Aim - Design and Implementation of a Hidden Markov Model for Outcome Prediction

AAI EXP - 2 Yash Ashok Shirsath

```
import numpy as np

class HMM:
    def __init__(self, states, observations, start_prob, trans_prob, emission_prob):
        self.states = states
        self.observations = observations
        self.start_prob = np.array(start_prob)
        self.trans_prob = np.array(trans_prob)
        self.emission_prob = np.array(emission_prob)

        assert len(self.start_prob) == len(self.states), "Start probability vector dimension mismatch."
        assert self.trans_prob.shape == (len(self.states), len(self.states)), "Transition probability matrix dimension mismatch."
        assert self.emission_prob.shape == (len(self.states), len(self.observations)), "Emission probability matrix dimension mismatch."

    def forward(self, obs_seq):
        T = len(obs_seq)
        N = len(self.states)
        alpha = np.zeros((T, N))

        for s in range(N):
            alpha[0, s] = self.start_prob[s] * self.emission_prob[s, obs_seq[0]]

        for t in range(1, T):
            for s in range(N):
                alpha[t, s] = np.sum(alpha[t-1, :] * self.trans_prob[:, s]) * self.emission_prob[s, obs_seq[t]]

        return alpha

    def viterbi(self, obs_seq):
        T = len(obs_seq)
        N = len(self.states)
        delta = np.zeros((T, N))
        psi = np.zeros((T, N), dtype=int)


        for s in range(N):
            delta[0, s] = self.start_prob[s] * self.emission_prob[s, obs_seq[0]]
            psi[0, s] = 0


        for t in range(1, T):
            for s in range(N):
                delta[t, s] = np.max(delta[t-1, :] * self.trans_prob[:, s]) * self.emission_prob[s, obs_seq[t]]
```

```
                delta[t, s] = np.max(delta[t-1, :] * self.trans_prob[:, s]) * self.emission_prob[s, obs_seq[t]]
                psi[t, s] = np.argmax(delta[t-1, :] * self.trans_prob[:, s])

        path = [np.argmax(delta[T-1, :])]
        for t in range(T-2, -1, -1):
            path.insert(0, psi[t+1, path[0]])

        return path, delta

states = ["Rainy", "Sunny"]
observations = ["walk", "shop", "clean"]
start_probability = [0.6, 0.4]
transition_probability = [
    [0.7, 0.3],
    [0.4, 0.6]
]
emission_probability = [
    [0.1, 0.4, 0.5],
    [0.6, 0.3, 0.1]
]

hmm_model = HMM(states, observations, start_probability, transition_probability, emission_probability)
obs_seq = [0, 1, 2]  # walk, shop, clean (index into observations)

alpha = hmm_model.forward(obs_seq)
print("Forward probabilities:\n", alpha)
path, delta = hmm_model.viterbi(obs_seq)
print("\nMost likely sequence of hidden states:\n", [states[i] for i in path])
decoded_observations = [observations[i] for i in obs_seq]
print("\nObservations:", decoded_observations)
```

```
Forward probabilities:
 [[0.06     0.24    ]
 [0.0552   0.0486  ]
 [0.02904  0.004572]]

Most likely sequence of hidden states:
 ['Sunny', 'Rainy', 'Rainy']

Observations: ['walk', 'shop', 'clean']
```