CS640 Homework 4: Hidden Markov Model

In this assignment, you will solve a Hidden Markov Model problem with code.

Do **not** include any extra outputs in your final answer. If you use print() function for debugging, please run your code without them again before submitting.

Collaboration

You must answer written questions independently.

Instructions

General Instructions

In an ipython notebook, to run code in a cell or to render Markdown+LaTeX press `Ctrl+Enter` or `[>|]` (like "play") button above. To edit any code or text cell (double) click on its content. To change cell type, choose "Markdown" or "Code" in the drop-down menu above.

Most of the written questions are followed up a cell for you enter your answers. Please enter your answers in a new line below the **Answer** mark. If you do not see such cell, please insert one by yourself. Your answers and the questions should **not** be in the same cell.

Instructions on Math

Some questions require you to enter math expressions. To enter your solutions, put down your derivations into the corresponding cells below using LaTeX. Show all steps when proving statements. If you are not familiar with LaTeX, you should look at some tutorials and at the examples listed below between \$..\$. The OEIS website can also be helpful.

Alternatively, you can scan your work from paper and insert the image(s) in a text cell.

Submission

Once you are ready, save the note book as PDF file (File -> Print -> Save as PDF) and submit via Gradescope.

Please select pages to match the questions on Gradescope. **You may be subject to a 5% penalty if you do not do so**.

Problem Setup

Consider an HMM with the following properties (unknown values are marked with "?").

States: $S = \{S_1, S_2, S_3\}$

Initial state probability distribution: $\pi = \{\pi_1 = 0.3, \pi_2 = 0.5, \pi_3 = ?\}$

Transition probability matrix A, where each entry a_{ij} is the probability of moving from state S_i to state S_j:

$$\begin{bmatrix} ? & 0.5 & 0.4 \\ ? & 0.3 & 0.6 \\ ? & 0.1 & 0.7 \end{bmatrix}$$

Symbols probability matrix B, where each entry $b_{jk} = P[V_k \mid S_j]$ is the probability of yielding V_k in state S_j:

$$\begin{bmatrix} 0.5 & 0.4 & ? \\ ? & 0.3 & 0.1 \\ 0.8 & ? & 0.1 \end{bmatrix}$$

Let $\lambda = (A, B, \pi)$.

Suppose we observe a sequence $O = V_1 V_3 V_2 V_1$

Q1

Complete the model by filling the unknown values in the following code block and run the cell. Make sure the outputs are all **True**.

```
import numpy as np

N, M = 3, 3
V = [None, 0, 1, 2] # V[0] = None to align the indices

# fill in the blank values for the following three arrays
pi = np.array([0.3, 0.5, 0.2])
```

```
A = np.array([[0.1, 0.5, 0.4], [0.1, 0.3, 0.6], [0.2, 0.1, 0.7]])
B = np.array([[0.5, 0.4, 0.1], [0.6, 0.3, 0.1], [0.8, 0.1, 0.1]])

O = [V[1], V[3], V[2], V[1]]

print(np.linalg.norm(pi.sum() - 1) < 1e-9)
print(np.linalg.norm(A.sum(axis = 1) - 1) < 1e-9)
print(np.linalg.norm(B.sum(axis = 1) - 1) < 1e-9)
```

```
True
True
True
```

Q2

What is $P(O|\lambda)$? Answer the question by finish implementing the **Forward Procedure** and the **Backward Procedure** below. Make sure you run the cell and the final answers from the two procedures are identical.

Forward Procedure

```
T = len(O)
alpha = np.zeros((T, N))

# initialization
for i in range(N): # complete this loop
    ################### start of your code ###################
    alpha[0, i] = pi[i] * B[i, O[0]]
    #################### end of your code ####################

# inductive steps
for t in range(1, T): # complete this loop
    ################### start of your code ###################
    '''
    for j in range(N):
        alpha[t, j] = alpha[t - 1] @ A[:, j] * B[j, O[t]]
    '''
    # one-liner
    alpha[t] = alpha[t - 1] @ A * B[:, O[t]]
    #################### end of your code ####################

print(alpha)

# final answer
################### start of your code ###################
answer = 0 # this variable should store your final answer
answer = alpha[-1].sum()
#################### end of your code ####################
print("P(O | lambda) = " + str(answer))
```

```
[[0.15       0.3        0.16      ]
 [0.0077     0.0181     0.0352    ]
 [0.003848   0.00384    0.003858  ]
 [0.0007702  0.00207708 0.00523504]]
P(O | lambda) = 0.00808232
```

Backward Procedure

```
T = len(O)
beta = np.zeros((T, N))

# initialization
for i in range(N): # complete this loop
    ################### start of your code ###################
    beta[-1, i] = 1
    #################### end of your code ####################

# inductive steps
for t in range(T - 2, -1, -1): # complete this loop
    ################### start of your code ###################
    '''
    for i in range(N):
        beta[t, i] = A[i, :] @ (B[:, O[t + 1]] * beta[t + 1, :])

        for j in range(N):
            beta[t, i] += A[i, j] * B[j, O[t + 1]] * beta[t + 1, j]
    '''
    # one-liner
    beta[t] = A @ (B[:, O[t + 1]] * beta[t + 1])
    #################### end of your code ####################
print(beta)

# final answer
# complete this part
################### start of your code ###################
answer = 0 # this variable should store your final answer
'''
for j in range(N):
    answer += pi[j] * B[j, O[0]] * beta[0, j]
'''
# one-liner
answer = pi @ (B[:, O[0]] * beta[0])
#################### end of your code ####################
print("P(O | lambda) = " + str(answer))
```

```
[[0.013328 0.013156 0.013352]
 [0.1621   0.1339   0.1253  ]
 [0.67     0.71     0.72    ]
 [1.       1.       1.      ]]
P(O | lambda) = 0.00808232
```

Q3

Suppose we observe a sequence $O = V_1 V_3 V_2 V_1$, what is the most likely state sequence? Answer the question by finish implementing the Viterbi algorithm in the following code block.

```
T = len(O)
delta = np.zeros((T, N)) # stores the optimal probabilities
psi = np.zeros((T, N), dtype = np.int8) # stores the corresponding sources

# initialization
for i in range(N): # complete this loop
    ################### start of your code ###################
    delta[0, i] = pi[i] * B[i, O[0]]
    #################### end of your code ####################

# inductive steps
for t in range(1, T): # complete this loop
    ################### start of your code ###################
    for j in range(N):
        delta[t, j] = np.max(delta[t - 1] * A[:, j]) * B[j, O[t]]
        psi[t, j] = np.argmax(delta[t - 1] * A[:, j])
    #################### end of your code ####################

print(delta)
print(psi)

# backtrack for the optimal path
################### start of your code ###################
optimal_path = [] # this variable should store your final answer
optimal_path = [np.argmax(delta[-1])]
for t in range(T - 1, 0, -1):
    optimal_path.append(psi[t, optimal_path[-1]])
optimal_path.reverse()
################### start of your code ###################
print("Optimal path = " + str(optimal_path))
```

```
[[1.500e-01 3.000e-01 1.600e-01]
 [3.200e-03 9.000e-03 1.800e-02]
 [1.440e-03 8.100e-04 1.260e-03]
 [1.260e-04 4.320e-04 7.056e-04]]
[[0 0 0]
 [2 1 1]
 [2 1 2]
 [2 0 2]]
Optimal path = [1, 2, 2, 2]
```