6.864, Fall 2005: Problem Set 4

Total points: 160 regular points

Due date: 5pm, 15 November 2005

Late policy: 5 points off for every day late, 0 points if handed in after 5pm on 19 November 2005

Question 1 (15 points)

Describe an algorithm for hierarchical topic segmentation of a text. In contrast to linear segmentation algorithms we studied in class, your algorithm should not only find topic changes, but it has to find nesting relations among different topics. Specify the parameters of your algorithm, and suggest a method for their estimation.

Question 2 (15 points)

The min-cut segmentation algorithm presented in class computes text segmentation based on changes in lexical distribution. We would like to refine this algorithm by taking into account the length of computed segments. The function $cost(l)$ captures the penalty associated with a segment of length l — the assumption is that some values of segment length are more likely than others.

- There are several ways to specify $cost(l)$. Provide a definition for $cost(l)$, and explain your choice.
- Explain how to modify the definition of the K-way partitioning cost to take the length penalty $cost(l)$ into account. Similarly to the original definition, the modified partitioning cost has to be defined recursively.

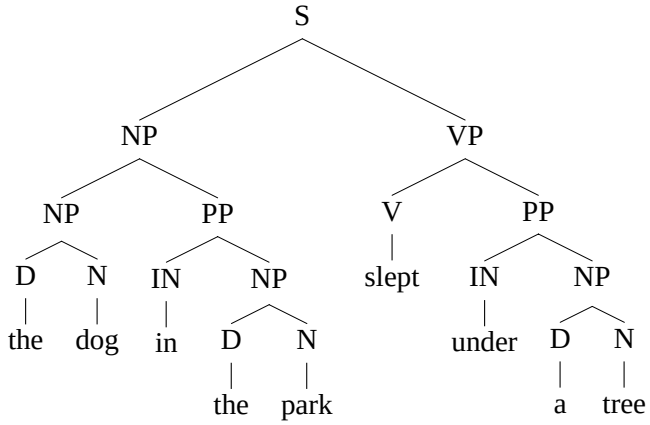
Question 3 (15 points)

We are going to come up with a modified version of the Viterbi algorithm for trigram taggers. Assume that the input to the Viterbi algorithm is a word sequence $w_1 \dots w_n$. For each word in the vocabulary, we have a *tag dictionary* $T(w)$ that lists the tags t such that $P(w|t) > 0$. Take K to be a constant such that $|T(w)| \leq K$ for all w. Give pseudo-code for a version of the Viterbi algorithm that runs in $O(nK^3)$ time where n is the length of the input sentence.

Question 4 (15 points)

In lecture we saw how the Ratnaparkhi parser maps a parse tree to a sequence of *decisions* in a history-based model. For example, see slide 70 of lecture 10 for the sequence of decisions associated with the parse tree on slide 49 of the lecture.

Question: Show the sequence of decisions that is used to represent the following parse tree:



Question 5 (100 points)

In this programming question, we are going to build a trigram HMM tagger. The joint probability of a word sequence $w_1, w_2, \ldots w_n$ and a tag sequence $t_1, t_2, \ldots, t_n$ is defined as

$$P(w_1 \ldots w_n, t_1 \ldots t_n) = P(\texttt{\#END}|t_{n-2}, t_{n-1}) \prod_{i=1}^{n} P(t_i|t_{i-2}, t_{i-1}) \prod_{i=1}^{n} P(w_i|t_i)$$

The Viterbi algorithm searches for

$$\arg\max_{t_1 \ldots t_n} P(w_1 \ldots w_n, t_1 \ldots t_n)$$

for a given word sequence $w_1 \ldots w_n$.

In the file `poscounts.gz` you will find counts that will allow you to estimate the parameters of the model. There are 4 types of counts in this file:

- Lines where the second token is `DENOM`, for example

  ```
  124953 DENOM NN
  ```

  These are the counts used in the denominators of any maximum likelihood estimates in the model. For example, in this case the count of the unigram tag `NN` is 124952.

- Lines where the second token is `NUMER`, for example

  ```
  32545 NUMER NNP NNP
  ```

  These are the counts used in the numerators of any maximum likelihood estimates in the model. For example, in this case the count of the tag bigram `NNP NNP` is 32545.

- Lines where the second token is `WORDTAG1`, for example

  `38612 WORDTAG1 DT the`

  These are counts used in the numerators of maximum-likelihood estimates of $P(w_i|t_i)$. For example, the word `the` is seen tagged 38612 times as `DT` in this case.

- Lines where the second token is `WORDTAG2`, for example

  `77079 WORDTAG2 DT`

  These are counts used in the denominators of maximum-likelihood estimates of $P(w_i|t_i)$. For example, the tag `DT` is seen 77079 times, according to this count.

Some further notes:

- Each sentence has $t_{-2} = \#\texttt{S1}$, $t_{-1} = \#\texttt{S2}$, and $t_{n+1} = \#\texttt{END}$. Hence some of the n-grams will include these symbols.
- Words occurring less than 5 times in training data have been mapped to the word token `UNKA`.

Part 1: Estimating $P(w|t)$ parameters. You should write a function that returns $P(w|t)$ for a particular word w and tag t, where

$$P(w|t) = \frac{Count(w,t)}{Count(t)}$$

The counts in this case are taken from the `WORDTAG1` and `WORDTAG2` lines in poscounts.gz.

Part 2: Estimating $P(t_i|t_{i-2}, t_{i-1})$ parameters. You should write a function that returns $P(t_i|t_{i-2}, t_{-1})$ for a particular tag trigram t_{i-2}, t_{i-1}, t_i. This estimate should be defined as follows:

`If` $Count_d(t_{i-2}, t_{i-1}) > 0$

Return $\frac{1}{3}P_{ML}(t_i|t_{i-2}, t_{i-1}) + \frac{1}{3}P_{ML}(t_i|t_{i-1}) + \frac{1}{3}P_{ML}(t_i)$

`Else if` $Count_d(t_{i-1}) > 0$

Return $\frac{1}{2}P_{ML}(t_i|t_{i-1}) + \frac{1}{2}P_{ML}(t_i)$

`Else`

Return $P_{ML}(t_i)$

Here $Count_d$ are the denominator counts (lines with `DENOM` in the `poscounts.gz` file). The P_{ML} estimates are defined as

$$P_{ML}(t_i|t_{i-1}, t_{i-2}) = \frac{Count_n(t_{i-2}, t_{i-1}, t_i)}{Count_d(t_{i-2}, t_{i-1})}$$

$$P_{ML}(t_i|t_{i-1}) = \frac{Count_n(t_{i-1}, t_i)}{Count_d(t_{i-1})}$$

$$P_{ML}(t_i) = \frac{Count_n(t_i)}{Count_d()}$$

where $Count_n$ are the counts from the `NUMER` lines in the `poscounts.gz` file. Note that you can get $Count_d()$ from the following line of the file:

950563 DENOM BLANK

Note: make sure your code has the following functionality. To test the code it should be possible to read in a file, line by line, that contains one tag trigram per line. For example, the file might contain

NN NN DT
NN DT NN

As output, the code should write the probability for each trigram in turn, for example

0.054
0.032

Part 3: Implementing the Viterbi algorithm

You should now implement a version of the Viterbi algorithm, which searches for the most probable sequence of tags under the model.

Note: make sure your code has the following functionality. To test the code it should be possible to read in a file, line by line, that contains one sentence per line. To see a test file, look at

wsj.19-21.test

For each line, your code should

(1) print the most likely sequence of tags under the model

(2) print the log-probability of this sequence of tags

Note: ***The sentences in this input file already have infrequent words replaced with the*** UNKA ***token***

One important note about the efficiency of your implementation: for each word in the vocabulary, you should compile a "tag dictionary" that lists the set of tags that have been seen at least once with that word. You should use this fact to make a more efficient algorithm (see question 3 of this problem set).