ABIN Assignment 4

Group 16

Q1

Defining function

```
def Q1(observations, states, start_p, trans_p, emit_p):
    V = [{}]
    path = {}

    # base case
    for state in states:
        V[0][state] = start_p[state] * emit_p[state][observations[0]]
        path[state] = [state]

    # forward pass
    for t in range(1, len(observations)):
        V.append({})
        new_path = {}

        for current_state in states:
            (max_prob, prev_state) = max(
                (V[t - 1][prev_state] * trans_p[prev_state][current_state] * emit_p[current_state][observations[t]], prev_state)
                for prev_state in states
            )
            V[t][current_state] = max_prob
            new_path[current_state] = path[prev_state] + [current_state]

        path = new_path

    # Backtrack for most probable hidden states
    (max_prob, best_path) = max((V[len(observations) - 1][final_state], path[final_state]) for final_state in states)

    # probabilities of hidden states for each step
    state_probabilities = {}
    for t in range(len(observations)):
        state_probabilities[t + 1] = {}
        for state in states:
            state_probabilities[t + 1][state] = V[t][state]

    return state_probabilities, max_prob, best_path
```

Code explanation:

This function performs the Viterbi algorithm to find the most likely sequence of hidden states that could have generated a given sequence of observations.

Step 1: Initialization

Create an empty list V to store the trellis table. Create an empty dictionary path to store the backpointers. For each state, calculate the initial probability V[0][state] and store the corresponding state in path[state].

Step 2: Forward Pass

Iterate over the observations from index 1 to the end. For each state at the current time step, compute the maximum probability of reaching that state from any of the states at the previous time step. Store the maximum probability in V[t][current_state] and the corresponding backpointer in path[current_state].

Step 3: Backtracking

Find the most probable final state and its corresponding probability. Backtrack through the path dictionary to recover the most likely sequence of hidden states.

Step 4: Calculate State Probabilities

For each time step, calculate the probability of each state using the values in the trellis table V.

Step 5: Return Results

Return the state probabilities, the maximum probability of the most likely path, and the most likely path itself.

Input

```
observations = ("normal", "cold", "dizzy")
states = ("Healthy", "Fever")
start_p = {"Healthy": 0.6, "Fever": 0.4}
trans_p = {
    "Healthy": {"Healthy": 0.7, "Fever": 0.3},
    "Fever": {"Healthy": 0.4, "Fever": 0.6},
}
emit_p = {
    "Healthy": {"normal": 0.5, "cold": 0.4, "dizzy": 0.1},
    "Fever": {"normal": 0.1, "cold": 0.3, "dizzy": 0.6},
}
```

Output

```
output = Q1(observations, states, start_p, trans_p, emit_p)

state_probabilities = output[0]
max_prob = output[1]
best_path = output[2]

for t in state_probabilities:
  print(f"Step {t} State Probabilities: {state_probabilities[t]}")

print()
print(f"Highest Probability at Step 3: {max_prob}")

print()
print("Most Probable Hidden States:", best_path)
```

```
Step 1 State Probabilities: {'Healthy': 0.3, 'Fever': 0.04000000000000001}
Step 2 State Probabilities: {'Healthy': 0.084, 'Fever': 0.027}
Step 3 State Probabilities: {'Healthy': 0.00588, 'Fever': 0.01512}

Highest Probability at Step 3: 0.01512

Most Probable Hidden States: ['Healthy', 'Healthy', 'Fever']
```

Q2

Defining function

```
def Q2(x, states, transition, emission):
    n = len(x)
    m = len(states)

    # Initialize the probabilities matrix
    forward = [[0.0] * m for _ in range(n)]

    # Initialize the initial probabilities
    for i, state in enumerate(states):
        forward[0][i] = 1.0 / m * emission[state][x[0]]

    # Calculate forward probabilities
    for t in range(1, n):
        for j, state_to in enumerate(states):
            forward[t][j] = sum(forward[t - 1][i] * transition[states[i]][state_to] * emission[state_to][x[t]] for i in range(m))

    # Sum the final forward probabilities
    pr_x = sum(forward[n - 1])

    return pr_x
```

Code explanation:

Step 1: Initialization

Initialize a matrix forward to store the forward probabilities. Initialize the initial probabilities in the forward matrix.

Step 2: Calculate Forward Probabilities

For each time step, iterate over the possible hidden states at that time step. For each state at the current time step, calculate the sum of the products of the forward probabilities at the previous time step, the transition probabilities from the previous states to the current state, and the emission probabilities of the current state given the observation at the current time step. Store the calculated probability in the forward matrix.

Step 3: Calculate Final Probability

Sum the final forward probabilities to obtain the probability of the entire sequence of observations.

Step 4: Return Result

Return the calculated probability pr_x.

Input

```
x = "xzyyzzyzyy"
alphabet = "xyz"
states = "AB"
transition = {
    'A': {'A': 0.303, 'B': 0.697},
    'B': {'A': 0.831, 'B': 0.169}
}
emission = {
    'A': {'x': 0.533, 'y': 0.065, 'z': 0.402},
    'B': {'x': 0.342, 'y': 0.334, 'z': 0.324}
}
```

Output

```
probability = Q2(x, states, transition, emission)
print(probability)
```

```
1.1005510319694851e-06
```