

        """

        Function implementing the Viterbi algorithm

        Args:

            seq: Observation sequence (list of observations. must be in the emmissions dict)

        Returns:

            Porbability of the hidden state at time t given an obeservation sequence

        """

        # Initialize variables

        T = len(seq)  # Length of observation sequence

        delta = torch.zeros((T, self.N))  # Initialize the delta matrix

        psi = torch.zeros((T, self.N), dtype=torch.long)  # Initialize the psi matrix

        # Find the index for the first observation

        first\_obs\_index = self.emissions\_dict[seq[0]]

        # Initialization Step: Calculate initial probabilities for the first observation

        pi\_tensor = torch.tensor(self.pi)  # Convert pi to a tensor if it's not already

        delta[0] = pi\_tensor \* self.B[:, first\_obs\_index]

        # Recursion Step: Calculate probabilities for subsequent observations

        for t in range(1, T):

            obs\_index = self.emissions\_dict[seq[t]]

            for j in range(self.N):

                # Calculate delta values and back pointers (psi)

                probabilities = delta[t - 1] \* self.A[:, j] \* self.B[j, obs\_index]

                delta[t, j], psi[t, j] = torch.max(probabilities, 0)

        # Termination Step: Backtracking to find the most probable sequence

        states\_sequence = [0] \* T

        \_, states\_sequence[T - 1] = torch.max(delta[T - 1], 0)

        for t in range(T - 2, -1, -1):

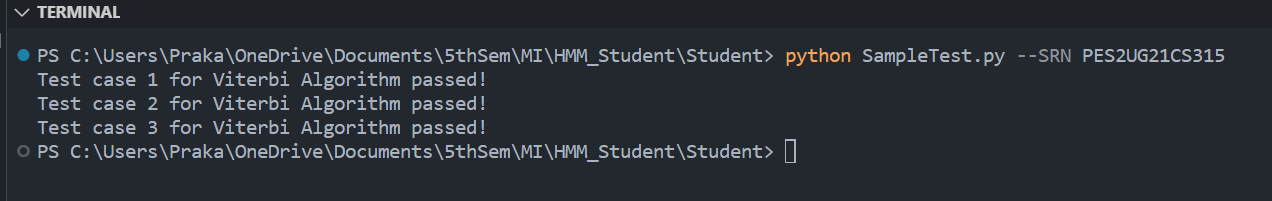
            states\_sequence[t] = psi[t + 1, states\_sequence[t + 1]]

        # Convert indices to state names

        decoded\_states = [self.states[state\_index] for state\_index in states\_sequence]

        return decoded\_states

OUTPUT:



THANK YOU 😊