EQ2341 - Assignment 3

Matay Mayrany - mayrany@kth.se

May 2022

1 Forward Algorithm

The forward algorithm helps us reduce the computational complexity when evaluating the probability of a set of observations given a Hidden Markov Model. This is done by storing an alpha variable that helps us not repeat computations that are already done. The instructions in the book were followed to implement the code needed for this algorithm, different parts of it can be seen below.

One of the preliminary steps that had to be done was to implement the prob function in the GaussD function. This is because only the rand function was implemented in the first assignment, which "returns random vectors drawn from a single GaussD object" but we need to determine the probability of the given observations based on each feature vector. The implementation for that can be seen below.

```
def prob(self, observation):
    randomVariable = mvn(self.means, self.cov)
    return randomVariable.pdf(observation)
```

Moreover, the prob function in the HMM class was implemented to be able to get the **Px** matrix that will be used in the forward algorithm. Code fore that can be seen below. Note that some invalid indentations were made to make the screenshot fit better.

```
def prob(self, observations, shouldScale):
    numberOfObservations = len(observations)
    numberOfFeatures = len(self.outputDistr)
    probOfObservations = np.zeros((numberOfFeatures, numberOfObservations))
    scaledProbOfObservations = np.zeros((numberOfFeatures, numberOfObservations))
    for feature in range(numberOfFeatures):
        for observation in range(numberOfObservations):
            probOfObservations[feature, observation] =
                self.outputDistr[feature].prob(observations[observation])
    if shouldScale:
        for feature in range(numberOfFeatures):
            for observation in range(numberOfObservations):
                scaledProbOfObservations[feature, observation] =
                    probOfObservations[feature,observation] / np.amax(probOfObservations[:,observation])
    else:
        scaledProbOfObservations = probOfObservations

    return scaledProbOfObservations
```

Now onto the the forward algorithm implementation in the MarkovChain file. The full code can be seen in the files attached, but below we can see snippets of it corresponding to the three phases of the algorithm as describe by the text book.

1.1 Initialization Step

Initialization step formulas from the text book:

Initialization: At $t = 1$ we easily obtain

$$\alpha_{j,1}^{temp} = P\left[\boldsymbol{X}_1 = \boldsymbol{x}_1, S_1 = j \mid \lambda\right] = q_j b_j(\boldsymbol{x}_1), \quad j = 1 \ldots N \tag{5.42}$$

$$c_1 = \sum_{k=1}^{N} \alpha_{k,1}^{temp} \tag{5.43}$$

$$\hat{\alpha}_{j,1} = \alpha_{j,1}^{temp} / c_1, \quad j = 1 \ldots N \tag{5.44}$$

Implementation of initialization step:

```
#Initialization step
temp[0] = self.q.dot(scaledProbOfObservations[:, 0])
c[0] = np.sum(temp)
alpha[:, 0] = temp / c[0]
```

1.2 Forward Step

Forward step formulas from the text book:

$$\alpha_{j,t}^{temp} = b_j(\boldsymbol{x}_t)\left(\sum_{i=1}^{N}\hat{\alpha}_{i,t-1}a_{ij}\right), \quad j = 1\ldots N \tag{5.50}$$

$$c_t = \sum_{k=1}^{N}\alpha_{k,t}^{temp} \tag{5.51}$$

$$\hat{\alpha}_{j,t} = \alpha_{j,t}^{temp}/c_t, \quad j = 1\ldots N \tag{5.52}$$

Implementation of forward step:

```
#Forward step
for t in range(1, T):
    for j in range(J):
        temp[j] = alpha[:, t - 1].dot(self.A[:, j]) * scaledProbOfObservations[j, t]
    c[t] = np.sum(temp)
    alpha[:,t] = temp/c[t]
```

1.3 Termination Step - Finite Chains Only

Termination step formulas from the text book:

$$\begin{aligned} c_{T+1} &= P\left[S_{T+1} = N+1 \mid \boldsymbol{x}_1 \ldots \boldsymbol{x}_T, \lambda\right] \\ &= \sum_{k=1}^{N} P\left[S_T = k \cap S_{T+1} = N+1 \mid \boldsymbol{x}_1 \ldots \boldsymbol{x}_T, \lambda\right] \\ &= \sum_{k=1}^{N} \hat{\alpha}_{k,T}a_{k,N+1} \end{aligned}$$

Implementation of termination step:

```
#Termination step for finite chains
if self.is_finite:
    c = np.append(c, [alpha[:, alpha.shape[1] - 1].dot(self.A[:, self.A.shape[1] - 1])])
```

2 Tests and Verification of Implementation

In order to verify the implementation the examples given in the assignment descriptions were created an the results from tests now match those described in the textbook. Code for the testing and screenshots of the results can be seen below.

2.1 Finite Duration Chain

Code for testing the HMM with a finite chain as described in the assignment:

```
from PattRecClasses import GaussD, HMM, MarkovChain
import numpy as np

#Test Forward algorithm FINITE CHAIN
finiteDurationMC = MarkovChain(np.array([1, 0]), np.array([[0.9, 0.1, 0], [0, 0.9, 0.1]]))

g1 = GaussD( means=[0], stdevs=[1] )   # Distribution for state = 1
g2 = GaussD( means=[3], stdevs=[2] )   # Distribution for state = 2
finiteDurationHMM  = HMM(finiteDurationMC, [g1, g2])
observations = np.array([-0.2, 2.6, 1.3])
scaledProbOfObservations = finiteDurationHMM.prob(observations)
alpha, c = finiteDurationMC.forward(scaledProbOfObservations)
print("Values for finite duration HMM:")
print("alfaHat:")
print(np.around(alpha, 4))
print("c: ")
print(np.around(c, 4))
```

Results:

```
Values for finite duration HMM:
alfaHat:
[[1.     0.3847 0.4189]
 [0.     0.6153 0.5811]]
c:
[1.     0.1625 0.8266 0.0581]
```

2.2 Finite Duration Chain

Code for testing the HMM with a infinite chain:

```
#Test Forward algorithm NON-FINITE CHAIN
nonfiniteDurationMC = MarkovChain( np.array([1, 0]), np.array([[0.9, 0.1], [0.1, 0.9]]))
nonfiniteDurationHMM  = HMM(nonfiniteDurationMC, [g1, g2])
scaledProbOfObservations = nonfiniteDurationHMM.prob(observations)
alpha, c = nonfiniteDurationMC.forward(scaledProbOfObservations)
print("Values for infinite duration HMM:")
print("alfaHat:")
print(np.around(alpha, 4))
print("c: ")
print(np.around(c, 4))
```

Results:

```
Values for infinite duration HMM:
alfaHat:
[[1.     0.3847 0.4591]
 [0.     0.6153 0.5409]]
c:
[1.     0.1625 0.8881]
```

2.3 logprob

Finally the code used to implement the logprob function in the HMM as well the test code and results validating that it works can be seen below.

```
def logprob(self, observations):
    scaledProbOfObservations = self.prob(observations, False)
    alpha, c = self.stateGen.forward(scaledProbOfObservations)
    return np.sum(np.log(c))
```

Test code and results:

```
#Test logprob
logprob = finiteDurationHMM.logprob(observations)
print("logprob: ", logprob)
```

```
logprob:  -9.187726979475208
```

We can see that the results match those described in the Assignment, hence validating the implementation.

3 Notes

Code used to answer all questions can be found in its respective section in the jupyter notebook and given python classes.