In the name of God
Machine Learning (Summer 2020)
Assignment #5: Hidden Markov Models

Due date: 14th of Mordad

In this assignment, you will implement solutions to two problems associated with HMMs: the **Viterbi** algorithm, and the **Forward-Backward** algorithm. The Viterbi algorithm is used for supervised tasks and the Forward-Backward algorithm is employed for semi-supervised, and unsupervised tasks. A **hidden Markov model** (**HMM**) allows us to talk about both **observed** events (e.g. words) and **hidden** events (e.g. part-of-speech tags). An HMM is specified by the following components:

$Q = q_1 q_2 \dots q_N$	a set of N **states**
$A = a_{11} \dots a_{ij} \dots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \dots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, \dots, v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state i
$\pi = \pi_1, \pi_2, \dots, \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

Where $\boldsymbol{a_{ij} = Pr(q_{t+1} = S_j | q_t = S_i) = (q_{t+1} = tag_j | q_t = tag_i)}$, and $\boldsymbol{b_{jk} = Pr(O_t = k | q_t = S_j) = Pr(O_t = word_k | q_t = tag_j)}$.

Problem1) The Viterbi Algorithm (supervised task)

For any model, such as an HMM, that contains hidden variables, the task of finding which sequence of variables is the most likely tag sequence given the sequence of observations (words), is called the **decoding** task. The task of the **decoder** is to find the best hidden variable sequence ($q_1 q_2 q_3 \dots q_n$). The most common decoding algorithms for HMMs is the **Viterbi algorithm**. This algorithm is a kind of **dynamic programming**.

Each cell $\boldsymbol{v_t(j)}$, represents the probability that the HMM is in state $\boldsymbol{j}$ after seeing the first $\boldsymbol{t}$ observations and passing through the most probable state sequence $q_1 \dots q_{t-1}$. The value of each cell $\boldsymbol{v_t(j)}$ is computed by recursively taking the most probable path.
Like other dynamic programming algorithms, **Viterbi** fills each cell recursively. The Viterbi probability is computed by taking the most probable of the extensions of the paths that lead to the current cell, provided the Viterbi probability had already been calculated in every state at time $t-1$. For a given state $\boldsymbol{q_j}$ at time **t**, the Viterbi probability $\boldsymbol{v_t(j)}$ is computed in log space as

$$v_t(j) \leftarrow \max_{i=1}^{N} \left(v_{t-1}(i) + \ln(a_{ij}) + \ln\left(b_j(o_t)\right) \right)$$

Where $v_{t-1}(i)$ is the previous Viterbi path probability from the previous time step, a_{ij} is the transition probability from previous state q_i to the current state q_j, and $b_j(o_t)$ is the emission probability of the observation symbol o_t given the current state j. Pseudocode for the Viterbi algorithm is given in the following.

In the name of God
Machine Learning (Summer 2020)
Assignment #5: Hidden Markov Models

Function VITERBI (observations of len T, state-graph of len N) **returns** best-path
create a path probability matrix $viterbi[N,T]$
for each state **s from** 1 to N **do** // Initialization step
$viterbi[s,1] \leftarrow \ln(\pi_s) + \ln(b_s(o_1))$
$backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 to T **do** // recursion step
for each state s **from** 1 to N **do**

$$viterbi[s,t] \leftarrow \ln(b_s(o_t)) + \max_{s'=1}^{N} \left(viterbi[s',t-1] + \ln(a_{s',s}) \right)$$

$$backpointer[s,t] \leftarrow \operatorname*{argmax}_{s'=1}^{N} viterbi[s',t-1] + \ln(a_{s',s}))$$

$$bestpathpointer \leftarrow \operatorname*{argmax}_{s=1}^{N} viterbi[s,T]$$ // termination step

$bestpath \leftarrow the\ path\ starting\ at\ state\ bestpathointer, the\ follows\ backpointer[\ \ ]to\ states\ back\ in\ time$
return $bestpath$

Problem2) The Forward-Backward Algorithm (semi-supervised task)

This algorithm learns the parameters of an HMM, which are, the transition probability matrix **A,** and the emission probability matrix **B** in a semi-supervised manner. In fact, the input to such a learning algorithm would be an unlabeled sequence of observations O and a vocabulary of potential hidden states Q.

The standard algorithm for HMM training is the **forward-backward**, or **Baum-Welch** algorithm, a special case of the **Expectation-Maximization** or **EM** algorithm. The algorithm trains both the transition probabilities A and the emission probabilities B of the HMM. EM is an iterative algorithm, computing an initial estimate for the probabilities, then using those estimates to computing a better estimate, and so on, iteratively improving the probabilities that it learns. The Baum-Welch algorithm solves this problem by iteratively estimating the counts. The Baum-Welch algorithm starts with an estimate for the transition and observation probabilities and then uses these estimated probabilities to derive better and better probabilities.

To understand the algorithm, we need to define the **forward** and **backward probabilities**. The **forward algorithm** is a kind of dynamic programming algorithm, that is, an algorithm that uses a table to store intermediate values as it builds up the probability of the observation sequence. The forward algorithm computes the observation probability by summing over the probabilities of all possible hidden state paths that could generate the observation sequence.
Each cell of the forward algorithm at $\boldsymbol{\alpha_t(j)}$ represents the probability of being in state $\boldsymbol{j}$ after seeing the first $\boldsymbol{t}$ observations. The value of each cell at $\boldsymbol{\alpha_t(j)}$ is computed by summing over the probabilities of every path that could lead to this cell. Formally, each cell expresses the following probability:

$$\alpha_t(j) = P(o_1, o_2, \dots o_t., q_t = j)$$

Here, $\boldsymbol{q_t = j}$ means "the $\boldsymbol{t^{th}}$ state in the sequence of states is state $\boldsymbol{j}$". This probability at $\boldsymbol{\alpha_t(j)}$ is computed by summing over the extensions of all the paths that lead to the current cell. For a given state $\boldsymbol{q_j}$ at time t, the value at $\boldsymbol{\alpha_t(j)}$ is computed as

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t)$$

Where $\alpha_{t-1}(j)$ is the previous forward path probability from the previous time step. The pseudocode for the forward algorithm is given in the following.

Function ForwardAlg1 (observations of len T, state-graph of len N) **returns** forward-prob
create a probability matrix $forward[N,T]$
for each state **s from** 1 to N **do** // Initialization step
    $forward[s,1] \leftarrow \pi_s * b_s(o_1)$
for each time step t **from** 2 to T **do** // recursion step
    for each state s **from** 1 to N **do**
        $$forward[s,t] \leftarrow \sum_{s'=1}^{N} forward[s',t-1] * a_{s's} * b_s(o_t)$$
return $forward$

There are some implementational issues both for the Forward algorithm and the Backward algorithm described later. The most severe practical problem is that multiplying many probabilities always yields very small numbers that will give underflow errors on any computer. For this reason, the Forward algorithm has been presented by the **ForwardAlg2** done in log space, which will make the numbers stay reasonable.

Function ForwardAlg2 (observations of len T, state-graph of len N) **returns** forward-prob
create a probability matrix $forward[N,T]$
for each state **s from** 1 to N **do** // Initialization step
    $forward[s,1] \leftarrow \ln(\pi_s) + \ln(b_s(o_1))$
for each time step t **from** 2 to T **do** // recursion step
    for each state s **from** 1 to N **do**
        $tmp = forward[1,t-1] + \ln(a_{1s}) + ln(b_s(o_t))$
        for each state s' **from** 2 to N
            $tmp1 = forward[s',t-1] + \ln(a_{s's}) + ln(b_s(o_t))$
            $$tmp \leftarrow \begin{cases} \text{tmp} + \ln(1+\exp(tmp1 - tmp))) & if\ tmp1 \le tmp \\ \text{tmp1} + \ln(1+\exp(tmp - tmp1))) & o,w. \end{cases}$$
        $forward[s,t] = tmp$
return $forward$

The **backward** probability $\boldsymbol{\beta}$ is the probability of seeing the observations from time $\boldsymbol{t+1}$ to the end, given in state i at time t: $\boldsymbol{\beta_t(i) = P(o_{t+1}, o_{t+2}, \dots, o_T | q_t = i)}$.

It is computed inductively in a similar manner to the forward algorithm. The pseudocode for the backward algorithm is given in the following.

Function BackwardAlg1 (observations of len T, state-graph of len N) **returns** backward-prob
create a probability matrix backward[N,T]
for each state **s from** 1 to N **do** // Initialization step
$backward[s,T] \leftarrow 1$
for each time step t **from** T-1 downto 1 **do** // recursion step
for each state s **from** 1 to N **do**

$$backward[s,t] \leftarrow \sum_{s'=1}^{N} backward[s',t+1] * a_{ss'} * b_{s'}(o_{t+1})$$

return $backward$

As mentioned before, there are some implementational issues for the Backward algorithm. The most severe practical problem is that multiplying many probabilities always yields very small numbers that will give underflow errors on any computer. For this reason, the Backward algorithm has been presented by the **BackwardAlg2** done in log space.

Function BackwardAlg2 (observations of len T, state-graph of len N) **returns** backward-prob
create a probability matrix backward[N,T]
for each state **s from** 1 to N **do** // Initialization step
$backward[s,T] \leftarrow 0$
for each time step t **from** T-1 downto 1 **do** // recursion step
for each state s **from** 1 to N **do**
$tmp = backward[1,t+1] + \ln(a_{s1}) + ln(b_1(o_{t+1}))$
for each state s' **from** 2 to N **do**
$tmp1 = backward[s',t+1] + \ln(a_{ss'}) + \ln(b_{s'}(o_{t+1}))$

$$tmp \leftarrow \begin{cases} \text{tmp} + \ln(1+\exp(tmp1 - tmp))) & if\ tmp1 \le tmp \\ \text{tmp1} + \ln(1+\exp(tmp - tmp1))) & o,w. \end{cases}$$

$backward[s,t] \leftarrow tmp$
return $backward$

We are now ready to see how the forward and backward probabilities can help compute the transition probability $\boldsymbol{a_{ij}}$ and observation probability $\boldsymbol{b_i(o_t)}$ from an observation sequence.

Let's define the probability $\boldsymbol{\xi_t}$ as the probability of being in state $\boldsymbol{i}$ at time $\boldsymbol{t}$ and state $\boldsymbol{j}$ at time $\boldsymbol{t+1}$, given the observation sequence:

$$\boldsymbol{\xi_t(i,j) = P(q_t = i, q_{t+1} = j | O)}$$

The following equation is used to compute $\boldsymbol{\xi_t}$.

$$\boldsymbol{\xi_t(i,j)} = \frac{\alpha_t(i) a_{ij} \mathrm{b_j}(\mathrm{o_{t+1}}) \beta_{\mathrm{t+1}}(\mathrm{j})}{\sum_{m=1}^{N} \alpha_T(m)}$$

The transition probability $\boldsymbol{a_{ij}}$ is computed by the equation below:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i,k)}$$

We also need a formula for recomputing the observation probability $\boldsymbol{b_{ij}}$. This is the probability of a given symbol $\boldsymbol{v_k}$ from the observation vocabulary **V**, given a state j: $\boldsymbol{\hat{b}_j(v_k)}$**.** For this, we need to know the probability of being in state j at time t, called $\gamma_t(j)$:

$$\gamma_t(j) = P(q_t = j|O)$$

The following equation is used to compute $\gamma_t(j)$:

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O)} = \frac{\alpha_t(j)\beta_t(j)}{\sum_{i=1}^{N} \alpha_T(i)}$$

The observation probability $\boldsymbol{b_j(v_k)}$ is re-estimated by the equation below:

$$\hat{b}_j(v_k) = \frac{\sum_{t=1,s.t,o_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

The re-estimations of the transition **A** and observation **B** probabilities form the core of the iterative forward-backward algorithm. The forward-backward algorithm starts with some initial estimate of the HMM parameters $\lambda = (A, B)$. The algorithm then iteratively runs two steps. Like other cases of the EM algorithm, the forward-backward algorithm has two steps: the **expectation step** (**E-step**), and the **maximization step** (**M-step**). In the E-step, the expected state occupancy count γ and the expected state transition count ξ is computed from the earlier A and B probabilities. In the M-step, γ and ξ are used to recompute new A and B probabilities. Pseudocode for the forward-backward algorithm is given in the following.

Function Forward-Backward (training sequences, output vocabulary V, hidden state set Q, initial transition probability matrix A, initial emission probability matrix B) **returns** $HMM = (A, B)$

iterate until convergence

// for each train sequence with length *T*, use E-step & M-step iteratively.

E-step

Calculate $\alpha_t(j)$ using the Forward Algorithm $\forall t: 1 \dots T \text{ and } j: 1 \dots N$

Calculate $\beta_t(j)$ using the Backward Algorithm $\forall t \text{ and } j$

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\sum_{m=1}^{N} \alpha_T(m)} \quad \forall t \text{ and } j$$

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{m=1}^{N} \alpha_T(m)} \quad \forall t, i, \text{and } j$$

M-step:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i,k)}$$

$$\hat{b}_j(v_k) = \frac{\sum_{t=1,s.t,o_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

return A, B

In the name of God
Machine Learning (Summer 2020)
Assignment #5: Hidden Markov Models

Note1: Each sequence (sentence) ends at the word ###. In fact, the word ### is the boundary of a sentence. The vocabulary contains the word ###. Consider its corresponding tag ###. Calculate the initial probability distribution π_i for each tag located at the start of the sentence (usually after the word ###). Use the add-one smoothing method for computing the initial probability π_i **;** ($\pi_i = \frac{\boldsymbol{count(q_1 = tag_i) + 1}}{\boldsymbol{\#sentences + \#tags}}$), and then normalize the probability distribution **π** such that $\sum_{i=1}^{N} \pi_i = 1$. Use this initial probability distribution for both the supervised and semi-supervised problems.

Note2: There are three datasets: **ic:** Ice cream cone sequences with 1-character tags (C, H); **en:** English word sequences with 1-character tags; and **cz:** Czech word sequences with 2-character tags.

Each dataset consists of three files: **train:** tagged data for supervised training (en provides 4,000-100,000 words. entrain4k, entrain10k, and entrain25k are shorter versions of entrain.); **test:** tagged data for testing (25,000 words for en); ignore the tags in this file except when measuring the accuracy of its tagging; and **raw:** untagged data for reestimating parameters (100,000 words for en).

The file format is simple. Each line has a single word/tag pair separated by the / character. (In the raw file, only the word appears.) Punctuation marks count as words. The special word ### is used for sentence boundaries.

Use train files for calculating transition probability matrix A, emission probability matrix B, and the initial probability distribution π. Employ raw files for the Forward-Backward algorithm in a semi-supervised task. Evaluate the model performance on test files.

Follow these steps to implement the Viterbi algorithm in a supervised manner. Consider words as observations and tags as hidden states.

1. Use the train file and calculate the transition probability matrix A, the emission probability matrix B, and the initial probability **π**. Compute state transition probabilities $\boldsymbol{a_{ij}}$, and observation probabilities b_{jk} based on the add-one smoothing method. ($\boldsymbol{a_{ij} = Pr(q_{t+1} = tag_j | q_t = tag_i) = \frac{count(q_{t+1}=tag_j \wedge q_t=tag_i) + 1}{count(q_t=tag_i) + \#tags}}$, and $\boldsymbol{b_{jk} = Pr(O_t = word_k | q_t = tag_j) = \frac{count(O_t=word_k \wedge q_t=tag_j) + 1}{count(q_t=tag_j) + \#words}}$ **).**
2. Implement the Viterbi algorithm. Run the Viterbi algorithm on each sentence as a sequence of the test file separately, and specify a tag sequence. Report tag sequences, and evaluate the model performance based on the accuracy.

Follow these steps to implement the Forward-Backward algorithm in a semi-supervised manner. Implement the Forward-Backward algorithm in two ways (steps 2 and 3):

1. Use the train file and calculate the transition probabilities A, and emission probabilities B, and the initial probability **π**. Compute state transition probabilities $\boldsymbol{a_{ij}}$, and observation probabilities b_{jk} based on the add-one smoothing method. ($\boldsymbol{a_{ij} = Pr(q_{t+1} = tag_j | q_t = tag_i) = \frac{count(q_{t+1}=tag_j \wedge q_t=tag_i) + 1}{count(q_t=tag_i) + \#tags}}$, and $\boldsymbol{b_{jk}}$ =

In the name of God
Machine Learning (Summer 2020)
Assignment #5: Hidden Markov Models

$\boldsymbol{Pr(O_t = word_k | q_t = tag_j) = \frac{count(O_t = word_k \wedge q_t = tag_j) + 1}{count(q_t = tag_j) + \#words}}$). Consider these obtained probabilities as initial probabilities for Forward-Backward algorithm.

2. Implement the Forward-Backward algorithm. In E-step, obtain $\alpha_t(j)$ using the **ForwardAlg1** and $\beta_t(j)$ using the **BackwardAlg1**, and then normalize $\alpha_t(j)$ and $\beta_t(j)$ for each time t such that $\sum_{j=1}^{N} \alpha_t(j) = 1$, and $\sum_{j=1}^{N} \beta_t(j) = 1$. After normalizing, Use the normalized forward probability $\alpha_t(j)$ and, the normalized backward probability $\beta_t(j)$ to calculate $\gamma_t(j)$ and $\xi_t(i,j)$. Run the algorithm on raw files for 100 iterations. Learn the parameters of an HMM, which are, the state transition probability matrix, **A** and the observation probability matrix, **B** matrices. Employ the Viterbi algorithm based on the learned parameters (A, B). For each given word sequence (sentence) of the test file, run the Viterbi algorithm, and specify a tag sequence. Report tag sequences, and Evaluate the model performance based on the accuracy.
3. Implement the Forward-Backward algorithm. In E-step, obtain $\alpha_t(j)$ using the **ForwardAlg2** and $\beta_t(j)$ using the **BackwardAlg2**, and then calculate $\gamma_t(j)$ and $\xi_t(i,j)$. Run the algorithm on raw files for 100 iterations. Learn the parameters of an HMM, which are, the state transition probability matrix, **A** and the observation probability matrix, **B** matrices. Employ the Viterbi algorithm based on the learned parameters (A, B). For each given word sequence (sentence) of the test file, run the Viterbi algorithm, and specify a tag sequence. Report tag sequences, and Evaluate the model performance based on the accuracy.

Questions:

1) Explain why you think the EM reestimation procedure (the Forward-Backward algorithm) helped where it did. How did it get additional value out of the raw files?

2) What is the role of the parameter $\boldsymbol{\gamma_t(j)}$?

3) What is the role of the parameter $\boldsymbol{\xi_t(i,j)}$?

4) Compare the results of the Forward-Backward algorithm on test files to the results of the Viterbi algorithm in a supervised way. Is the performance of the Forward-Backward Algorithm always improved? Why? Explain your answer.

5) Suggest at least two reasons to explain why EM didn't always help.

Important Notes:

- Pay extra attention to the due date. It will not extend.
- Be advised that submissions after the deadline would not grade.
- Provide a report for your assignment, answer the questions and explain about your results.
- The name of the uploading file should be your **Lastname_Firstname.**