Middle East Technical University | Department of Computer Engineering

CENG 499

Introduction to Machine Learning

Fall '2022-2023

Homework 4

Due date: January 22, 2023, 23:55

Objectives

This assignment aims to fulfill the following objectives:

- To familiarize you with the forward and Viterbi algorithms that are crucial for solving the two important problems associated with Hidden Markov Models (namely evaluation and decoding problems) at the implementation level.

The assignment involves solely coding.

Part 1 - Hidden Markov Models

Hidden Markov models (HMMs) offer a mathematical framework to probabilistically model the data generation process of dependently distributed data instances/observations (i.e. time series dataset, DNA sequence, speech data, .etc) [EA(15), Duda(3), 6]. The framework features three parameters: $\lambda = (A, B, \Pi)$ where A is the transition probability matrix, B is the emission probability function and Π is the initial starting state distribution. For this part, we are going to assume an HMM has a finite number of states (discrete HMM, there are also HMMs with continuous states [5] but they are out of our assignment scope) and observations. Each state is represented with an integer number between $[0, N-1]$ when there are N states for the HMM. Similarly, each observation is represented with an integer number from the range $[0, M-1]$ when there are M different unique observations. With this assumption, the parameters of an HMM are defined as follows: $a_{ij} = P(q_{t+1} = S_j | q_t = S_i)$ (a_{ij} denotes the transition probability from state i to state j. q_{t+1} is the random variable of being in a state at time step $t+1$, similarly, q_t is the random variable of being in a state at time step t), $b_{jv} = P(\mathrm{O}_t = O_v | q_t = S_j)$ (b_{jv} is the probability of observing the observation v at time step t when the current state is state j at time step t. O_t is the random variable for the observation at the time step t), $\pi_i = P(q_1 = S_i)$ (π_i denotes the initial starting state probability for state i). Since we have assumed that there is a finite number of states and observations, A and B can be represented with matrices. $A = [a_{ij}]$, $B = [b_j v]$:

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \dots & a_{0,N-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & \dots & a_{1,N-1} \\ a_{2,0} & a_{2,1} & a_{2,2} & \dots & a_{2,N-1} \\ . & . & . & . & . \\ . & . & . & . & . \\ a_{N-1,0} & a_{N-1,1} & a_{N-1,2} & \dots & a_{N-1,N-1} \end{bmatrix}$$

$$B = \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & ... & b_{0,M-1} \\ b_{1,0} & b_{1,1} & b_{1,2} & ... & b_{1,M-1} \\ b_{2,0} & b_{2,1} & b_{2,2} & ... & b_{2,M-1} \\ . & . & . & . & . \\ . & . & . & . & . \\ b_{N-1,0} & b_{N-1,1} & b_{N-1,2} & ... & b_{N-1,M-1} \end{bmatrix}$$

$$\Pi = \begin{bmatrix} \pi_0 & \pi_1 & \pi_2 & ... & \pi_{N-1} \end{bmatrix}$$

With the following constraints: $\sum_{i=0}^{N-1} \pi_i = 1, \sum_{m=0}^{M-1} b_{im} = 1, \sum_{j=0}^{N-1} a_{ij} = 1$

Once the parameter (λ) is given or learned for an HMM, basically we can tackle with two main problems: evaluation and decoding problem [EA(15), Duda(3)]. The evaluation problem requires estimating a probability score for a given observation sequence O (O=$\{O_i, O_j, O_k, ..., O_m\}$ where each O_s is an observation). and formally formulated [EA(15)] as $P(\mathrm{O}_1 = O_i, \mathrm{O}_2 = O_j, \mathrm{O}_3 = O_k, ..., \mathrm{O}_T = O_m|\lambda)$ (O_1 is the random variable of the observation for the time step 1, O_2 is for the time step 2 and so on.). The decoding problem requires finding the most likely state sequence that has led to the emergence of a given observation sequence O and it is defined formally as [EA(15)]:

$$Q^* = \arg\max_Q P(Q|\mathrm{O}, \lambda)$$

For the evaluation problem, in order to solve it efficiently we can consider two algorithms which are instances of the dynamic programming paradigm: the forward algorithm, and the backward algorithm [EA(15)]. For this part, we are going to consider only the forward algorithm. The forward algorithm defines a forward variable $\alpha_t(i)$ (α value for state i at time step t) which can be calculated iteratively [EA(15)], lets assume that the given observation sequence is $\mathrm{O} = (O_j, O_l, O_b, O_v, O_m, O_v, ..., O_y)$:

$$\alpha_t(i) = P(\mathrm{O}_1 = O_j, \mathrm{O}_2 = O_l, ..., \mathrm{O}_t = O_m, q_t = S_i|\lambda)$$

The variable is initialized as follows for the first time step:

$$\alpha_1(i) = P(\mathrm{O}_1 = O_k, q_1 = S_i) = \pi_i b_{ik}$$

When the variable is written for a state j at the next time step:

$$\alpha_{t+1}(j) = P(\mathrm{O}_1 = O_1, \mathrm{O}_2 = O_2, ..., \mathrm{O}_t = O_m, \mathrm{O}_{t+1} = O_v, q_{t+1} = S_j|\lambda)$$

The following recursive calculation is obtained [EA(15)]:

$$\alpha_{t+1}(j) = \left[\sum_{i=0}^{N-1} \alpha_t(i) a_{ij}\right] b_{jv}$$

When the all $\alpha_T(i)$ values are calculated for all states (where T is the last time step), the probability of the observation sequence O can be calculated as follows [EA(15)]:

$$P(\mathrm{O}_1 = O_j, \mathrm{O}_2 = O_l, ..., \mathrm{O}_T = O_y|\lambda) = \sum_{i=0}^{N-1} \alpha_T(i) \quad (1)$$

For the decoding problem (or finding the most probable state sequence), we consider the Viterbi algorithm [Duda(3), EA(15)]. Similar to the forward algorithm, it defines a variable, namely $\delta_t(i)$ (δ value for state i at time step t), as follows, lets assume that the given observation sequence is O = $(O_e, O_x, O_z, O_h, O_k, O_u, O_r, ..., O_p)$:

$$\delta_t(i) = \max_{\Delta_1, \Delta_2, ..., \Delta_{t-1}} P(q_1 = \Delta_1, q_2 = \Delta_2, ..., q_t - 1 = \Delta_{t-1}, q_t = S_i, \mathrm{O}_1 = O_e, \mathrm{O}_2 = O_x, ..., \mathrm{O}_t = O_u|\lambda)$$

The variable is initialized as follows for the first time step:

$$\delta_1(i) = P(q_1 = S_i, \mathrm{O}_1 = O_e) = \pi_i b_{ie} \tag{2}$$

When the variable is written for a state j at the next time step:

$$\delta_{t+1}(j) = \max_{\Delta_1,\Delta_2,...,\Delta_t} P(q_1 = \Delta_1, q_2 = \Delta_2, ..., q_t = \Delta_t, q_{t+1} = S_j, \mathrm{O}_1 = O_e, \mathrm{O}_2 = O_x, ..., \mathrm{O}_t = O_u, \mathrm{O}_{t+1} = O_r|\lambda)$$

The following recursive computation is obtained [EA(15)]:

$$\delta_{t+1}(j) = \max_i \delta_t(i) a_{ij} b_{jr} \tag{3}$$

The algorithm also defines one extra variable to keep track of the most probable state sequence, namely $\psi_t(i)$ (ψ value for state i at time step t) (since δ keeps track of the most likely state sequence probability):

$$\psi_{t+1}(j) = \arg\max_i \delta_t(i) a_{ij} \tag{4}$$

The variable keeps track of the most likely state at the previous time for each state individually at the current time step. This additional variable is initialized as (since there is no previous most likely state before the first time step for any state):

$$\psi_1(i) = null$$

After calculating all δ and ψ values for each state at the last time step (T), we can find the probability score of the most likely state sequence ($Q^* = (\Delta_1^*, \Delta_2^*, \Delta_3^*, ..., \Delta_T^*)$, Δ_t^* denotes the most probable state at time step t) as follows [EA(15)]:

$$P(Q^*|\mathrm{O}, \lambda) = \max_i \delta_T(i) \tag{5}$$

The most likely state for the last time step (Δ_T^*) is found as follows:

$$\Delta_T^* = \arg\max_i \delta_T(i)$$

And finally, the most like state for each time step is found by backtracking the ψ variable:

$$\Delta_{T-1}^* = \psi_T(\Delta_T^*), \Delta_{T-2}^* = \psi_{T-1}(\Delta_{T-1}^*), \Delta_{T-3}^* = \psi_{T-2}(\Delta_{T-2}^*), ..., \Delta_1^* = \psi_2(\Delta_2^*)$$

$$\Delta_t^* = \psi_{t+1}(\Delta_{t+1}^*)$$

Similar to the naive Bayes calculations, during calculations of both Viterbi and forward algorithms, many small probability values are multiplied together hence these algorithms can suffer from numerical precision limitations (especially when the length of the observation sequence is high). To remedy this problem for both forward and Viterbi algorithms, first of all, we can apply the logarithm function to operations since both methods involve a series of multiplications. Further, for the forward algorithm we can normalize each $\alpha_t(i)$ by multiplying with c_t which is defined as follows [EA(15)]:

$$c_t = \frac{1}{\sum_{k=0}^{N-1} \alpha_t(k)}$$

$$\alpha_t^{normalized}(i) = \alpha_t(i).c_t$$

After normalizing all $\alpha_t(i)$, $\ln P(\mathrm{O}_1 = O_j, \mathrm{O}_2 = O_l, ..., \mathrm{O}_T = O_y|\lambda)$ has to be calculated as [EA(15)]:

$$\ln P(\mathrm{O}_1 = O_j, \mathrm{O}_2 = O_l, ..., \mathrm{O}_T = O_y|\lambda) = -\sum_{t=1}^{T} \ln c_t \tag{6}$$

When the logarithm function is applied to the operations of the Viterbi algorithm, the variable calculations become:

$$\delta_1(i) = \ln P(q_1 = S_i, \mathrm{O}_1 = O_e) = \ln \pi_i + \ln b_{ie} \tag{7}$$

$$\delta_{t+1}(j) = \max_i(\delta_t(i) + \ln a_{ij}) + \ln b_{jr} \tag{8}$$

$$\psi_{t+1}(j) = \arg\max_i \delta_t(i) + \ln a_{ij} \tag{9}$$

$$\ln P(Q^*|\mathrm{O}, \lambda) = \max_i \delta_T(i) \tag{10}$$

In this part, you are expected to implement both the forward algorithm and Viterbi algorithms by considering the possible numerical representation issues.

Note 1: c_t value should be calculated after finding α values of all states at time step t. Then each $\alpha_t(i)$ should be multiplied with the calculated c_t for normalization.

Part 1 Specifications

- For this part, you are given two Python files: `mainHMM.py`, `HMM.py`. In the `mainHMM.py`, an HMM object whose class implementation is in `HMM.py` is created with a predetermined set of parameter values. With the created object the forward algorithm and the Viterbi algorithms are executed for a given observation sequence and results are displayed. All the parameter values are specified by sticking to the notation considered in this part (i.e. A, B, Π definitions). You are expected to complete the missing functions of the HMM class:

```
class HMM:
    def __init__(self, A, B, Pi):
        self.A = A
        self.B = B
        self.Pi = Pi

    def forward_log(self, O: list):
        """
        :param O: is the sequence (an array of) discrete (integer) observations, i.e. [0, 2,1 ,3, 4]
        :return: ln P(O|λ) score for the given observation, ln: natural logarithm
        """


    def viterbi_log(self, O: list):
        """
        :param O: is an array of discrete (integer) observations, i.e. [0, 2,1 ,3, 4]
        :return: the tuple (ln P(Q*|O,λ), Q*), Q* is the most probable state sequence for the given O
        """
```

- Your implementation should work with arbitrary parameter definitions (i.e. different matrices with different sizes for A, B, Π). All observations are represented with an integer value in the range [0, $M-1$], where M is the number of distinct observations.

- You are expected to implement the numerically stable versions of the forward and Viterbi algorithms.

- For both methods, you are expected to use only the Numpy package, other library usages are forbidden.
- You may add further functions and variable definitions to the HMM class.
- Here is a couple of expected results taken from a sample implementation of HMM.py (a partial code segment from `mainHMM.py`):

```
A = np.array([[0.4, 0.6],
              [0.7, 0.3]], dtype=np.float128)

B = np.array([[0.3, 0.4, 0.3],
              [0.1, 0.2, 0.7]], dtype=np.float128)

Pi = np.array([0.6, 0.4], dtype=np.float128)

hmm = HMM(A, B, Pi)

O1 = [2, 1, 0]
O2 = [0,0,2,1,0]

print(hmm.forward_log(O1))
print(hmm.forward_log(O2))
print(hmm.viterbi_log(O1))
print(hmm.viterbi_log(O2))
```

Output:

```
-3.557430227693738272
-6.6208575140190808516
(-4.6661948878258659437, [1, 0, 0])
(-8.095791744009719443, [0, 0, 1, 0, 0])
```

Regulations

1. You are expected to write your code in Python by using the Numpy library.
2. Falsifying results or changing the composition of training, validation, and test data is strictly forbidden, and you will receive 0 if this is the case. Your programs will be examined to see if you have actually reached the results and if it is working correctly.
3. **Commenting:** Since all implementations are going to be inspected manually, comments are of great importance for the evaluation procedure. Please add extensive explanatory comments in all of your implementations.
4. **Late Submission:** You have a total of 5 late days for all homework without receiving a penalty. As soon as you have depleted your quota, penalization will be in effect. The late submission penalty will be calculated using $5d^2$, that is, 1 day late submission will cost you 5 points, 2 days will cost you 20 points, and 3 days will cost you 45 points. No late submission is accepted after reaching a total of 3 late days (No matter whether you have still a remaining late-day quota or not).
5. **Cheating:** Using any piece of code that is not your own is strictly forbidden and constitutes cheating. This includes friends, previous homework, or the internet. However, example code snippets shared on Scikitlearn's website can be used. **We have a zero-tolerance policy for cheating**. People involved in cheating will be punished according to university regulations.

6. **Discussion:** You must follow ODTUClass for discussions and possible updates/corrections/clarifications on a daily basis. **For the previous assignments, we received many questions that could have been asked on the discussion forum. Please ask your questions on ODTUClass unless you really think that your question is private and does not concern anyone**.

7. **Evaluation:** Part 1 is going to be evaluated with black-box testing. When black-box testing fails, your implementations are going to be checked manually.

Submission

Submission will be done via the ODTUClass system. For Part 1, you are expected to upload **HMM.py**.

References

1. Duda(X) := Richard O. Duda, Peter E. Hart, and David G. Stork. 2000. Pattern Classification (2nd Edition) Chapter X.

2. EA(X) := Introduction to Machine Learning, 2nd edition, Ethem Alpaydın, Chapter X.

3. Bishop(X) := Pattern Recognition and Machine Learning, Christopher M. Bishop, Chapter X.

4. Haykin(X) := Neural Networks and Learning Machines, 3rd edition, Simon Haykin, Chapter X.

5. Ainsleigh, Phillip L. Theory of continuous-state hidden Markov models and hidden Gauss-Markov models. NAVAL UNDERSEA WARFARE CENTER DIV NEWPORT RI, 2001.

6. Rabiner, Lawrence R. "A tutorial on hidden Markov models and selected applications in speech recognition." Proceedings of the IEEE 77, no. 2 (1989): 257-286.

7. Lecture notes

8. Announcements Page

9. Discussion Forum