Machine Learning
Homework #6 Supplementary

Daniel Ho Kwan Leung
104360098
Year 3, CSIE

Question 4:

The following code was uploaded to GitHub:

https://github.com/DHKLeung/NTUT_Machine_Learning/blob/master/HW6_Q4_refined.ipynb

This refined version changes the epsilon from 10e-6 to 10e-12 and the training iterations from 100 to 10000.

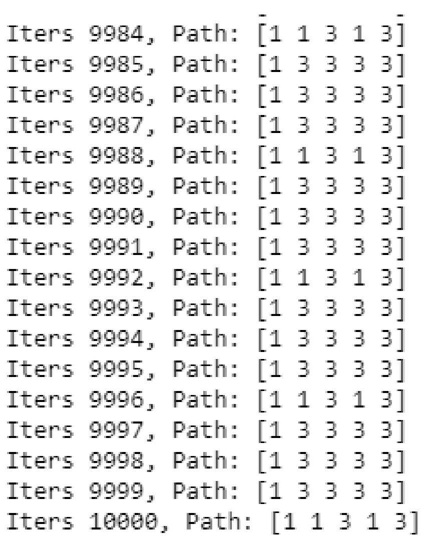
```python
import numpy as np
def evaluation_forward(A, B, PI, O, alpha):
    alpha[:, 0] = np.multiply(PI, B[:, O[0]])
    for t in range(1, O.shape[0]):
        for j in range(A.shape[0]):
            alpha[j, t] = np.sum(np.multiply(alpha[:, t - 1], A[:, j])) * B[j, O[
t]]
    return alpha
def evaluation_backward(A, B, PI, O, beta):
    beta[:, -1] = np.ones((A.shape[0]), dtype=np.float64)
    for t in range(O.shape[0] - 2, -1, -1):
        for i in range(A.shape[0]):
            beta[i, t] = np.sum(np.multiply(np.multiply(A[i, :], B[:, O[t]]), bet
a[:, t + 1]))
    return beta
def decode(A, B, PI, O, delta):
    delta[:, 0] = np.multiply(PI, B[:, O[0]])
    for t in range(1, O.shape[0]):
        for j in range(A.shape[0]):
            delta[j, t] = np.amax(np.multiply(delta[:, t - 1], A[:, j])) * B[j, O
[t]]
    path = np.argmax(delta, axis=0) + 1
    return delta, path
def learn(A, B, PI, O, V, alpha, beta, XI, gamma, epsilon=10e-12):
    alpha = evaluation_forward(A, B, PI, O, alpha)
    beta = evaluation_backward(A, B, PI, O, beta)

    #Compute XI#
    p = np.sum(np.multiply(alpha, beta), axis=0)
    for t in range(O.shape[0] - 1):
        for i in range(A.shape[0]):
            for j in range(A.shape[0]):
                XI[i, j, t] = (alpha[i, t] * beta[j, t + 1] * A[i, j] * B[j, O[t
+ 1]]) / (p[t] + epsilon)

    #Compute gamma#
    for t in range(O.shape[0]):
        for i in range(A.shape[0]):
            gamma[i, t] = np.sum(XI[i, :, t])

    #Compute new A#
    for i in range(A.shape[0]):
        for j in range(B.shape[0]):
            A[i, j] = np.sum(XI[i, j, :]) / (np.sum(gamma[i]) + epsilon)
    #Compute new PI#
    for i in range(A.shape[0]):
        PI[i] = gamma[i, 0]

    #Compute new B#
    for j in range(A.shape[0]):
        for k in range(V.shape[0]):
            B[j, k] = np.sum(np.multiply(gamma[j], (O == V[k]).astype(np.float64
))) / (np.sum(gamma[j]) + epsilon)
    return A, B, PI, gamma, XI, alpha, beta
```

```
A = np.array([ #1st Rank = From State, 2nd Rank = To State
    [1 / 3, 1 / 3, 1 / 3],
    [1 / 3, 1 / 3, 1 / 3],
    [1 / 3, 1 / 3, 1 / 3]
], dtype=np.float64)
PI = np.array([1/ 3, 1 / 3, 1 / 3], dtype=np.float64) #1st Rank = Init State
B = np.array([ #1st Rankg = State, 2nd Rank = Output Value
    [4 / 6, 2 / 6],
    [2 / 6, 4 / 6],
    [3 / 6, 3 / 6]
], dtype=np.float64)
O = np.array([0, 0, 1, 0, 1], dtype=np.int32) #0 = red, 1 = blue
V = np.array([0, 1], dtype=np.int32) #0 = red, 1 = blue
alpha = np.zeros((B.shape[0], O.shape[0]), dtype=np.float64) #1st Rank = State, 2
nd Rank = Time
beta = np.zeros((B.shape[0], O.shape[0]), dtype=np.float64) #1st Rank = State, 2n
d Rank = Time
delta = np.zeros((B.shape[0], O.shape[0]), dtype=np.float64) #1st Rank = State, 2
nd Rank = Time
gamma = np.zeros((B.shape[0], O.shape[0]), dtype=np.float64) #1st Rank = State, 2
nd Rank = Time
XI = np.zeros((B.shape[0], B.shape[0], O.shape[0]), dtype=np.float64) #1st Rank =
 From State, 2nd Rank = To State, 3rd Rank = Time
```

```
records = list()
records.append((np.copy(A), np.copy(B), np.copy(PI)))
for i in range(10000):
    A, B, PI, gamma, XI, alpha, beta = learn(A, B, PI, O, V, alpha, beta, XI, gam
ma)
    delta, path = decode(A, B, PI, O, delta)
    records.append((np.copy(A), np.copy(B), np.copy(PI)))
    print('Iters {}, Path: {}'.format(i + 1, path))
```

```
for i in range(9997, 10001):
    print('============ Iter {} ============'.format(i))
    print('A:\n{}'.format(records[i][0]))
    print('B:\n{}'.format(records[i][1]))
    print('PI:\n{}'.format(records[i][2]))
```

The last 17 iterations,



```
Iters 9984, Path: [1 1 3 1 3]
Iters 9985, Path: [1 3 3 3 3]
Iters 9986, Path: [1 3 3 3 3]
Iters 9987, Path: [1 3 3 3 3]
Iters 9988, Path: [1 1 3 1 3]
Iters 9989, Path: [1 3 3 3 3]
Iters 9990, Path: [1 3 3 3 3]
Iters 9991, Path: [1 3 3 3 3]
Iters 9992, Path: [1 1 3 1 3]
Iters 9993, Path: [1 3 3 3 3]
Iters 9994, Path: [1 3 3 3 3]
Iters 9995, Path: [1 3 3 3 3]
Iters 9996, Path: [1 1 3 1 3]
Iters 9997, Path: [1 3 3 3 3]
Iters 9998, Path: [1 3 3 3 3]
Iters 9999, Path: [1 3 3 3 3]
Iters 10000, Path: [1 1 3 1 3]
```

We can see that the HMM converges to the path [1 3 3 3 3] mainly and may jump to the path [1 1 3 1 3] occasionally. This may be due to the short observation sequence and the initial distributions of the three urns.