Abelardo Riojas
STA 5635
Prof. Barbu Spring 2022

Homework 9 Report

Code for Viterbi, Forward, and Backward algorithms from
http://www.adeveloperdiary.com/data-science/machine-learning/implement-viterbi-algorithm-in-hidden-markov-model-using-python-and-r/

Baum Welch algorithm uses the HMM package from
https://github.com/maximtrp/mchmm

1. Optimal State Sequence:

```
Observation sequence:   O =  [1 2 5 5 1 3 6 3 2 3 6 6 1 3 3 2 3 4 3 1 2 2 4 2 5 6 3 2 1 5 5 2 2 5 3 4 1
 6 6 1 5 5 2 6 2 1 1 3 6 1 4 3 1 2 2 6 3 3 1 2 1 6 4 3 2 3 1 1 5 1 2 4 1 3
 2 6 1 6 6 1 6 6 5 4 6 5 2 3 3 1 5 6 2 6 3 6 6 3 3 3 6 6 6 6 5 1 2 6 6 6 2
 6 6 6 3 5 2 6 3 6 2 6 6 1 3 1 1 4 5 6 2 3 1 3 6 2 6 6 3 5 4 5 6 3 3 2 6 2
 6 3 3 2 5 3 6 5 3 5 2 1 3 6 2 5 5 1 6 6 6 6 1 5 2 4 6 1 2 4 3 1 3 3 1 2 2
 6 6 6 5 4 3 3 3 6 3 2 2 3 1 4 5 5 2 5 4 3 1 6 1 2 6 4 2 6 6 3 1 2 6 5 5 6
 6 6 6 6 6 6 3 6 6 6 6 6 6 6 6 6 6 6 6 4 4 6 6 2 2 6 6 3 6 6 2 5 6 3 6 4 4
 6 6 6 6 2 4 6 2 6 6 6 6 6 6 4 3]
Optimal state sequence: S =  [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 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 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 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 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

2. Alphas: a^1_{128} = 9.637945880716333e-98 a^2_{128} = 4.674122544571388e-98
3. Betas: b^1_{128} = 4.858102864508214e-102, b^2_{128} = 3.7506263681542445e-102

1. Baum Welch Algorithm:
   (Ran for 1528 iterations)

Transition matrix from BW:
```
        [[0.41253564 0.58746436]
       [0.60551091 0.39448909]]
```
Initial pi from BW:
```
        [0.47226973 0.52773027]
```
Emission probs from BW:
```
        [0.30746943 0.17579383 0.19160409 0.15348056 0.13082977 0.04082233]
        [0.08583591 0.22827831 0.18542988 0.23740927 0.11738693 0.14565970]
```

Code below:

```python
from csv import reader
import numpy as np
import pandas as pd
from tqdm import tqdm
file = open('hmm_pb1.csv')
row = list(reader(file))[0]

O = np.array(row).astype(int)-1

def viterbi_log(A, C, B, O):
    """Viterbi algorithm (log variant) for solving the uncovering problem

    Notebook: C5/C5S3_Viterbi.ipynb

    Args:
        A (np.ndarray): State transition probability matrix of dimension I x I
        C (np.ndarray): Initial state distribution  of dimension I
        B (np.ndarray): Output probability matrix of dimension I x K
        O (np.ndarray): Observation sequence of length N

    Returns:
        S_opt (np.ndarray): Optimal state sequence of length N
        D_log (np.ndarray): Accumulated log probability matrix
        E (np.ndarray): Backtracking matrix
    """
    I = A.shape[0]    # Number of states
    N = len(O)  # Length of observation sequence
    tiny = np.finfo(0.).tiny
    A_log = np.log(A + tiny)
    C_log = np.log(C + tiny)
    B_log = np.log(B + tiny)

    # Initialize D and E matrices
    D_log = np.zeros((I, N))
    E = np.zeros((I, N-1)).astype(np.int32)
    D_log[:, 0] = C_log + B_log[:, O[0]]

    # Compute D and E in a nested loop
    for n in range(1, N):
        for i in range(I):
            temp_sum = A_log[:, i] + D_log[:, n-1]
            D_log[i, n] = np.max(temp_sum) + B_log[i, O[n]]
            E[i, n-1] = np.argmax(temp_sum)
```

```
    # Backtracking
    S_opt = np.zeros(N).astype(np.int32)
    S_opt[-1] = np.argmax(D_log[:, -1])
    for n in range(N-2, -1, -1):
        S_opt[n] = E[int(S_opt[n+1]), n]

    return S_opt, D_log, E

# Define model parameters
A = np.array([
    [.95, .05],
    [.05, .95]
])

C = np.array([0.5, 0.5])

B = np.array([[1/6, 1/6, 1/6, 1/6, 1/6, 1/6],
              [0.1, 0.1, 0.1, 0.1, 0.1, 0.5]])


# Apply Viterbi algorithm (log variant)
S_opt, D_log, E = viterbi_log(A, C, B, O)

print('Observation sequence:   O = ', O+1)
print('Optimal state sequence: S = ', S_opt+1)
np.set_printoptions(formatter={'float': "{: 7.2f}".format})
print('D_log =', D_log, sep='\n')

def forward(V, a, b, initial_distribution):
    alpha = np.zeros((V.shape[0], a.shape[0]))
    alpha[0, :] = initial_distribution * b[:, V[0]]

    for t in range(1, V.shape[0]):
        for j in range(a.shape[0]):
            # Matrix Computation Steps
            #                  ((1x2) . (1x2))      *     (1)
            #                        (1)            *     (1)
            alpha[t, j] = alpha[t - 1] @ a[:, j] * b[j, V[t]]

    return alpha

alpha = forward(O, A, B, C)
```

```
def backward(V, a, b):
    beta = np.zeros((V.shape[0], a.shape[0]))

    # setting beta(T) = 1
    beta[V.shape[0] - 1] = np.ones((a.shape[0]))

    # Loop in backward way from T-1 to
    # Due to python indexing the actual loop will be T-2 to 0
    for t in range(V.shape[0] - 2, -1, -1):
        for j in range(a.shape[0]):
            beta[t, j] = (beta[t + 1] * b[:, V[t + 1]]) @ a[j, :]

    return beta
beta = backward(O, A, B)

**BAUM WELCH CODE HERE**

import mchmm._hmm as hmm
from csv import reader
import math

data = open('hmm_pb2.csv')

data = list(reader(data))[0]




string = ' '.join([str(item) for item in data])
string = string.replace(' ', '')

obs_seq = string


for i in range(3, 12):
      thres = math.pow(10, -1*i)
      print(thres)
      a = hmm.HiddenMarkovModel().from_baum_welch(obs_seq, states=['fair',
'loaded'], thres=thres)

      print(f'Transition matrix from BW:\n\t {a.tp}')
      print(f'Initial pi from BW:\n\t {a.pi}')
      print(f'Emission probs from BW:\n\t {a.ep}')
```