Name: Ulugbek Khamdamov
ID: 20204120

Subject: Artificial Intelligence
Spring Semester 2021
Homework #2

Report

In simplest explanation my code is divided into for main functions to perform each inference task. I will describe the functionality and the logic behind the implementation of each function below:

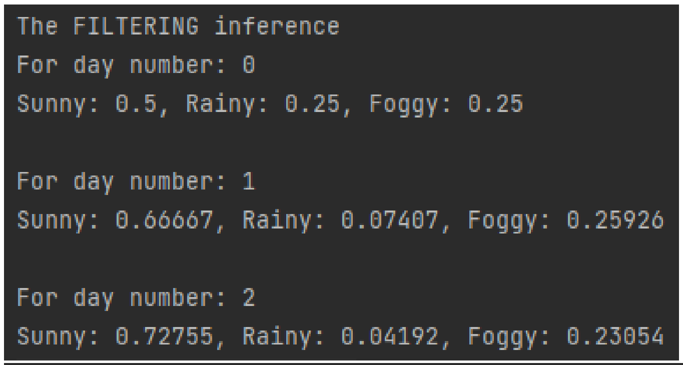
1) def filtering()
   As we covered during our classes, given the result of filtering up to time t, the agent needs to compute the result for t+1 from the evidence e_{t+1}. I have used formulas below, as a basis for the implementation of the filtering inference.

$$\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{e}_{1:t+1}) = f(\mathbf{e}_{t+1}, \mathbf{P}(\mathbf{X}_t \mid \mathbf{e}_{1:t})) ,$$

$$= \alpha \mathbf{P}(\mathbf{e}_{t+1} \mid \mathbf{X}_{t+1}) \sum_{\mathbf{x}_t} \mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{x}_t) P(\mathbf{x}_t \mid \mathbf{e}_{1:t}) \quad \text{(Markov assumption).}$$

   I have created a function called normalize to perform normalization throughout my code.
   For example, initially we will start at day 0 and we will have the following probability:
   P(D_0)=(0.5, 0.25, 0.25)
   Day 1: Using the “weather-test1-1000.txt” dataset we know that U_1 = false. So, the following is calculated:
   P(D_1) = (0.8, 0.05, 0.15) * 0.5 + (0.2, 0.6, 0.2)*0.25 + (0.2, 0.3, 0.5) * 0.25
   P(D_1|U_1) = αP(U_1| D_1)*P(D_1) = α (0.1, 0.8, 0.3) * (D_1)

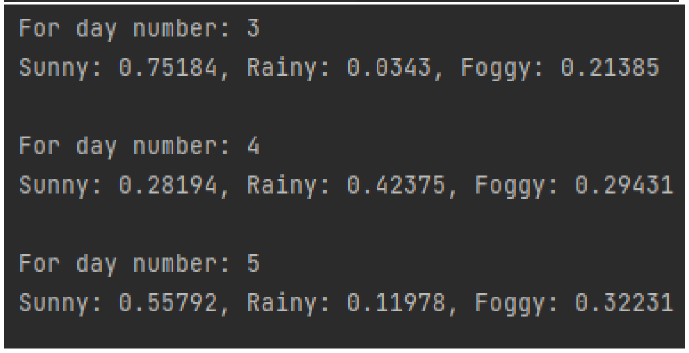
   The algorithm will iterate the same steps until the ‘t’ state is reached.
   Sample output for the filtering inference is given below:



```
The FILTERING inference
For day number: 0
Sunny: 0.5, Rainy: 0.25, Foggy: 0.25

For day number: 1
Sunny: 0.66667, Rainy: 0.07407, Foggy: 0.25926

For day number: 2
Sunny: 0.72755, Rainy: 0.04192, Foggy: 0.23054
```



```
For day number: 3
Sunny: 0.75184, Rainy: 0.0343, Foggy: 0.21385

For day number: 4
Sunny: 0.28194, Rainy: 0.42375, Foggy: 0.29431

For day number: 5
Sunny: 0.55792, Rainy: 0.11978, Foggy: 0.32231
```

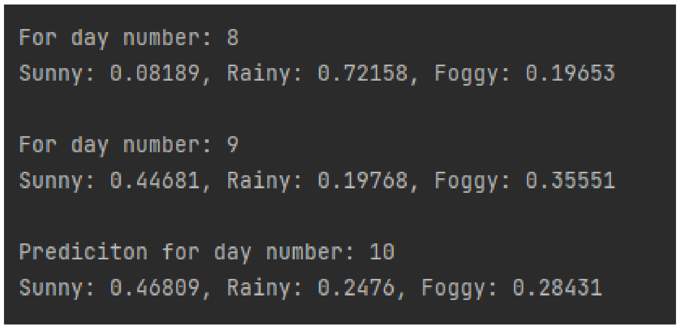
2) def prediction()

By leveraging the knowledge of implementing filtering inference, implementing prediction was easy because the task of prediction is simply filtering without the addition of new evidence. The formula below was used as a basis for implementing my code for this task. It is interesting to see that as we try to predict further and further into the future the predicted distribution for rain converges to a fixed point (0.5, 0.25, 0.25).

$$\mathbf{P}(\mathbf{X}_{t+k+1} \mid \mathbf{e}_{1:t}) = \sum_{\mathbf{x}_{t+k}} \mathbf{P}(\mathbf{X}_{t+k+1} \mid \mathbf{x}_{t+k}) P(\mathbf{x}_{t+k} \mid \mathbf{e}_{1:t}) .$$

To implement the prediction inference, the if control flow was created for checking whether we should start predicting (updating our probability with transition matrix only since we no longer have evidence data).

Sample output for the prediction inference is given below:



```
For day number: 8
Sunny: 0.08189, Rainy: 0.72158, Foggy: 0.19653

For day number: 9
Sunny: 0.44681, Rainy: 0.19768, Foggy: 0.35551

Prediciton for day number: 10
Sunny: 0.46809, Rainy: 0.2476, Foggy: 0.28431
```

3) def smoothing()

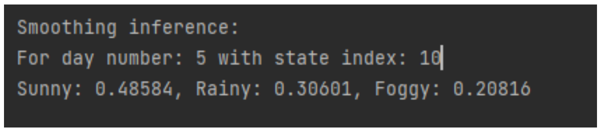
To implement smoothing interface, I used the forward-backward algorithm given in our book. For the forward part the filtering function was used and for implementing the backward part, a new function called "recursiveBackwardAlgorithm" was used. The formula below was used as a basis for implementing this function:

$$\begin{aligned}
\mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:t}) &= \mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\
&= \alpha\, \mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:k}) \mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k, \mathbf{e}_{1:k}) \quad \text{(using Bayes' rule)} \\
&= \alpha\, \mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:k}) \mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k) \quad \text{(using conditional independence)} \\
&= \alpha\, \mathbf{f}_{1:k} \times \mathbf{b}_{k+1:t} . \qquad (15.8)
\end{aligned}$$

The formula below was used as a basis for implementing recursiveBackwardAlgorithm.

$$\begin{aligned}
\mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k) &= \sum_{\mathbf{x}_{k+1}} \mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k, \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k) \quad \text{(conditioning on } \mathbf{X}_{k+1}\text{)} \\
&= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t} \mid \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k) \quad \text{(by conditional independence)} \\
&= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}, \mathbf{e}_{k+2:t} \mid \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k) \\
&= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1} \mid \mathbf{x}_{k+1}) P(\mathbf{e}_{k+2:t} \mid \mathbf{x}_{k+1}) \mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k) , \qquad (15.9)
\end{aligned}$$

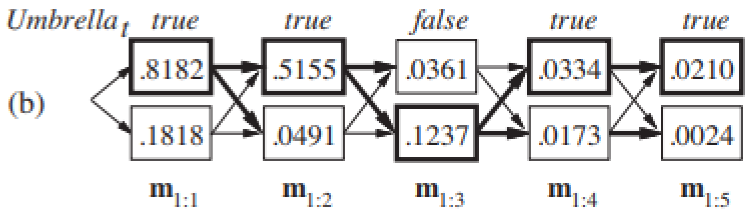
Sample output for the smoothing inference is given below:



```
Smoothing inference:
For day number: 5 with state index: 10
Sunny: 0.48584, Rainy: 0.30601, Foggy: 0.20816
```

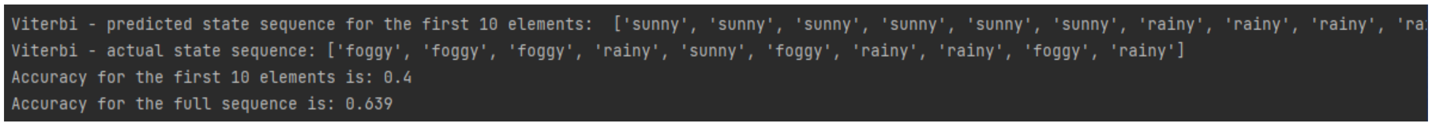
4) def viterbi()

To find the most likely sequence the Viterbi algorithm was implemented. For implementing Viterbi algorithm many possible sequences (or sometimes referred as a path) are found and the path with the highest probability is chosen as the most likely sequence.
The sample output of how Viterbi algorithm is given in our textbook. Below you can see that at each step we calculate the probability of reaching that state and we choose the most highly probably path at the end of the execution:



The implementation of viterbi algorithm achieved around 64 % of accuracy over the entire dataset.

Sample output for the viterbi algorithm is given below:



```
Viterbi - predicted state sequence for the first 10 elements:  ['sunny', 'sunny', 'sunny', 'sunny', 'sunny', 'sunny', 'rainy', 'rainy', 'rainy', 'ra
Viterbi - actual state sequence: ['foggy', 'foggy', 'foggy', 'rainy', 'sunny', 'foggy', 'rainy', 'rainy', 'foggy', 'rainy']
Accuracy for the first 10 elements is: 0.4
Accuracy for the full sequence is: 0.639
```