LAB 11 : Hidden Markov Model

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Please refer to the following article to understand Hidden Markov Model

Here we will be dealing with 3 major problems :

1. Evaluation Problem
2. Learning Problem
3. Decoding Problem

1. Evaluation Problem : Implementation of Forward and Backward Algorithm

In [ ]:
```
data = pd.read_csv('data_python.csv') ## Read the data, change the path acc

V = data['Visible'].values

# Transition Probabilities
a = np.array(((0.54, 0.46), (0.49, 0.51)))

# Emission Probabilities
b = np.array(((0.16, 0.26, 0.58), (0.25, 0.28, 0.47)))

# Equal Probabilities for the initial distribution
initial_distribution = np.array((0.5, 0.5))


def forward(V, a, b, initial_distribution):
        # Probability that hidden state at time t is i given the observatio
        alpha = np.zeros((V.shape[0], a.shape[0]))
        # Define alpha for t=0
        alpha[0, :] = initial_distribution * b[:, V[0]]

        # Define other alpha values
        for t in range(1, V.shape[0]):
                for j in range(a.shape[0]):
                        alpha[t, j] = np.dot(alpha[t-1, :], a[:, j]) * b[j,

        return alpha


alpha = forward(V, a, b, initial_distribution)


def backward(V, a, b):
        # Probability that the hidden state at time t is i given the observa
        beta = np.zeros((V.shape[0], a.shape[0]))

        # Set beta to one for last time step
        beta[V.shape[0]-1, :] = np.ones(a.shape[0])

        # Define other beta values
        for t in range(V.shape[0]-2, -1, -1):
                for j in range(a.shape[0]):
```

```
                    beta[t, j] = np.dot(beta[t+1, :] * b[:, V[t+1]], a[j

        return beta


beta = backward(V, a, b)
```

1. Learning Problem : Implementation of Baum Welch Algorithm

In [ ]:
```
def baum_welch(V, a, b, initial_distribution, n_iter=100):
        # Number of hidden states
        M = a.shape[0]
        # Number of emission states
        K = b.shape[1]
        # Number of observations
        T = len(V)

        for _ in range(n_iter):
                # Calculate alpha and beta
                alpha = forward(V, a, b, initial_distribution)
                beta = backward(V, a, b)

                # Calculate xi
                # ie. probability of being in state i at time t and state j
                xi = np.zeros((M, M, T-1))

                for t in range(T-1):
                        for i in range(M):
                                for j in range(M):
                                        xi[i, j, t] = alpha[t, i] * a[i, j]

                # Normalize xi
                for t in range(T-1):
                        xi[:, :, t] /= np.sum(xi[:, :, t])

                # Calculate gamma
                # ie. probability of being in state i at time t given the o
                gamma = np.sum(xi, axis=1)

                # Recalculate transition probabilities
                a = np.sum(xi, axis=2) / np.sum(gamma, axis=1)

                # Add the last gamma ie. at time t=T
                gamma = np.hstack((gamma, np.sum(xi[:, :, T-2], axis=0).resh

                # Recalculate emission probabilities
                denom = np.sum(gamma, axis=1)

                for k in range(K):
                        b[:, k] = np.sum(gamma[:, V == k], axis=1)

                b /= denom.reshape((-1, 1))

        return (a,b)

data = pd.read_csv('data_python.csv')

V = data['Visible'].values

# Transition Probabilities
a = np.ones((2, 2))
a = a / np.sum(a, axis=1)
```

```
# Emission Probabilities
b = np.array(((1, 3, 5), (2, 4, 6)))
b = b / np.sum(b, axis=1).reshape((-1, 1))

# Equal Probabilities for the initial distributionhidden markov model
initial_distribution = np.array((0.5, 0.5))

a,b = baum_welch(V, a, b, initial_distribution, n_iter=100)
```

1. Decoding Problem : Implementation of Viterbi Algorithm

In [ ]:
```
def viterbi(V, a, b, initial_distribution):
        # Number of hidden states
        M = a.shape[0]
        # Number of observations
        T = V.shape[0]

        # Calculate omega
        # ie. at time t probability of being in state i given the previous (
        # Note we use the log scale to avoid underflow
        omega = np.zeros((T, M))
        omega[0, :] = np.log(initial_distribution * b[:, V[0]])

        # prev matrix store the most likely previous state
        prev = np.zeros((T-1, M))

        for t in range(1, T):
                for j in range(M):
                        # Calculate probability of being in state j coming 
                        probabilities = omega[t-1, :] + np.log(a[:, j]) + np

                        # Store the most likely previous state
                        prev[t-1, j] = np.argmax(probabilities)

                        # Store the probability of being in state j at time
                        omega[t, j] = np.max(probabilities)

        # Calculate the most likely path
        path = np.zeros(T, dtype=int)

        # The last state is the one with the highest probability
        last = np.argmax(omega[T-1, :])

        # Calculate the rest of the path
        for t in range(T-2, -1, -1):
                path[t] = prev[t, path[t+1]]

        # Convert state indices to names
        states = {0: 'A', 1: 'B'}
        result = [states[int(i)] for i in path]

        return result


data = pd.read_csv('data_python.csv')

V = data['Visible'].values

# Transition Probabilities
a = np.ones((2, 2))
a = a / np.sum(a, axis=1)
```

```
# Emission Probabilities
b = np.array(((1, 3, 5), (2, 4, 6)))
b = b / np.sum(b, axis=1).reshape((-1, 1))

# Equal Probabilities for the initial distribution
initial_distribution = np.array((0.5, 0.5))

a, b = baum_welch(V, a, b, initial_distribution, n_iter=100)

result1 = viterbi(V, a, b, initial_distribution)

print(result1)

accuracy = np.mean(result1 == data['Hidden'].values)
```

```
['B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'B', 'A',
'B', 'B', 'A', 'A', 'A', 'B', 'A', 'B', 'A', 'A', 'B', 'A', 'A', 'A', 'B',
'B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'B',
'B', 'B', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'B', 'B',
'A', 'B', 'B', 'A', 'B', 'B', 'B', 'A', 'B', 'B', 'B', 'A', 'B', 'B', 'B',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'B', 'B', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'A',
'B', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'A', 'A', 'B', 'A', 'B', 'B', 'B',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'B', 'B', 'A', 'B', 'B', 'B',
'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B',
'A', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'A', 'A', 'A',
'B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'B', 'A', 'B', 'B', 'B', 'A', 'B', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'B',
'B', 'A', 'A', 'A', 'B', 'B', 'A', 'B', 'A', 'A', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'A', 'A', 'B', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'A', 'A',
'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'A', 'A', 'B', 'A', 'B',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B',
'A', 'A', 'A', 'A', 'A', 'B', 'B', 'A', 'B', 'B', 'B', 'B', 'B', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'B', 'A', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A',
'B', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'A', 'A', 'A', 'A', 'B', 'B', 'A', 'B', 'B', 'A', 'A', 'A', 'B', 'A',
'B', 'B', 'A', 'A', 'B', 'A', 'A', 'B', 'A', 'A', 'A', 'A', 'B', 'A', 'A',
'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'A',
'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A', 'B', 'B',
'B', 'B', 'A', 'B', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'A', 'A', 'A',
'A', 'A', 'B', 'A', 'A', 'A', 'A', 'B', 'B', 'A', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'A', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'A', 'A',
'A', 'A', 'A', 'A', 'A']
```

1. Use the built-in **hmmlearn** package to fit the data and generate the result using the decoder

In [ ]:
```
#%pip install hmmlearn
from hmmlearn import hmm
```

In [ ]:
```
# Load the data
data = pd.read_csv('data_python.csv')

V = data['Visible'].values
```

```python
a = np.ones((2, 2))
a = a / np.sum(a, axis=1)

b = np.array(((1, 3, 5), (2, 4, 6)))
b = b / np.sum(b, axis=1).reshape((-1, 1))

initial_distribution = np.array((0.5, 0.5))

# Use the hmmlearn library to train the model
model = hmm.CategoricalHMM(n_components=2)
model.startprob_ = initial_distribution
model.transmat_ = a
model.emissionprob_ = b

# Predict the most likely hidden states
log_prob, result2 = model.decode([V])
states = {0: 'A', 1: 'B'}
result2 = [states[i] for i in result2]

print(result2)

accuracy = np.mean(result2 == data['Hidden'].values)
```

```
['B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'A', 'B', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'A', 'B', 'A', 'B', 'A',
'B', 'B', 'A', 'B', 'A', 'B', 'A', 'B', 'A', 'A', 'B', 'A', 'A', 'A', 'B',
'B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'B', 'A', 'B',
'B', 'B', 'B', 'A', 'B', 'A', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'B', 'B',
'A', 'B', 'B', 'A', 'B', 'B', 'B', 'A', 'B', 'B', 'B', 'A', 'B', 'B', 'B',
'A', 'B', 'A', 'B', 'B', 'A', 'B', 'A', 'A', 'A', 'B', 'A', 'B', 'A', 'B',
'B', 'B', 'A', 'A', 'B', 'A', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'B', 'B', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'A',
'B', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'A', 'A', 'B', 'A', 'B', 'B', 'B',
'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'B', 'A',
'B', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'B', 'A',
'B', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A',
'A', 'B', 'A', 'B', 'A', 'B', 'A', 'B', 'A', 'B', 'B', 'A', 'B', 'B', 'B',
'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B',
'A', 'B', 'B', 'A', 'A', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'A', 'A', 'A',
'B', 'B', 'B', 'B', 'A', 'A', 'B', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'A',
'B', 'A', 'B', 'B', 'B', 'A', 'B', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'B',
'B', 'A', 'B', 'A', 'B', 'B', 'A', 'B', 'A', 'A', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'B', 'A', 'A', 'B', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'A', 'A',
'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'A', 'A', 'B', 'A', 'B',
'A', 'A', 'A', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'A', 'B', 'B', 'B',
'A', 'A', 'B', 'B', 'A', 'B', 'B', 'A', 'B', 'B', 'B', 'B', 'B', 'A', 'B',
'A', 'B', 'A', 'A', 'A', 'B', 'A', 'B', 'B', 'A', 'A', 'A', 'A', 'A', 'A',
'B', 'B', 'B', 'B', 'B', 'A', 'B', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'A',
'A', 'A', 'A', 'A', 'A', 'B', 'B', 'A', 'B', 'B', 'A', 'B', 'A', 'B', 'A',
'B', 'B', 'A', 'A', 'B', 'A', 'A', 'B', 'A', 'A', 'A', 'A', 'B', 'A', 'A',
'A', 'B', 'A', 'B', 'A', 'B', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'B', 'B',
'A', 'B', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'A', 'A', 'A', 'B', 'A',
'A', 'B', 'A', 'A', 'A', 'A', 'B', 'A', 'B', 'A', 'B', 'A', 'A', 'B', 'B',
'B', 'B', 'A', 'B', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'A', 'B', 'A',
'A', 'A', 'B', 'A', 'B', 'A', 'A', 'B', 'B', 'A', 'B', 'B', 'B', 'B', 'B',
'B', 'B', 'A', 'B', 'B', 'B', 'A', 'B', 'A', 'A', 'B', 'B', 'B', 'A', 'B',
'B', 'A', 'A', 'B', 'A']
```