MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.806/6.864 Advanced Natural Language Processing
Fall 2015

Assignment 2, Due: 9am on Tuesday Oct 13. Upload pdf or scan to Stellar.

This homework contains some reading material and 16 questions for a total of 65 points. Question 5.3 requires you to work with the Python code package provided.

Hidden Markov Models

In many practical problems, we would like to model pairs of sequences. Consider, for instance, the task of part-of-speech (POS) tagging. Given a sentence, we would like to compute the corresponding tag sequence:

Input:"Faith is a fine invention"
Output:"Faith/N is/V a/D fine/A invention/N"

More generally, a *sequence labeling problem* involves mapping a sequence of observations $x_1, x_2, \ldots, x_n$ into a sequence of tags $y_1, y_2, \ldots, y_n$. In the example above, every word x is tagged by a single label y. One possible approach for solving this problem would be to label each word independently. For instance, a classifier could predict a part-of-speech tag based on the word, its suffix, its position in the sentence, etc. In other words, we could construct a feature vector on the basis of the observed "context" for the tag, and use the feature vector in a linear classifier. However, tags in a sequence are dependent on each other and this classifier would make each tagging decision independently of other tags. We would like our model to directly incorporate these dependencies. For instance, in our example sentence, the word *"fine"* can be either noun (N), verb (V) or adjective (A). The label V is not suitable since a tag sequence "D V" is very unlikely. Today, we will look at a model – a Hidden Markov Model – that allows us to capture some of these correlations.

Generative Tagging Model

Assume a finite set of words Σ and a finite set of tags $\mathcal{T}$. Define S to be the set of all sequence tag pairs $(x_1, \ldots, x_n, y_1, \ldots, y_n)$, $x_i \in \Sigma$ and $y_i \in \mathcal{T}$ for $i = 1 \ldots n$. S here contains sequences of different lengths as well, i.e., n varies as well. A generative tagging model is a probability distribution p over pairs of sequences:

- $p(x_1, \ldots, x_n, y_1, \ldots, y_n) \geq 0, \ \forall (x_1, \ldots, x_n, y_1, \ldots, y_n) \in S$
- $\sum_{(x_1,\ldots,x_n,y_1,\ldots,y_n)\in S} p(x_1, \ldots, x_n, y_1, \ldots, y_n) = 1$

If we have such a distribution, then we can use it to predict the most likely sequence of tags $y_1, \ldots, y_n$ for any observed sequence of words $x_1, \ldots, x_n$, as follows

$$f(x_1, \ldots, x_n) = \text{argmax}_{y_1,\ldots,y_n} p(x_1, \ldots, x_n, y_1, \ldots, y_n)$$

where we view f as a mapping from word sequences to tags.
Three key questions:

- How to specify $p(x_1, \ldots, x_n, y_1, \ldots, y_n)$ with a few number of parameters (degrees of freedom)
- How to estimate the parameters in this model based on observed sequences of words (and tags).
- How to predict, i.e., how to find the most likely sequence of tags for any observed sequence of words: evaluate $\text{argmax}_{y_1,\ldots,y_n} p(x_1, \ldots, x_n, y_1, \ldots, y_n)$

1 Model definition

Let $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$ be sequences of random variables of length n. We wish to specify a joint probability distribution

$$P(X_1 = x_1, \ldots, X_n = x_n, Y_1 = y_1, \ldots, Y_n = y_n)$$

where $x_i \in \Sigma$, $y_i \in \mathcal{T}$. For brevity, we will write it as $p(x_1, \ldots, x_n, y_1, \ldots, y_n)$, i.e., treat it as a function of values of the random variables without explicating the variables themselves. We will define one additional random variable Y_{n+1}, which always takes the value STOP. Since our model is over variable length sequences, we will use the end symbol to model when to stop. In other words, if we observe $x_1, \ldots, x_n$, then clearly the symbol after $y_1, \ldots, y_n$, i.e., y_{n+1}, had to be STOP (otherwise we would have continued generating more symbols).

Now, let's start by rewriting the distribution a bit according to general rules that apply to any distribution. The goal is to put the distribution in a form where we can easily explicate our assumptions. First,

$$p(x_1, \ldots, x_n, y_1, \ldots, y_{n+1}) = p(y_1, \ldots, y_{n+1}) p(x_1, \ldots, x_n | y_1, \ldots, y_{n+1}) \qquad \text{(chain rule)}$$

Then we will use the chain rule repeatedly along the sequence of tags

$$p(y_1, \ldots, y_{n+1}) = p(y_1) p(y_2|y_1) p(y_3|y_1, y_2) \ldots p(y_{n+1}|y_1, \ldots, y_n) \qquad \text{(chain rule)}$$

So far, we have made no assumptions about the distribution at all. Since we don't expect tags to have very long dependences along the sequence, we will simply say that the next

tag only depends on the current tag. In other words, we will "drop" the dependence on tags further back

$$\begin{aligned} p(y_1, \ldots, y_{n+1}) &\approx p(y_1)p(y_2|y_1)p(y_3|y_2)\ldots p(y_{n+1}|y_n) \quad \text{(independence assumption)} \\ &= \prod_{i=1}^{n+1} p(y_i|y_{i-1}) \end{aligned}$$

Put another way, we assume that the tags form a Markov sequence (future tags are independent of the past tags given the current one). Let's now make additional assumptions about the observations as well

$$\begin{aligned} p(x_1, \ldots, x_n|y_1, \ldots, y_{n+1}) &= \prod_{i=1}^{n} p(x_i|x_1, \ldots, x_{i-1}, y_1, \ldots, y_{n+1}) && \text{(chain rule)} \\ &\approx \prod_{i=1}^{n} p(x_i|y_i) && \text{(independence assumption)} \end{aligned}$$

In other words, we say that the identity of each word only depends on the corresponding tags. This is a drastic assumption but still (often) leads to a reasonable tagging model, and simplifies our calculations. A more formal statement here is that the random variable X_i is conditionally independent of all the other variables in the model given Y_i (see more on conditional independence in the Bayesian networks lectures).

Now, we have a much simpler tagging model

$$p(x_1, \ldots, x_n, y_1, \ldots, y_{n+1}) = p(y_1) \prod_{i=2}^{n+1} p(y_i|y_{i-1}) \prod_{i=1}^{n} p(x_i|y_i)$$

For notational convenience, we also assume a special fixed START symbol $y_0 = *$ so that $p(y_1)$ becomes $p(y_1|y_0)$. As a result, we can write

$$p(x_1, \ldots, x_n, y_1, \ldots, y_{n+1}) = \prod_{i=1}^{n+1} p(y_i|y_{i-1}) \prod_{i=1}^{n} p(x_i|y_i)$$

Let's understand this model a bit more carefully by looking at how the pairs of sequences could be generated from the model. Here's the recipe

1. Set $y_0 = *$ (we always start from the START symbol) and let $i = 1$.
2. Generate tag y_i from the conditional distribution $p(y_i|y_{i-1})$ where y_{i-1} already has a value (e.g., $y_0 = *$ when $i = 1$)
3. If $y_i =$ STOP, we halt the process and return $y_1, \ldots, y_i, x_1, \ldots, x_{i-1}$. Otherwise we generate x_i from the output/emission distribution $p(x_i|y_i)$
4. Set $i = i + 1$, and return to step 2.

HMM formal definition

The model we have defined is a Hidden Markov Model or HMM for short. An HMM is defined by a tuple $\langle N, \Sigma, \theta \rangle$, where

- N is the number of states $1, \ldots, N$ (assume the last state N is the final state, i.e. what we called "STOP" earlier).
- Σ is the alphabet of output symbols. For example, $\Sigma = \{$"the", "dog"$\}$.
- $\theta = \langle a, b, \pi \rangle$ consists of three sets of parameters
  - Parameter $a_{i,j} = p(y_{next} = j | y = i)$ for $i = 1, \ldots, N-1$ and $j = 1, \ldots, N$ is the probability of transitioning from state i to state j: $\sum_{k=1}^{N} a_{i,k} = 1$
  - Parameter $b_j(o) = p(x = o | y = j)$ for $j = 1 \ldots N-1$ and $o \in \Sigma$ is the probability of emitting symbol o from state j: $\sum_{o \in \Sigma} b_j(o) = 1$.
  - Parameter $\pi_i = p(y_1 = i)$ for $i = 1 \ldots N$ specifies probability of starting at state i: $\sum_{i=1}^{N} \pi_i = 1$.

(Question 1.1) [1 points] What is the dimensionality of θ as a function of N and Σ?

Example:

- N = 3. States are $\{1, 2, 3\}$
- Alphabet $\Sigma = \{the, dog\}$
- Distribution over initial states: $\pi_1 = 1, \pi_2 = \pi_3 = 0$.
- Parameters $a_{i,j}$ are

	$j=1$	$j=2$	$j=3$
$i=1$	0.5	0.5	0
$i=2$	0	0.8	0.2

- Parameters $b_j(o)$ are

	o =the	o =dog
$i=1$	0.9	0.1
$i=2$	0.1	0.9

An HMM specifies a probability for each possible (x, y) pair, where $x = (x_1, \ldots, x_n)$ is a sequence of symbols drawn from Σ and $y = (y_1, \ldots, y_n)$ is a sequence of states drawn from the integers $1, \ldots, (N-1)$.

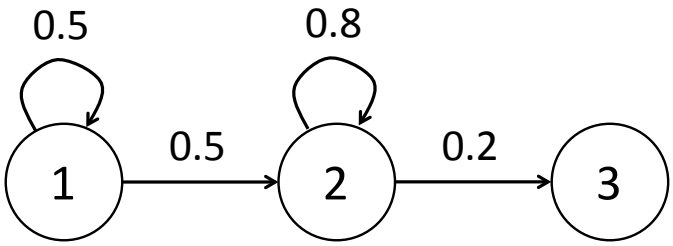

Figure 1: Transition graph for the example.

$$
\begin{aligned}
p(x, y|\theta) = & \pi_{y_1} \cdot && \text{(prob. of choosing } y_1 \text{ as an initial step)} \\
& a_{y_n, N} \cdot && \text{(prob. of transitioning to the final step)} \\
& \prod_{i=2}^{n} a_{y_{i-1}, y_i} \cdot && \text{(transition probability)} \\
& \prod_{i=1}^{n} b_{y_i}(x_i) && \text{(emission probability)}
\end{aligned}
$$

(Question 1.2) [1 points] Compute the probability of the following sequence: "the/1, dog/2, the/1".

Parameter estimation

We will first look at the fully observed case (complete data case), where our training data contains both xs and ys. We will do maximum likelihood estimation. Consider the examples: $\Sigma = \{e, f, g, h\}, N = 3$. Observation: $(e/1, g/2), (e/1, h/2), (f/1, h/2), (f/1, g/2)$. To find the MLE, we will simply look at the counts of events, i.e., the number of transitions between tags, the number of times we saw an output symbol together with an specific tag (state). After normalizing the counts to yield valid probability estimates, we get

$$a_{i,j} = \frac{count(i, j)}{count(i)}$$

$$a_{1,2} = \frac{count(1, 2)}{count(1)} = \frac{4}{4} = 1, \; a_{2,2} = \frac{count(2, 2)}{count(2)} = \frac{0}{4} = 0, \ldots$$

where $count(i, j)$ is the number of times we have (i, j) as two successive states and $count(i)$ is the number of times state i appears in the sequence. Similarly,

$$b_i(o) = \frac{count(i \to o)}{count(i)}$$

$$b_1(e) = \frac{count(1 \to e)}{count(1)} = \frac{2}{4} = 0.5, \ldots$$

where $count(i \to o)$ is the number of times we see that state i emits symbol o. $count(i)$ was already defined above.

2 Decoding with HMM

Suppose now that we have the HMM parameters θ (see above) and the problem is to infer the underlying tags $y_1, \ldots, y_n$ corresponding to an observed sequence of words $x_1, \ldots, x_n$. In other words, we wish to evaluate

$$\text{argmax}_{y_1,\ldots,y_n}\, p(x_1, \ldots, x_n, y_1, \ldots, y_{n+1})$$

where

$$p(x_1, \ldots, x_n, y_1, \ldots, y_{n+1}) = \prod_{i=1}^{n+1} a_{y_{i-1}, y_i} \prod_{i=1}^{n} b_{y_i}(x_i)$$

and $y_0 = *, y_{n+1} =$ STOP. Note that by defining a fixed starting state, we have again folded the initial state distribution π into the transition probabilities $\pi_i = a_{*,i}$, $i \in \{1, \ldots, N\}$ (where $N =$STOP).

One possible solution for finding the most likely sequence of tags is to do brute force enumeration. Consider the example: $\Sigma =$ {the, dog}, $x =$"the the the dog". The possible state sequences include:

1 1 1 2 STOP

1 1 2 2 STOP

1 2 2 2 STOP

$\vdots$

(Question 2.1) [1 points] How many possible tag sequences can generate string of length n, assuming the size of tag alphabet to be $|\mathcal{T}|$?

Solving the tagging problem by enumerating the tag sequences will be prohibitively expensive.

Viterbi algorithm

The HMM has a simple dependence structure (recall, tags form a Markov sequence, observations only depend on the underlying tag). We can exploit this structure in a dynamic programming algorithm.

Input: $x = x_1, \ldots, x_n$ and model parameters θ.

Output: $\text{argmax}_{y_1,\ldots,y_{n+1}} p(x_1, \ldots, x_n, y_1, \ldots, y_{n+1})$.

Now, let's look at a truncated version of the joint probability, focusing on the first k tags for any $k \in \{1, \ldots, n\}$. In other words, we define

$$r(y_1, \ldots, y_k) = \prod_{i=1}^{k} a_{y_{i-1},y_i} \prod_{i=1}^{k} b_{y_i}(x_i)$$

where y_k does not equal STOP. Note that our notation $r(y_1, \ldots, y_k)$ suppresses any dependence on the observation sequence. This is because we view $x_1, \ldots, x_n$ as given (fixed). Note that, according to our definition,

$$\begin{aligned} p(x_1, \ldots, x_n, y_1, \ldots, y_{n+1}) =& r(y_1, \ldots, y_n) \cdot a_{y_n,y_{n+1}} \\ =& r(y_1, \ldots, y_n) \cdot a_{y_n,\text{STOP}} \end{aligned}$$

Let $S(k, v)$ be the set of tag sequences $y_1, \ldots, y_k$ such that $y_k = v$. In other words, $S(k, v)$ is a set of all sequences of length k whose last tag is v. The dynamic programming algorithm will calculate

$$\pi(k, v) = \max_{(y_1,\ldots,y_k) \in S(k,v)} r(y_1, \ldots, y_k)$$

recursively in the forward direction. In other words, $\pi(k, v)$ can be thought as solving the maximization problem partially, over all the tags $y_1, \ldots, y_{k-1}$ with the constraint that we use tag v for y_k. If we have $\pi(k, v)$, then $\max_v \pi(k, v)$ evaluates $\max_{y_1,\ldots,y_k} r(y_1, \ldots, y_k)$. We leave v in the definition of $\pi(k, v)$ so that we can extend the maximization one step further as we unravel the model in the forward direction. More formally,

- Base case reflects the assumption that $y_0 = *$.

(Question 2.2) [1 points] Complete the following entries:

$$\begin{aligned} \pi(0, *) =& \quad \text{(starting state, no observations)} \\ \pi(0, v) =& \quad \text{if } v \neq * \text{ (an actual state has observations)} \end{aligned}$$

- Moving forward recursively: for any $k \in \{1, \ldots, n\}$

$$\pi(k, v) = \max_{u \in \mathcal{T}} \{\pi(k-1, u) \cdot a_{u,v} \cdot b_v(x_k)\}$$

  In other words, when extending $\pi(k-1, u)$, $u \in \mathcal{T}$, to $\pi(k, v)$, $v \in \mathcal{T}$, we must transition from $y_{k-1} = u$ to $y_k = v$ (part $a_{u,v}$) and generate the corresponding observation x_k (part $b_v(x_k)$). Then we maximize over the previous tag $y_{k-1} = u$ so that $\pi(k, v)$ only depends on the value of y_k.

  (Question 2.3) [5 points] Provide detailed derivations for the Viterbi iteration, i.e., show why we have

$$\pi(k, v) = \max_{u \in \mathcal{T}} \{\pi(k-1, u) \cdot a_{u,v} \cdot b_v(x_k)\}$$

Finally, we must transition from y_n to STOP so that

$$\max_{y_1, \ldots, y_n} p(x_1, \ldots, x_n, y_1, \ldots, y_n, y_{n+1} = \text{STOP}) = \max_{v \in \mathcal{T}} \{\pi(n, v) \cdot a_{v,\text{STOP}}\}$$

Now, having values $\pi(k, v)$, how do we reconstruct the most likely sequence of tags which we denote as $\hat{y}_1, \ldots, \hat{y}_n$? We can do this via *back-tracking*. In other words, at the last step, $\pi(n, v)$ represents maximizations of all $y_1, \ldots, y_n$ such that $y_n = v$. What is the best value for this last tag v, i.e., what is $\hat{y}_n$? It is

$$\hat{y}_n = \text{argmax}_v \left\{\pi(n, v) a_{v,\text{STOP}}\right\}$$

Now we can fix $\hat{y}_n$ and work backwards. What is the best value $\hat{y}_{n-1}$ such that we end up with tag $\hat{y}_n$ in position n? It is simply

$$\hat{y}_{n-1} = \text{argmax}_u \left\{\pi(n-1, u) a_{u,\hat{y}_n}\right\}$$

and so on.

(Question 2.4) [1 points] What is time complexity of Viterbi algorithm, as a function of tag set size $|\mathcal{T}|$ and sentence length n?

3 Hidden Variable Problem

When we no longer have the tags, we must resort to other ways of estimating the HMMs. It is not trivial to construct a model that agrees with the observations except in very simple scenarios. Here is one:

- We have an HMM with $N = 3, \Sigma = \{a, b, c\}$

- We see the following output sequences in training data: $(a, b), (a, c), (a, b)$.

How would you choose the parameter values for $\pi_i, a_{i,j}$ and $b_i(o)$? A reasonable choice is:

$$\pi_1 = 1.0, \pi_2 = \pi_3 = 0$$
$$b_1(a) = 1.0, b_1(b) = b_1(c) = 0$$
$$b_2(a) = 0, b_2(b) = b_2(c) = 0.5$$
$$a_{1,2} = 1.0, a_{1,1} = a_{1,3} = 0$$
$$a_{2,3} = 1.0, a_{2,1} = a_{2,2} = 0$$

Expectation-Maximization (EM) for HMM

Suppose now that we have multiple observed sequences of outputs (no observed tags). We will denote these sequences with superscripts, i.e., $x^1, x^2, \ldots, x^m$. In the context of each sequence, we must evaluate a posterior probability over possible tag sequences. For estimation, we only need expected counts that are used in the re-estimation step (M-step). To this end, let count$(x^i, y, p \to q)$ be the numbers of times a transition from state p to state q occurs in a tag sequence y corresponding to observation x^i. We will only show here the derivations for transition probabilities; the equations for emission and initial state parameters are obtained analogously.

E-step: calculate expected counts, added across sequences

(Question 3.1) [4 points] Complete the following expression:

$$\overline{count}(u \to v) =$$

M-step: re-estimate transition probabilities based on the expected counts

(Question 3.2) [4 points] Complete the following expression:

$$a_{u,v} =$$

The main problem in running the EM algorithm is calculating the sum over the possible tag sequences in the E-step.

The sum is over an exponential number of possible hidden state sequences y. Next we will discuss a dynamic programming algorithm – forward-backward algorithm. The algorithm is analogous to the Viterbi algorithm for maximizing over the hidden states.

The Forward-Backward Algorithm for HMMs

Suppose we could efficiently calculate marginal posterior probabilities

$$p(y_j = p, y_{j+1} = q|x, \theta) = \sum_{y:y_j=p,y_{j+1}=q} p(y|x, \theta)$$

for any $p \in \{1, \ldots, (N-1)\}, q \in \{1, \ldots, N\}, j \in \{1, \ldots, n\}$. These are the posterior probabilities that the state in position j was p and we transitioned into q at the next step. The probability is conditioned on the observed sequence x and the current setting of the model parameters θ. Now, under this assumption, we could rewrite the difficult computation of fractional counts as:

$$\sum_y p(y|x^i, \theta^{t-1}) count(x^i, y, p \to q) = \sum_{j=1}^{n} p(y_j = p, y_{j+1} = q|x^i, \theta^{t-1})$$

The key remaining question is how to calculate these posterior marginals effectively. In other words, our goal is to evaluate $p(y_j = p, y_{j+1} = q|x^i, \theta^{t-1})$.

Now, consider a single observation sequence $x_1, \ldots, x_n$. We will make use of the following forward probabilities:

$$\alpha_p(j) = p(x_1, \ldots, x_{j-1}, y_j = p|\theta)$$

for all $j \in \{1, \ldots, n\}$, for all $p \in \{1, \ldots, N-1\}$. $\alpha_p(j)$ is the probability of emitting the symbols $x_1, \ldots, x_{j-1}$ and ending in state p in position j without (yet) emitting the corresponding output symbol. These are analogous to the $\pi(k, v)$ probabilities in the Viterbi algorithm with the exception that $\pi(k, v)$ included generating the corresponding observation in position k. Note that, unlike before, we are summing over all the possible sequences of states that could give rise to the observations $x_1, \ldots, x_{j-1}$. In the Viterbi algorithm, we maximized over the tag sequences.

Similarly to the forward probabilities, we can define the backward probabilities:

$$\beta_p(j) = p(x_j, \ldots, x_n|y_j = p, \theta)$$

for all $j \in \{1, \ldots, n\}$, for all $p \in \{1, \ldots, N-1\}$. $\beta_p(j)$ is the probability of emitting symbols $x_j, \ldots, x_n$ and transitioning into the final (STOP) state, given that we begun in state p in position j. Again, this definition involves summing over all the tag sequences that could have generated the observations from x_j onwards, provided that the tag at j is p.

Why are these two definitions useful? Suppose we had been able to evaluate α and β probabilities effectively. Then the marginal probability we were after could be calculated as:

$$p(y_j = p, y_{j+1} = q|x, \theta) = \frac{1}{Z} \alpha_p(j) a_{p,q} b_p(o_j) \beta_q(j+1)$$

$$Z = p(x_1, \ldots, x_n|\theta) = \sum_p \alpha_p(j) \beta_p(j) \text{ for any } j = 1 \ldots n$$

This is just the sum over all possible tag sequences that include the transition $y_j = p$ and $y_{j+1} = q$ and generates the observations, divided by the sum over all tag sequences that generate the observations. As a result, we obtain the relative probability of the transition, relative to all the alternatives given the observations, i.e., the posterior probability. Note that $\alpha_p(j)$ involves all the summations over tags $y_1, \ldots, y_{j-1}$, and $\beta_q(j+1)$ involves all the summations over the tags $y_{j+2}, \ldots, y_n$.

Let's finally discuss how we can calculate α and β.

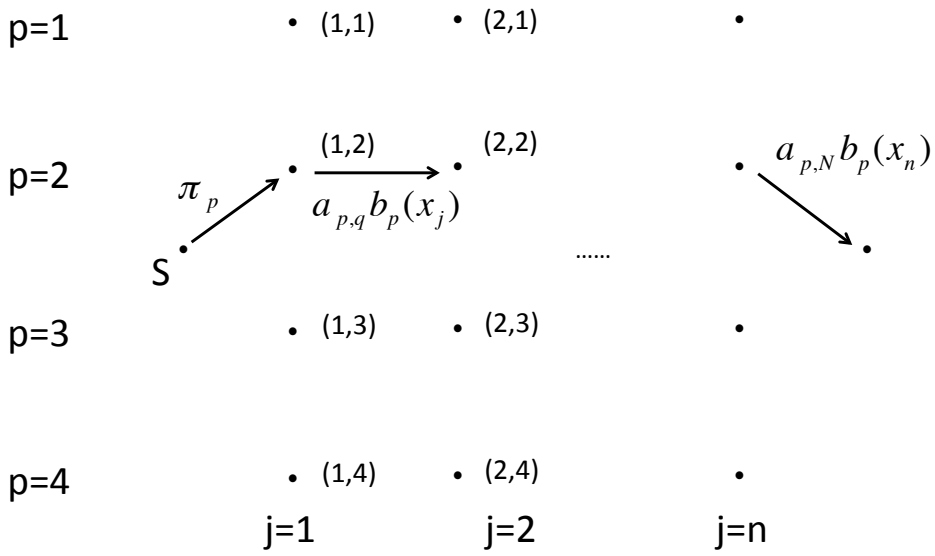

Figure 2: A path associated with state sequence.

As Fig. 2 shows, for every state sequence $y_1, y_2, \ldots, y_n$ there is

- a path through graph that has the sequence of states $s, \langle 1, y_1\rangle, \ldots, \langle n, y_n\rangle, f$.
- The path associated with state sequence $y_1, \ldots, y_n$ has weight equal to $p(x, y|\theta)$.
- $\alpha_p(j)$ is the sum of weights at all paths from s to the state (j, p).
- $\beta_p(j)$ is the sum of weights at all paths from state $\langle j, p\rangle$ to the final state f.

Given an input sequence $x_1, \ldots, x_n$, for any $p \in \{1, \ldots, N\}, j \in \{1, \ldots, n\}$, the forward and backward probability can be calculated recursively.

Forward probability:

$$\alpha_p(j) = p(x_1, \ldots, x_{j-1}, y_j = p|\theta)$$

- Base case:

$$\alpha_p(1) = \pi_p, \quad \forall p \in \{1, \ldots, N-1\}$$

- Recursive case

$$\alpha_p(j+1) = \sum_q \alpha_q(j) a_{q,p} b_q(x_j) \quad \forall p \in \{1, \ldots, N-1\}, j \in \{1, \ldots, n-1\}$$

Backward probability:

$$\beta_p(j) = p(x_j, \ldots, x_n | y_j = p, \theta)$$

- Base case:

$$\beta_p(n) = a_{p,N} b_p(x_n), \quad \forall p \in \{1, \ldots, N-1\}$$

- Recursive case

$$\beta_p(j) = \sum_q a_{p,q} b_p(x_j) \beta_q(j+1), \quad \forall p \in \{1, \ldots, N-1\}, j \in \{1, \ldots, n-1\}$$

(Question 3.3) [2 points] Show that $p(x_1, \ldots, x_n|\theta) = \sum_p \alpha_p(i)\beta_p(i)$ for any $i \in \{1, \ldots, n\}$.

(Question 3.4) [5 points] For $i \in \{1, \ldots, n-1\}$ and $p \in \{1, \ldots, N-1\}$, show that

$$p(y_i = p | x_1, \ldots, x_n, \theta) = \frac{\alpha_p(i)\beta_p(i)}{\sum_q \alpha_q(i)\beta_q(i)} .$$

Problem: EM Algorithm for Language Model Smoothing

In this problem, we consider smoothing language models via linear interpolation. It turns out that estimating the associated linear weights is a problem that EM can solve very well. Recall that in linear interpolation, the desired smoothed trigram model is written as a combination of maximum likelihood estimates of trigram, bigram, and unigram models as follows

$$p_\Lambda(w|u,v) = \lambda_3 p_{\text{ML}}(w|u,v) + \lambda_2 p_{\text{ML}}(w|u) + \lambda_1 p_{\text{ML}}(w)$$

where the interpolation weights $\Lambda = \{\lambda_1, \lambda_2, \lambda_3\}$ satisfy $\lambda_y \geq 0$ and $\sum_{y=1}^3 \lambda_y = 1$. Here $p_{\text{ML}}(w|u,v)$, $p_{\text{ML}}(w|v)$, and $p_{\text{ML}}(w)$ are the maximum likelihood estimates of the trigram, bigram, and unigram models, respectively, estimated on the basis of the training corpus. The interpolation weights, however, are obtained by maximizing the log-likelihood that the smoothed model $p_\Lambda(w|u,v)$ assigns to the words in the development set, not training set. Note that the p_{ML} estimates remain unchanged during this process.

Let (w_{t-2}, w_{t-1}, w_t) specify the t^{th} trigram in the development set D. w_{t-2} and w_{t-1} are set to specific start symbols STA_{-2} and STA_{-1} whenever the trigram appears in the

beginning of a sentence. In this case, the log-likelihood of the words in the development set can be written as

$$l(\mathrm{D};\Lambda) = \sum_{t=1}^{T} \log p_\Lambda(w_t|w_{t-1},w_{t-2}) \tag{1}$$

$$= \sum_{t=1}^{T} \log \Big(\lambda_3 p_{\mathrm{ML}}(w_t|w_{t-1},w_{t-2}) + \lambda_2 p_{\mathrm{ML}}(w_t|w_{t-1}) + \lambda_1 p_{\mathrm{ML}}(w_t)\Big) \tag{2}$$

You are asked to use the EM algorithm to find the maximum likelihood estimates (MLEs) of parameters Λ based on D, i.e., to find

$$\Lambda^* = \arg\max_{\Lambda} \log l(D;\Lambda)\,.$$

(Question 4.1) [4 points] Suppose we deviated from the procedure and instead estimated the interpolation weights on the basis of the same training corpus from which $p_{\mathrm{ML}}(w|u,v)$, $p_{\mathrm{ML}}(w|u)$, and $p_{\mathrm{ML}}(w)$ are derived. Which setting of the weights Λ in $p_\Lambda(w|u,v)$ maximize the log-likelihood of the words in the training corpus? Briefly justify your answer.

(Question 4.2) [5+3 = 8 points] Now we can make use of the EM algorithm to estimate the parameters Λ based on the development set. Let $\Lambda^{(k)} = \{\lambda_1^{(k)}, \lambda_2^{(k)}, \lambda_3^{(k)}\}$ be the parameters after the kth EM iteration.

(a) *E-Step:* Provide expressions for the fractional counts $\hat{n}(\lambda_1)$, $\hat{n}(\lambda_2)$, and $\hat{n}(\lambda_3)$.

(b) *M-Step:* Now estimate the new values $\lambda_y^{(k+1)}$, $y = 1, 2, 3$ using the above fractional counts.

(Question 4.3) [3 points] Let $\Lambda^{(0)} = \{\lambda_1^{(0)}, \lambda_2^{(0)}, \lambda_3^{(0)}\}$ be the initial setting for the EM algorithm. Explain what will happen during the EM iterations if $\lambda_2^{(0)} = 0$.

Problem: EM Algorithm for Topic Models

Topic models are useful for discovering patterns in text and creating groups of words called *topics*. Recall from class that a latent topic model aims to assign a topic $z \in [1, K]$ to each word w in a given set of documents $\{d_t : t \in [1, n]\}$. We saw that the posterior probability can be modeled as:

$$p(z|w,t) = \frac{\theta_{z|t}\theta_{w|z}}{\sum_{z'=1}^{k} \theta_{z'|t}\theta_{w|z'}}$$

where $\theta_{z|t}$ and $\theta_{w|z}$ are the parameters of the topic model representing the conditional probabilities $p(z|t)$ and $p(w|z)$, respectively.

(Question 5.1) [4 points] *E-Step:* Write down the expressions for the soft counts (or fractional counts) $\hat{n}_t(z)$ and $\hat{n}(w, z)$ using the above posterior probability. You can assume the number of documents to be n and the number of words in document t to be N_t.

(Question 5.2) [4 points] *M-Step:* Using these soft counts, provide the update equations for $\theta_{z|t}$ and $\theta_{w|z}$ using MLE.

(Question 5.3) [17 points] *Programming:* The questions in this part require you to work with the Python code template provided in the folder topic_model_em.

(a) Using the expressions you derived in (1) and (2), complete the sections marked with a *TODO* in the code template to obtain a working implementation of EM for topic modeling. You can test it on the sample data included.

(b) Using the code, experiment with various settings of number of topics and number of iterations to obtain "good" topics. Can you think of a way to determine the quality of a topic obtained from the model?

(c) Run the model with 10 topics for 50 iterations. Interpret the "topics" that you obtain (you can provide a list of the top words in each topic and a single word description of the topic). Are all the topics unique? Provide a short (one or two sentence) explanation for your observations.

(d) In the code, you will have noticed that the parameters (θ) of the model are initialized randomly at the start of the EM algorithm. What do you expect will happen if you initialize the parameters uniformly? You can verify your intuitions using the code (only your reasoning is required as an answer though).

References

[1]. Lawrence R. Rabiner. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition." Proceedings of the IEEE, 77(2), Feb. 1989.