LAB 11 : Hidden Markov Model

Background

- HMM is a graphical model, which is generally used in predicting states (hidden, non-observable) using sequential data like weather, text, speech, etc.
- Central Concept: Future state depends solely on the current state, and depends on no other states. (Processes that obey this property are called *Markov Processes*.)
- The parameter learning task in HMMs is to find, given an output sequence or a set of such sequences, the best set of state transition and emission probabilities. The task is usually to derive MLE of the parameters of the HMM, given the set of output sequences.

Terms

- **State Transition Probabilities** determine the probabilities of each future state, given the current state.
- **Observation Symbol (Emission) Probabilities** determine the probabilities of observations emitted by the current state of the system.

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

Given the model λ and Sequence of visible/observable symbol V^T, we need to determine $P(V^T|\lambda)$ the probability that a particular sequence of observable states/symbol V^T that was generated from the model.

The forward variable is defined as the probability of the partial observation sequence v_1,v_2,..., v_t, when it terminates at the state i. Mathematically,

$$\alpha_t(i) = p(v_1, v_2, \ldots, v_t, q_t = i|\lambda)$$

$$\alpha_{t+1}(i) = b_j(v_{t+1}) \sum_{i=1}^{N} \alpha_t(i) a_{ij} \quad ,1 \leq j \leq N, 1 \leq t \leq T-1$$

$$\alpha_1(j) = \pi_j b_j(v_1) \quad ,1 \leq j \leq N$$

In [ ]:

```
data = pd.read_csv('data_python.csv') ## Read the data, change the path accordingly

V = data['Visible'].values # get only observable data


# Transition Probabilities
a = np.array(((0.54, 0.46), (0.49, 0.51)))                   # a: N x N = 2 x 2

# Emission Probabilities
b = np.array(((0.16, 0.26, 0.58),
              (0.25, 0.28, 0.47)))                           # b: N x M = 2 x 3

# Equal Probabilities for the initial distribution
initial_distribution = np.array((0.5, 0.5))                  # pi: 1 x N = 1 x 2 ; t


def forward(V, a, b, initial_distribution):
    T = V.shape[0]
    N = a.shape[0]
    alpha = np.zeros((T, N))                                 # T = num of obs, N = n
    alpha[0, :] = initial_distribution * b[:, V[0]]          # base case, t=0
    # print(alpha)
    # print(initial_distribution.shape)
    # print(b[:,V[0]])
    for t in range(1, T):
        for j in range(N):
            alpha[t, j] = np.dot(alpha[t-1, :], a[:,j]) * b[j, V[t]] # recursively
    # print(alpha)
    return alpha

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

In a similar way we can define the backward variable $\beta_t(i)$ as the probability of the partial observation sequence $v_{t+1}, v_{t+2}, \ldots, v_T$, given that the current state is i. Mathematically,

$$\beta_t(i) = p(v_{t+1}, v_{t+2}, \ldots, v_T | q_t = i, \lambda)$$

$$\beta_t(i) = \sum_{j=1}^{N} \beta_{t+1}(j) a_{ij} b_j(v_{t+1}) \qquad , 1 \leq i \leq N, 1 \leq t \leq T-1$$

$$\beta_T(i) = 1 \qquad , 1 \leq i \leq N$$

In [ ]:

```
def backward(V, a, b):
    T = V.shape[0]
    N = a.shape[0]
    beta = np.zeros((T, N))
    beta[T-1, :] = np.ones((N))          # set last row to ones
    for t in range(T-2, -1, -1):         # T-1 to 0 (non-incl), steps of -1
        for i in range(a.shape[0]):
            beta[t, i] = np.dot(beta[t+1]*b[:, V[t+1]], a[i, :])
    return beta

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

2. Learning Problem : Implementation of Baum Welch Algorithm

Given the model λ and Sequence of visible/observable symbol V^T, we need to determine how to adjust the model parameters λ, B, π in order to maximize $P(V^T|\lambda)$.

Let us define a new auxiliary variable $\xi_t(i,j)$ as the probability of being in state i at t=t and in state j at t=t+1. Then,

$$\begin{aligned}\xi_t(i,j) &= p(q_t = i, q_{t+1} = j|V^T, \lambda) \\ &= \frac{p(q_t = i, q_{t+1} = j, V^T|\lambda)}{p(V^T|\lambda)} \\ &= \frac{\alpha_t(i)a_{ij}\beta_{t+1}(j)b_j(v_{t+1})}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}\beta_{t+1}(j)b_j(v_{t+1})}\end{aligned}$$

Also, let us define $\gamma_t(i)$ as the a poteriori probability, that is the probability of being in state i at t=t, given the observation sequence and the model.

$$\begin{aligned}\gamma_t(i) &= p(q_t|V^T, \lambda) \\ &= \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N}\alpha_t(i)\beta_t(i)}\end{aligned}$$

Also,

$$\gamma_t(i) = \sum_{j=1}^{N}\xi_t(i,j) \quad ,1 \leq j \leq N$$

Our goal:

$$\begin{aligned}\bar{a}_{ij} &= \frac{\text{expected number of transitions from hidden state i to state j}}{\text{expected number of transition from hidden state i}} \\ \bar{b}_j(k) &= \frac{\text{expected number of times in hidden state j and observing v(k)}}{\text{expected number of times in hidden state j}}\end{aligned}$$

Re-estimations are:

$$\begin{aligned}\bar{a}_{ij} &= \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\gamma_t(i)} \quad ,1 \leq i \leq N, 1 \leq j \leq N \\ \bar{b}_j(k) &= \frac{\sum_{t=1}^{T}\gamma_t(j)1(v_t == k)}{\sum_{t=1}^{T}\gamma_t(j)} \quad ,1 \leq j \leq N, 1 \leq k \leq M\end{aligned}$$

In [ ]:
```
def baum_welch(V, a, b, initial_distribution, n_iter=100):
    T = V.shape[0]
    N = a.shape[0]
    M = b.shape[1]

    for it in range(n_iter):
        alpha = forward(V, a, b, initial_distribution)  # 500 x 2
        beta = backward(V, a, b)                        # 500 x 2

        xi = np.zeros((N, N, T-1))

        for t in range(T-1):
            d = np.dot(np.dot(alpha[t,:].T, a) * b[:,V[t + 1]].T, beta[t + 1, :]) #
```

```
            for i in range(N):
                n = alpha[t, i] * a[i,:] * b[:, V[t + 1]].T * beta[t + 1, :].T    #
                xi[i,:,t] = n / d

        gamma = np.sum(xi, axis=1)                                          # gamma is

        a = np.sum(xi, axis=2) / np.sum(gamma, axis=1).reshape((-1, 1)) # get re-es

        gamma = np.hstack((gamma, np.sum(xi[:, :, T - 2], axis=0).reshape((-1, 1)))

        d = np.sum(gamma, axis=1)
        for l in range(M):
            b[:, l] = np.sum(gamma[:, V == l], axis=1)

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


    return (a, b)
```

In [ ]:
```
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)
print("a = ",a)
print("b = ",b)
```

```
a =  [[0.53816345 0.46183655]
 [0.48664443 0.51335557]]
b =  [[0.16277513 0.26258073 0.57464414]
 [0.2514996  0.27780971 0.47069069]]
```

3. Decoding Problem : Implementation of Viterbi Algorithm

Given the model λ and Sequence of visible/observable symbol V^T, we need to determine what is the most likely state sequence in the model that generated the observations.

Let us define an auxiliary variable $\delta_t(i)$, which is the highest probability that partial observation sequence and state sequence up to t=t can have, when the current state is i. Then,

$$\delta_t(i) = \max_{q_1,q_2,\ldots,q_{t-1}} p(q_1, q_2, \ldots, q_{t-1}, q_t = i, v_1, v_2, \ldots, v_{t-1}|\lambda)$$

$$\delta_{t+1}(i) = b_j(v_{t+1})\Big[\max_{1\le i\le N} \delta_t(i)a_{ij}\Big]$$

$$\delta_1(j) = \pi_j b_j(v_1)$$

In [ ]:
```
def viterbi(V, a, b, initial_distribution):
```

```
    T = V.shape[0]
    N = a.shape[0]
    delta = np.zeros((T, N))

    delta[0, :] = np.log(initial_distribution * b[:, V[0]])
    prev = np.zeros((T - 1, N))

    for t in range(1, T):
        for i in range(N):
            p = delta[t - 1] + np.log(a[:, i]) + np.log(b[i, V[t]]) # log of the fo
            prev[t - 1, i] = np.argmax(p)                          # find the stat
            delta[t, i] = np.max(p)

    seq = np.zeros(T)
    last = np.argmax(delta[T - 1, :])       # get argmax of probs at T-1
    seq[0] = last

    ind = 1
    for i in range(T - 2, -1, -1):
        seq[ind] = prev[i, int(last)]
        last = prev[i, int(last)]
        ind += 1

    seq = np.flip(seq, axis=0)              # flip since backtracking

    result = []
    for ele in seq:
        if ele == 0:
            result.append("A")
        else:
            result.append("B")
    return result
```

In [ ]:

```
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)

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

print(result)
```

```
['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==0.2.7
```

```
Requirement already satisfied: hmmlearn==0.2.7 in c:\users\bsidd\appdata\local\prog
rams\python\python39\lib\site-packages (0.2.7)
Requirement already satisfied: scipy>=0.19 in c:\users\bsidd\appdata\local\programs
\python\python39\lib\site-packages (from hmmlearn==0.2.7) (1.7.1)
Requirement already satisfied: scikit-learn>=0.16 in c:\users\bsidd\appdata\local\p
rograms\python\python39\lib\site-packages (from hmmlearn==0.2.7) (1.0.2)
Requirement already satisfied: numpy>=1.10 in c:\users\bsidd\appdata\local\programs
\python\python39\lib\site-packages (from hmmlearn==0.2.7) (1.21.2)
Requirement already satisfied: joblib>=0.11 in c:\users\bsidd\appdata\local\program
s\python\python39\lib\site-packages (from scikit-learn>=0.16->hmmlearn==0.2.7) (1.
1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\bsidd\appdata\local
\programs\python\python39\lib\site-packages (from scikit-learn>=0.16->hmmlearn==0.
2.7) (3.1.0)
Note: you may need to restart the kernel to use updated packages.
```

```
WARNING: You are using pip version 22.0.4; however, version 22.3 is available.
You should consider upgrading via the 'c:\Users\bsidd\AppData\Local\Programs\Python
\Python39\python.exe -m pip install --upgrade pip' command.
```

In [ ]:
```
## Write your code here
from hmmlearn import hmm
import numpy as np
import math
```

```
data = pd.read_csv('data_python.csv')
V = data['Visible'].values

model = hmm.MultinomialHMM(n_components=2)
model.startprob_ = np.array([0.5, 0.5])
model.transmat_ = np.array([[0.5, 0.5],
                            [0.5, 0.5]])
model.emissionprob_ = np.array([[0.11111111, 0.33333333, 0.55555556],
                                [0.16666667, 0.33333333 ,0.5]])

logprob, sequence = model.decode((np.array(V).reshape(-1,1)).transpose())
out = []
```

In [ ]:

```
for i in sequence:
    if i == 1:
        i = "B"
    else:
        i = "A"
    out.append(i)
print(out)
```

```
['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']
```

In [ ]: