Assignment2

February 12, 2020

1 Assigment 2

1.0.1 Imports

```
In [1]: import numpy as np
```

1.1 Part A

Unobserved variables: Rain
Observable variables: Umbrella
Dynamic model:

$$P(X_t|X_{t-1}) = \begin{bmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{bmatrix}$$

Observation model:

$$P(E_t|X_t) = \begin{bmatrix} 0.9 & 0.1 \\ 0.2 & 0.8 \end{bmatrix}$$

Assumptions: - The Markov Assumption of the first degree is a somewhat reasonable assumption in the given circumstances. - The Sensor markov assumption of the usage of a umbrella is only dependent on the presence of rain is reasonable.

1.2 Part B

```
In [2]: def forward(sensor_model, transition_model, message):
            forward_return = sensor_model * transition_model.getT() * message  # Equation 15.1
            return forward_return/forward_return.sum()  # Normalization

In [3]: def forward_total(evidence, prior):
            '''

            :param evidence:
            :param prior:
            :return:
            '''

            # Models
            transition_model = np.matrix([[0.7, 0.3], [0.3, 0.7]])
            sensormodel_true = np.matrix([[0.9, 0.0], [0.0, 0.2]])
```

```
    sensormodel_false = np.matrix([[0.1, 0.0], [0.0, 0.8]])

    forward_message = [np.matrix([[0.0], [0.0]]) for i in range((len(evidence) + 1))]
    backward_message = np.matrix([[1.0], [1.0]])


    forward_message[0] = prior
    smoothed_stepvector = [np.matrix([[0.0],[0.0]]) for i in range(len(evidence))]
    t = len(evidence)

    for i in range(1, t + 1):
        if evidence[i-1]:
            forward_message[i] = forward(sensormodel_true, transition_model, forward_me
        else:
            forward_message[i] = forward(sensormodel_false, transition_model, forward_m

    return forward_message
```

```
In [17]: OG_msg = np.matrix('0.5; 0.5')
         evidence = [True, True, False, True, True]
         fm = forward_total(evidence, OG_msg)
         for n, i in enumerate(fm):
             print(str(n) + ':', str(i)[1:-1], '\n')
```

```
0: [0.5]
 [0.5]

1: [0.81818182]
 [0.18181818]

2: [0.88335704]
 [0.11664296]

3: [0.19066794]
 [0.80933206]

4: [0.730794]
 [0.269206]

5: [0.86733889]
 [0.13266111]
```

As we can see, $P(X_2|e_{1:2}) = 0.883$,. which is consistent with the given values in the assignment. To explicitly answer the assignment question: The probability of rain on day 5 is 0.867.

1.3 Part C

```
In [18]: def backward(sensor_model, transition_model, message):
             backward_return = transition_model * sensor_model * message # Equation 15.13
             return backward_return/backward_return.sum()  # Normalization

In [19]: def forward_backward(evidence, prior):
             '''
             Implementation of Forward-Backward algorithm
             :param evidence: The visible states given as an iterable
             :param prior: The initial probability distribution
             :return: The smoothed bacward probabilities
             '''

             # Models
             transition_model = np.matrix([[0.7, 0.3], [0.3, 0.7]])
             sensormodel_true = np.matrix([[0.9, 0.0], [0.0, 0.2]])
             sensormodel_false = np.matrix([[0.1, 0.0], [0.0, 0.8]])

             forward_message = [np.matrix([[0.0], [0.0]]) for i in range((len(evidence) + 1))]
             backward_message = np.matrix([[1.0], [1.0]])


             forward_message[0] = prior
             smoothed_stepvector = [np.matrix([[0.0], [0.0]]) for i in range(len(evidence))]
             t = len(evidence)

             for i in range(1, t + 1):
                 if evidence[i-1]:
                     forward_message[i] = forward(sensormodel_true, transition_model, forward_r
                 else:
                     forward_message[i] = forward(sensormodel_false, transition_model, forward_

             for j in range(t - 1, -1, -1):
                 k = forward_message[j].getA()
                 l = backward_message.getA()

                 step = np.asmatrix(forward_message[j+1].getA() * backward_message.getA())

                 smoothed_stepvector[j] = step/step.sum()

                 if evidence[j]:
                     backward_message = backward(sensormodel_true, transition_model, backward_r
                 else:
                     backward_message = backward(sensormodel_false, transition_model, backward_
                 # print(backward_message)
             return smoothed_stepvector
```

Verification:

In [22]:
```
evidence = [True, True]
fb = forward_backward(evidence, OG_msg)
for k, i in enumerate(fb):
    print("b_(" + str(len(evidence) - k) + ':' + str(len(evidence)) + ')', ':', str(i
```

```
b_(2:2) : [0.88335704]
 [0.11664296]

b_(1:2) : [0.88335704]
 [0.11664296]
```

In [23]:
```
evidence = [True, True, False, True, True]
fb = forward_backward(evidence, OG_msg)
for k, i in enumerate(fb):
    print("b_(" + str(len(evidence) - k) + ':' + str(len(evidence)) + ')', ':', str(i
```

```
b_(5:5) : [0.86733889]
 [0.13266111]

b_(4:5) : [0.82041905]
 [0.17958095]

b_(3:5) : [0.30748358]
 [0.69251642]

b_(2:5) : [0.82041905]
 [0.17958095]

b_(1:5) : [0.86733889]
 [0.13266111]
```

New function with printing of all backward messages:

In [27]:
```
def forward_backward(evidence, prior):
    '''
    Implementation of Forward-Backward algorithm
    :param evidence: The visible states given as an iterable
    :param prior: The initial probability distribution
    :return: The smoothed bacward probabilities
    '''

    # Models
    transition_model = np.matrix([[0.7, 0.3], [0.3, 0.7]])
    sensormodel_true = np.matrix([[0.9, 0.0], [0.0, 0.2]])
    sensormodel_false = np.matrix([[0.1, 0.0], [0.0, 0.8]])
```

```python
        # Initial messages
        forward_message = [np.matrix([[0.0], [0.0]]) for i in range((len(evidence) + 1))]
        backward_message = np.matrix([[1.0], [1.0]])


        # Following the textbook's Pseudocode
        forward_message[0] = prior
        smoothed_stepvector = [np.matrix([[0.0], [0.0]]) for i in range(len(evidence))]
        t = len(evidence)

        for i in range(1, t + 1):
            if evidence[i-1]:
                forward_message[i] = forward(sensormodel_true, transition_model, forward_m
            else:
                forward_message[i] = forward(sensormodel_false, transition_model, forward_

        for j in range(t - 1, -1, -1):
            k = forward_message[j].getA()
            l = backward_message.getA()

            step = np.asmatrix(forward_message[j+1].getA() * backward_message.getA())

            smoothed_stepvector[j] = step/step.sum()

            if evidence[j]:
                backward_message = backward(sensormodel_true, transition_model, backward_m
            else:
                backward_message = backward(sensormodel_false, transition_model, backward_
            print(str(backward_message)[1:-1])

        return smoothed_stepvector
```

```
In [29]: x = forward_backward(evidence, OG_msg)
```

```
[0.62727273]
 [0.37272727]
[0.65334282]
 [0.34665718]
[0.37626718]
 [0.62373282]
[0.5923176]
 [0.4076824]
[0.64693556]
 [0.35306444]
```

These are the bacward messages for each of the five steps.