LAB EXERCISE – 15

Hidden Markov Model

1. Aim of the Experiment:

Implement and demonstrate Hidden Markov Model (HMM) to decode the hidden states given a sequence of observation states using Viterbi algorithm.

2. Reference to Text book for Algorithms:

Refer to Section 9.3.2 and 9.4 in Chapter 9 Probabilistic Graphical Models to understand the working of the algorithm.

Listing 1:

Consider the Hidden Markov Model which shows the performance of a student in an exam as 'Good', 'Moderate' and 'Fail' as hidden states in Figure 10.13. The observations we see on a student are 'Happy', 'Blink', 'Fine' and 'Fine'. Based on the observations we see on a student, we want to predict the state of how the student has written the exam. The decoding problem predicts the sequence of hidden states which is predicting the performance of the student in the exam.

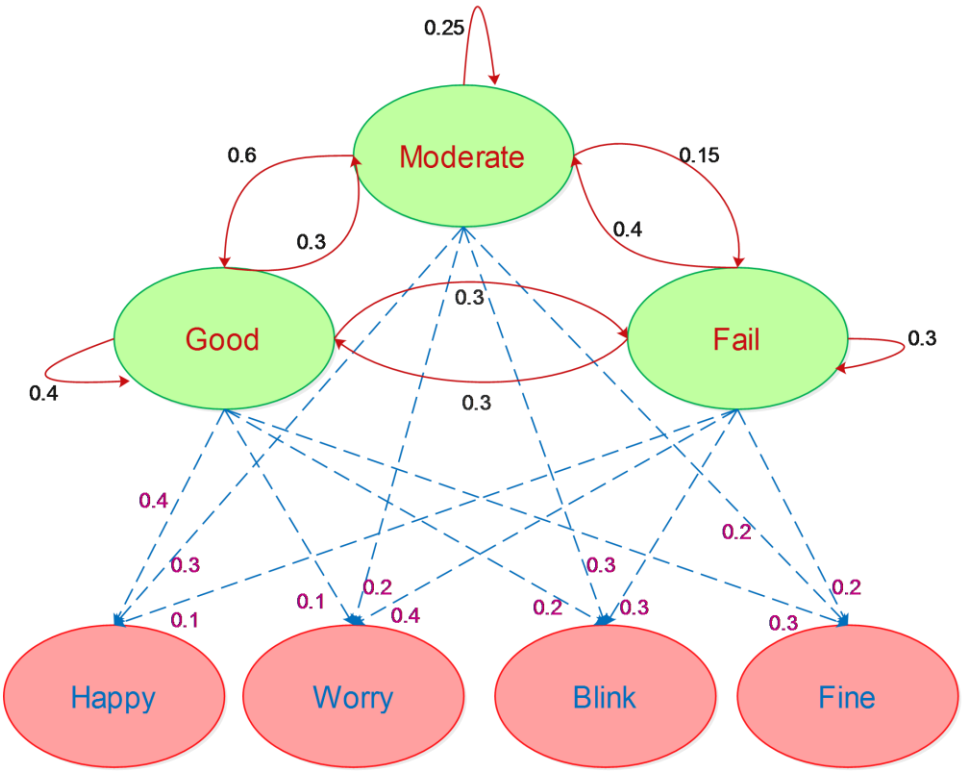


Figure 10.13: Sample Hidden Markov Model for Performance of a Student

Find the hidden states for the given observance sequence O_s = {Happy, Blink, Fine, Fine}.

3. Python Program with Explanation:

1. Initialize the 3 hidden states 'Good', 'Moderate' and 'Fail'.

```
states = ('Good', 'Moderate', 'Fail')
```

2. Initialize the 4 observation states 'Happy', 'Worry', 'Blink' and 'Fine'.

```
observations = ('Happy', 'Worry', 'Blink', 'Fine')
```

3. Set the initial probability of the hidden states.

```
initial_probability = {'Good': 0.5, 'Moderate': 0.25, 'Fail': 0.25}
```

4. Set the transition probability from each hidden state to other hidden states.

```
transition_probability = {
  'Good' : {'Good': 0.4, 'Moderate': 0.3, 'Fail' : 0.3},
  'Moderate' :   {'Good': 0.6, 'Moderate': 0.25, 'Fail' :0.15},
  'Fail' :   {'Good': 0.3, 'Moderate': 0.4, 'Fail' :0.3},
  }
```

5. Set the observation probability or emission probability from hidden states to observation states.

```
observation_probability = {
  'Good' : {'Happy': 0.4, 'Worry': 0.1, 'Blink': 0.2, 'Fine' :0.3 },
  'Moderate'   : {'Happy': 0.3, 'Worry': 0.2, 'Blink': 0.3, 'Fine' : 0.2},
  'Fail' : {'Happy': 0.1, 'Worry': 0.4, 'Blink': 0.3, 'Fine' :0.2 }
  }
```

6. Define the function ‘Viterbi’ to implement the logic of Viterbi algorithm. The function arguments are the observation sequence ‘obs’, the set of hidden states ‘states’, initial probability to hidden states i_pro, transition probability ‘t_pro’ and observation probability ‘o_pro’.

```
def Viterbi(obs, states, i_pro, t_pro, o_pro):
```

Set path to each state as empty.

```
        path = { s:[] for s in states}
```

Set the current probability alpha_pro to empty.

```
        alpha_pro = {}
```

Step 1: Initialization: Iterate this loop to calculate $\alpha_1(i)$ for each state i = 1 to no of states.

```
    for s in states:
        alpha_pro[s] = i_pro[s]*o_pro[s][obs[0]]
        print(alpha_pro)
```

Step 2: Recursion: Iterate this loop to calculate $\alpha_t(j)$ for each observation from the second one in the observation sequence and for each state.

```
    for i in range(1, len(obs)):
        last_pro = alpha_pro
        alpha_pro = {}
        for curr_state in states:
```

Calculate maximum probability at each step for each of the observation with each of the state.

```
    max_pro, last_sta =
max(((last_pro[last_state]*t_pro[last_state][curr_state]*o_pro[curr_state][obs[i]],
last_state) for last_state in states))
                alpha_pro[curr_state] = max_pro
                print(alpha_pro)
                path[curr_state].append(last_sta)
```

Step 3: Termination: Find the maximum probability path.

```
    max_pro = -1
    max_path = None
    for s in states:
        path[s].append(s)
        if alpha_pro[s] > max_pro:
            max_path = path[s]
            max_pro = alpha_pro[s]
```

```
        return max_path
```

7. Define the main function. Set the observation sequence ‘obs’ and call the Viterbi function defined.

```
    if __name__ == '__main__':
        obs = ['Happy', 'Blink', 'Fine', 'Fine']
        print("Given Observation Sequence\n", obs)
        print ("Maximum Probability Path \n", Viterbi(obs, states, initial_probability,
    transition_probability, observation_probability))
```

Complete Program:

```
states = ('Good', 'Moderate', 'Fail')

observations = ('Happy', 'Worry', 'Blink', 'Fine')

initial_probability = {'Good': 0.5, 'Moderate': 0.25, 'Fail': 0.25}

transition_probability = {
  'Good' : {'Good': 0.4, 'Moderate': 0.3, 'Fail' : 0.3},
  'Moderate' :   {'Good': 0.6, 'Moderate': 0.25, 'Fail' :0.15},
  'Fail' :   {'Good': 0.3, 'Moderate': 0.4, 'Fail' :0.3},
  }

observation_probability = {
  'Good' : {'Happy': 0.4, 'Worry': 0.1, 'Blink': 0.2, 'Fine' :0.3 },
  'Moderate'   : {'Happy': 0.3, 'Worry': 0.2, 'Blink': 0.3, 'Fine' : 0.2},
  'Fail' : {'Happy': 0.1, 'Worry': 0.4, 'Blink': 0.3, 'Fine' :0.2 }
  }

def Viterbi(obs, states, i_pro, t_pro, o_pro):
        path = { s:[] for s in states}
        print(path)
        alpha_pro = {}
        for s in states:
```

```
                alpha_pro[s] = i_pro[s]*o_pro[s][obs[0]]
                print(alpha_pro)
        for i in range(1, len(obs)):
                last_pro = alpha_pro
                alpha_pro = {}
                for curr_state in states:
                        max_pro, last_sta =
max(((last_pro[last_state]*t_pro[last_state][curr_state]*o_pro[curr_state][obs[i]], last_state)
                                for last_state in states))

                        alpha_pro[curr_state] = max_pro
                        print(alpha_pro)
                        path[curr_state].append(last_sta)

        # find the maximum probability path
        max_pro = -1
        max_path = None
        for s in states:
                path[s].append(s)
                if alpha_pro[s] > max_pro:
                        max_path = path[s]
                        max_pro = alpha_pro[s]
                return max_path

if __name__ == '__main__':
        obs = ['Happy', 'Blink', 'Fine', 'Fine']
        print("Given Observation Sequence\n", obs)
        print ("Maximum Probability Path \n", Viterbi(obs, states, initial_probability,
transition_probability, observation_probability))
```

Output:

```
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

========== RESTART: C:/Users/ADMIN/pythonpgms/final/jnf hmmtest.py ==========

Given Observation Sequence

['Happy', 'Blink', 'Fine', 'Fine']

{'Good': 0.2}

{'Good': 0.2, 'Moderate': 0.075}

{'Good': 0.2, 'Moderate': 0.075, 'Fail': 0.025}

{'Good': 0.016000000000000004}

{'Good': 0.016000000000000004, 'Moderate': 0.018}

{'Good': 0.016000000000000004, 'Moderate': 0.018, 'Fail': 0.018}

{'Good': 0.0032399999999999994}

{'Good': 0.0032399999999999994, 'Moderate': 0.00144}

{'Good': 0.0032399999999999994, 'Moderate': 0.00144, 'Fail': 0.00108}

{'Good': 0.00038879999999999996}

{'Good': 0.00038879999999999996, 'Moderate': 0.00019439999999999995}

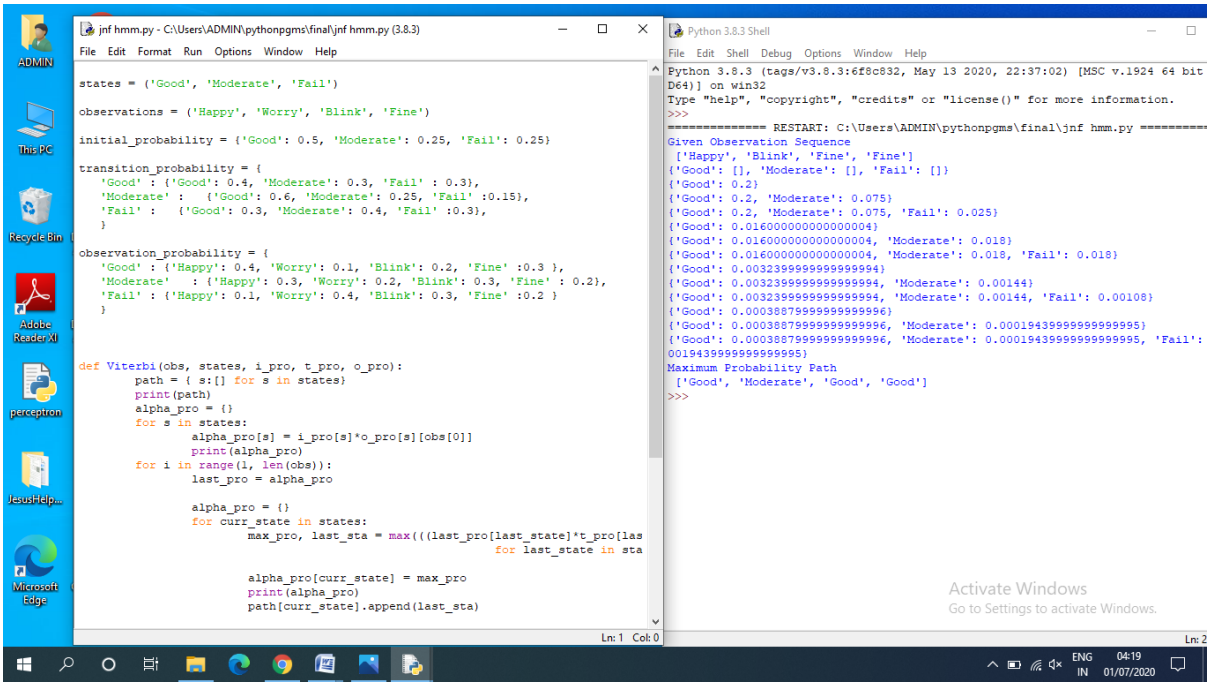
{'Good': 0.00038879999999999996, 'Moderate': 0.00019439999999999995, 'Fail': 0.00019439999999999995}

Maximum Probability Path

['Good', 'Moderate', 'Good', 'Good']

>>>

Screenshot of the Output:



Listing 2:

Refer Chapter 9 Probabilistic Graphical Models: Review Questions

Consider the Hidden Markov Model which shows the behaviour of a student going to school in the next morning as observation states 'Happy', 'Cry' and 'Dull' and hidden states 'Tuition', 'Sleep', ‘Movie’ and ‘Games’. Based on the observations seen on a student, predict the state of the student last evening.

Table 10.16 contains the observation probability, Table 10.17 contains the initial probability and Table 10.18 contains the transition probability.

Table 10.16: Observation Probability

Observation	**Happy**	**Cry**	**Dull**
Tuition	0.5	0.1	0.4
Sleep	0.35	0.45	0.2
Games	0.3	0.3	0.4
Movie	0.2	0.5	0.3

Table 10.17: Initial Probability

Initial Probability π			
0.25	0.25	0.25	0.25

Table 10.18: Transition Probability

Transition probability	**Tuition**	**Sleep**	**Games**	**Movie**
Tuition	0.4	0.3	0.2	0.1
Sleep	0.3	0.35	0.15	0.2
Games	0.6	0.05	0.1	0.25
Movie	0.1	0.5	0.2	0.2

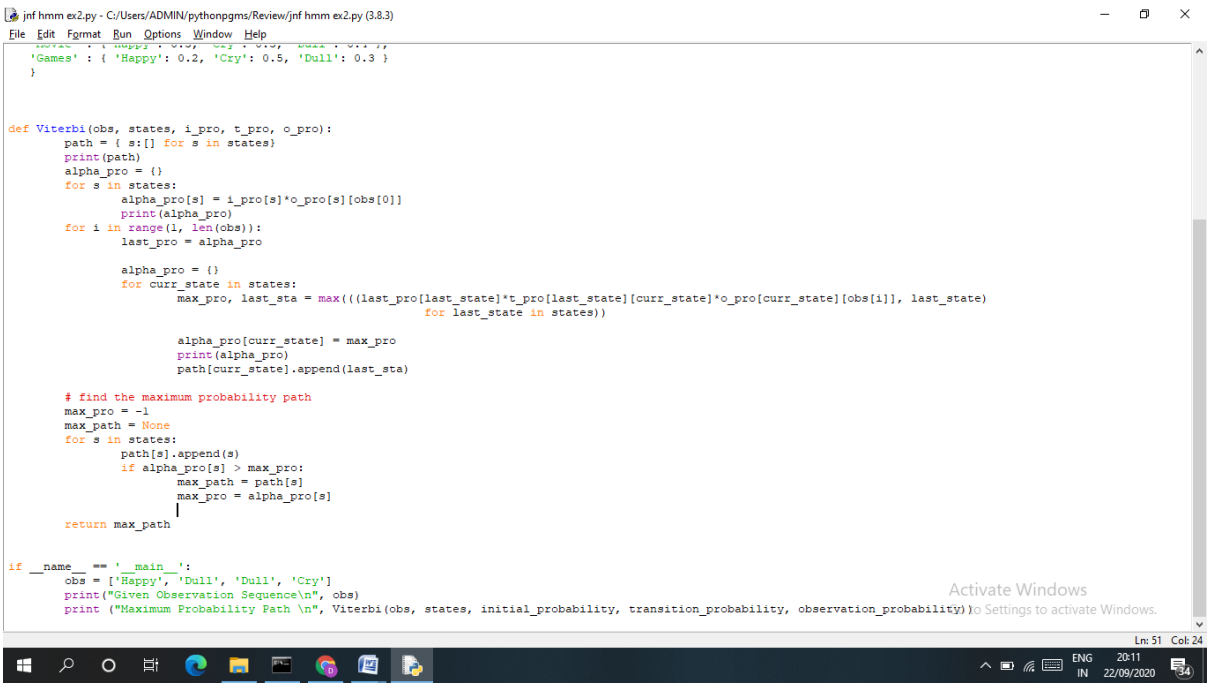
Assume the end probability as given in Table 10.19.

Table 10.19: End Probability

P(End|Tuition)	**P(End|Sleep)**	**P(End|Movie)**	**P(End|Games)**
0.1	0.2	0.1	0.2

Find the hidden states sequence for the given observance sequence O_s = {Happy, Dull, Dull, Cry}.

Screenshot of the Program:



Output:

Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:37:02) [MSC v.1924 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license()" for more information.

>>>

========== RESTART: C:\Users\ADMIN\pythonpgms\Review\jnf hmm ex2.py ==========

Given Observation Sequence

['Happy', 'Dull', 'Dull', 'Cry']

{'Tuition': [], 'Sleep': [], 'Movie': [], 'Games': []}

{'Tuition': 0.125}

{'Tuition': 0.125, 'Sleep': 0.0875}

{'Tuition': 0.125, 'Sleep': 0.0875, 'Movie': 0.075}

{'Tuition': 0.125, 'Sleep': 0.0875, 'Movie': 0.075, 'Games': 0.05}

{'Tuition': 0.020000000000000004}

{'Tuition': 0.020000000000000004, 'Sleep': 0.0075}

{'Tuition': 0.020000000000000004, 'Sleep': 0.0075, 'Movie': 0.010000000000000002}

{'Tuition': 0.020000000000000004, 'Sleep': 0.0075, 'Movie': 0.010000000000000002, 'Games': 0.005625}

{'Tuition': 0.003200000000000001}

{'Tuition': 0.003200000000000001, 'Sleep': 0.0012000000000000003}

{'Tuition': 0.003200000000000001, 'Sleep': 0.0012000000000000003, 'Movie': 0.0016000000000000005}

{'Tuition': 0.003200000000000001, 'Sleep': 0.0012000000000000003, 'Movie': 0.0016000000000000005, 'Games': 0.0007500000000000001}

{'Tuition': 0.00012800000000000005}

{'Tuition': 0.00012800000000000005, 'Sleep': 0.0004320000000000001}

{'Tuition': 0.00012800000000000005, 'Sleep': 0.0004320000000000001, 'Movie': 0.0001920000000000009}

{'Tuition': 0.00012800000000000005, 'Sleep': 0.0004320000000000001, 'Movie': 0.0001920000000000009, 'Games': 0.0002000000000000006}

Maximum Probability Path

['Tuition', 'Tuition', 'Tuition', 'Sleep']

>>>

Screenshot of the Output: