Homework 9

Submission by : Gagan Ullas Nirgun , Viswadeep Mallarapu Bhaskar

In [43]:
```
import numpy as np
import pandas as pd
```

In [44]:
```
observations = np.genfromtxt('hmm_pb1.csv', delimiter=',', dtype=int)

log_prob_fair = np.log(1/6)
log_prob_loaded = np.array([np.log(1/10), np.log(1/10), np.log(1/10), np.log(1/10), np.log(1/10), np.log(1/2)])

# Transition probabilities
transition_probs = np.log(np.array([[0.95, 0.05], [0.05, 0.95]]))

# Start probabilities
start_probs = np.log(np.array([0.5, 0.5]))

# Emission probabilities
emission_probs = np.array([np.full(6, log_prob_fair), log_prob_loaded])
```

In [45]:
```
import numpy as np

def viterbi(observations, start_probs, transition_probs, emission_probs):
    T = len(observations)
    N = transition_probs.shape[0]

    # Initialize the matrices
    C = np.zeros((N, T))
    Ptr = np.zeros((N, T), dtype=int)

    # Initialization step
    for i in range(N):
        C[i, 0] = emission_probs[i, observations[0] - 1] + start_probs[i]

    # Recursion step
    for t in range(1, T):
        for k in range(N):
            max_prob = float('-inf')
            max_idx = -1
            for i in range(N):
                prob = C[i, t - 1] + transition_probs[i, k]
                if prob > max_prob:
                    max_prob = prob
                    max_idx = i
            C[k, t] = emission_probs[k, observations[t] - 1] + max_prob
            Ptr[k, t] = max_idx

    # Termination step
    y_T_star = np.argmax(C[:, T - 1])
    max_prob = C[y_T_star, T - 1]

    # Backtracking
    y = np.zeros(T, dtype=int)
    y[T - 1] = y_T_star
    for t in range(T - 2, -1, -1):
        y[t] = Ptr[y[t + 1], t + 1]
```

```
    return y

# Load the observations


# Run Viterbi algorithm
most_likely_sequence = viterbi(observations, start_probs, transition_probs, emission_probs)

# Convert the sequence to describe as 1 and 2
sequence_description = np.where(most_likely_sequence == 0, 1, 2)

print("--------------- 1 a Solution ---------------")
print()
print("obtained sequence (y):", sequence_description)
```

```
--------------- 1 a Solution ---------------

obtained sequence (y): [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1]
```

In [42]:

```
def forward_algorithm(observations, start_probs, transition_probs, emission_probs):
    T = len(observations)
    num_states = len(start_probs)

    alpha = np.zeros((T, num_states))
    scaling_factors = np.zeros(T)

    # Initialization step
    scaling_factors[0] = 1.0
    for k in range(num_states):
        alpha[0][k] = emission_probs[k][observations[0] - 1] * start_probs[k]

    # Scaling factor for t = 0
    scaling_factors[0] = 1.0 / np.sum(alpha[0])
    alpha[0] *= scaling_factors[0]

    # Forward algorithm
    for t in range(1, T):
        for k in range(num_states):
            alpha[t][k] = emission_probs[k][observations[t] - 1] * np.sum(alpha[t-1] * transition_probs[:, k])

        # Scaling factor for current time step
        scaling_factors[t] = 1.0 / np.sum(alpha[t])
        alpha[t] *= scaling_factors[t]

    return alpha, scaling_factors

def backward_algorithm(observations, transition_probs, emission_probs, scaling_factors):
    T = len(observations)
    num_states = len(transition_probs)

    beta = np.zeros((T, num_states))

    # Initialization step
    beta[T-1] = 1.0

    # Backward algorithm
    for t in range(T-2, -1, -1):
```

```
        for k in range(num_states):
            for i in range(num_states):
                beta[t][k] += beta[t+1][i] * transition_probs[k][i] * emission_probs[i][observations[t+1] - 1]

        # Scaling factor for current time step
        beta[t] *= scaling_factors[t+1]

    return beta

# Given probabilities
prob_fair = 1/6
prob_loaded = np.array([1/10, 1/10, 1/10, 1/10, 1/10, 1/2])

# Transition probabilities
transition_probs = np.array([[0.95, 0.05],
                             [0.05, 0.95]])

# Start probabilities
start_probs = np.array([0.5, 0.5])

# Emission probabilities
emission_probs = np.array([np.full(6, prob_fair), prob_loaded])


alpha,scaling_factors = forward_algorithm(observations, start_probs, transition_probs, emission_probs)
beta = backward_algorithm(observations, transition_probs, emission_probs, scaling_factors)


alpha_138_state_1 = alpha[137][0]
alpha_138_state_2 = alpha[137][1]


beta_138_state_1 = beta[137][0]
beta_138_state_2 = beta[137][1]

print("--------------- 1 b Solution ---------------")
print()
print("Alpha_138 for State 1/Alpha_138 for State 2 : ", alpha_138_state_1/alpha_138_state_2)
print("beta_138_state_1/beta_138_state_1 : ", beta_138_state_1/beta_138_state_2)
```

```
--------------- 1 b Solution ---------------

Alpha_138 for State 1/Alpha_138 for State 2 :  9.795724733488365
beta_138_state_1/beta_138_state_1 :  5.761919016897876
```

In [46]:

```
def compute_xi(alpha, beta, transition_probs, emission_probs, observations):
    T = len(observations)
    num_states = transition_probs.shape[0]
    xi = np.zeros((T-1, num_states, num_states))

    for t in range(T-1):
        for i in range(num_states):
            for j in range(num_states):
                xi[t, i, j] = alpha[t, i] * transition_probs[i, j] * np.exp(emission_probs[j, observations[t+1] - 1]) * beta[t+1, j]
        xi[t] /= np.sum(xi[t])

    return xi

def compute_gamma(alpha, beta):
    gamma = alpha * beta
    gamma /= np.sum(gamma, axis=1, keepdims=True)
    return gamma

def baum_welch(observations, pi, transition_probs, emission_probs, num_iterations=100):
    for _ in range(num_iterations):
```

```
        # Expectation step
        alpha,scaling_factors = forward_algorithm(observations, start_probs, transition_probs, emission_probs)
        beta = backward_algorithm(observations, transition_probs, emission_probs, scaling_factors)

        # Compute xi and gamma
        xi = compute_xi(alpha, beta, transition_probs, emission_probs, observations)
        gamma = compute_gamma(alpha, beta)

        # Maximization step
        pi = gamma[0]

        # Update a
        for i in range(transition_probs.shape[0]):
            for j in range(transition_probs.shape[1]):
                transition_probs[i, j] = np.sum(xi[:, i, j]) / np.sum(gamma[:, i])

        # Update b
        for i in range(emission_probs.shape[0]):
            for j in range(emission_probs.shape[1]):
                emission_probs[i, j] = np.sum((observations == j+1) * gamma[:, i]) / np.sum(gamma[:, i])

    return pi, transition_probs, emission_probs


observations = np.genfromtxt('hmm_pb2.csv', delimiter=',', dtype=int)

# Giving random values
num_states = 2
pi = np.random.rand(num_states)
pi /= np.sum(pi)

transition_probs = np.random.rand(num_states, num_states)
transition_probs /= np.sum(transition_probs, axis=1, keepdims=True)

emission_probs = np.random.rand(num_states, 6)
emission_probs /= np.sum(emission_probs, axis=1, keepdims=True)

print("Initial state probabilities (pi):\n", pi)
print("Sum of (pi):", sum(pi))
print()
print("Initial transition probabilities (A):\n", transition_probs)
row_sums_A = np.sum(transition_probs, axis=1)
print("Sum of each row in A:", row_sums_A)
print()
print("Initial emission probabilities (B):\n", emission_probs)
row_sums_B = np.sum(emission_probs, axis=1)
print("Sum of each row in B:", row_sums_B)
print()

pi_estimated, transition_probs_estimated, emission_probs_estimated = baum_welch(observations, pi, transition_probs, emission_probs)


print("--------------- 2 Solution ---------------")
print()
print("Obtained state probabilities (pi):\n", pi_estimated)
print("Obtained transition probabilities (A):\n", transition_probs_estimated)
print("Obtained emission probabilities (B):\n", emission_probs_estimated)
```

```
Initial state probabilities (pi):
 [0.45100902 0.54899098]
Sum of (pi): 1.0

Initial transition probabilities (A):
 [[0.81616342 0.18383658]
 [0.52119392 0.47880608]]
Sum of each row in A: [1. 1.]

Initial emission probabilities (B):
 [[0.16358905 0.15592971 0.24471592 0.27944486 0.06724569 0.08907477]
 [0.14862719 0.14899228 0.12777206 0.31624405 0.2196553  0.03870913]]
Sum of each row in B: [1. 1.]

--------------- 2 Solution ---------------

Obtained state probabilities (pi):
 [0.78852526 0.21147474]
Obtained transition probabilities (A):
 [[0.88103219 0.11761893]
 [0.29787389 0.70538668]]
Obtained emission probabilities (B):
 [[0.20895491 0.20889349 0.20188789 0.18869351 0.09285253 0.09871767]
 [0.07727769 0.06164216 0.08907962 0.21781655 0.21279236 0.34139162]]
```