6.864, Fall 2007: Problem Set 2

Total points: 195 points

Due date: 5pm, 18th October 2007

Submit to Igor Malioutov either by email to `igorm@csail.mit.edu`, or by hand to Stata G360

Late policy: 5 points off for every day late, 0 points if handed in after 5pm on 23rd October

Question 1 (20 points)

Clarissa Linguistica decides to build a log-linear model for language modeling. She has a training sample (x_i, y_i) for $i = 1 \ldots n$, where each x_i is a prefix of a document (e.g., $x_i =$ "Yesterday, George Bush said") and y_i is the next word seen after this prefix (e.g., $y_i =$ "that"). As usual in log-linear models, she defines a function $\mathbf{f}(x, y)$ that maps any x, y pair to a vector in $\mathbb{R}^d$. Given parameter values $\mathbf{v} \in \mathbb{R}^d$, the model defines

$$P(y|x, \mathbf{v}) = \frac{e^{\mathbf{v} \cdot \mathbf{f}(x,y)}}{\sum_{y' \in \mathcal{V}} e^{\mathbf{v} \cdot \mathbf{f}(x,y')}}$$

where $\mathcal{V}$ is the *vocabulary*, i.e., the set of possible words; and $\mathbf{v} \cdot \mathbf{f}(x, y)$ is the inner product between the vectors $\mathbf{v}$ and $\mathbf{f}(x, y)$.

Given the training set, the training procedure returns parameters $\mathbf{v}^* = \arg\max_{\mathbf{v}} L(\mathbf{v})$, where

$$L(\mathbf{v}) = \sum_i \log P(y_i|x_i, \mathbf{v}) - C \sum_k v_k^2$$

and $C > 0$ is some constant.

Clarissa makes the following choice of her first two features in the model:

$$f_1(x, y) = \begin{cases} 1 & \text{if } y = \texttt{model} \text{ and previous word in } x \text{ is } \texttt{the} \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(x, y) = \begin{cases} 1 & \text{if } y = \texttt{model} \text{ and previous word in } x \text{ is } \texttt{the} \\ 0 & \text{otherwise} \end{cases}$$

So $f_1(x, y)$ and $f_2(x, y)$ are *identical features*.

Question (10 points): Show that for any training set, with f_1 and f_2 defined as above, the optimal parameters $\mathbf{v}^*$ satisfy the property that $v_1^* = v_2^*$.

Question (10 points): Now say we define the optimal parameters to be $\mathbf{v}^* = \arg\max_{\mathbf{v}} L(\mathbf{v})$, where

$$L(\mathbf{v}) = \sum_i \log P(y_i|x_i, \mathbf{v}) - C \sum_k |v_k|$$

and $C > 0$ is some constant. (Here $|v_k|$ is the absolute value of the k'th feature.) In this case, does the property $v_1^* = v_2^*$ necessarily hold? If not, what constraints do hold for the values v_1^* and v_2^*?

Question 2 (15 points)

Nathan L. Pedant now decides to build a bigram language model using log-linear models. He gathers a training sample (x_i, y_i) for $i = 1 \ldots n$. Given a vocabulary of words $\mathcal{V}$, each x_i and each y_i is a member of

$\mathcal{V}$. Each (x_i, y_i) pair is a *bigram* extracted from the corpus, where the word y_i is seen following x_i in the corpus.

Nathan's model is similar to Clarissa's, except he chooses the optimal parameters $\mathbf{v}^*$ to be $\arg\max L(\mathbf{v})$ where

$$L(\mathbf{v}) = \sum_i \log P(y_i|x_i, \mathbf{v})$$

The features in his model are of the following form:

$$f_i(x, y) = \begin{cases} 1 & \text{if } y = \texttt{model} \text{ and } x = \texttt{the} \\ 0 & \text{otherwise} \end{cases}$$

i.e., the features track pairs of words. To be more specific, he creates one feature of the form

$$f_i(x, y) = \begin{cases} 1 & \text{if } y = w_2 \text{ and } x = w_1 \\ 0 & \text{otherwise} \end{cases}$$

for every (w_1, w_2) in $\mathcal{V} \times \mathcal{V}$.

Question (15 points): Assume that the training corpus contains all possible bigrams: i.e., for all $w_1, w_2 \in \mathcal{V}$ there is some i such that $x_i = w_1$ and $y_i = w_2$. The optimal parameter estimates $\mathbf{v}^*$ define a probability $P(y = w_2|x = w_1, \mathbf{v}^*)$ for any bigram w_1, w_2. Show that for any w_1, w_2 pair, we have

$$P(y = w_2|x = w_1, \mathbf{v}^*) = \frac{Count(w_1, w_2)}{Count(w_1)}$$

where $Count(w_1, w_2) =$ number of times $(x_i, y_i) = (w_1, w_2)$, and $Count(w_1) =$ number of times $x_i = w_1$.

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)

We will construct a log-linear model that defines $P(y|x)$ for an "input" x paired with an "output" y. In this question the objects x and y are particularly simple: x can be either 0 or 1, and y is an ordered pair of symbols, $y = \langle y_1, y_2 \rangle$, where $y_1 \in \{0, 1\}$ and $y_2 \in \{0, 1\}$. Thus y can take one of four possible values: $\langle 0, 0 \rangle$, $\langle 0, 1 \rangle$, $\langle 1, 0 \rangle$, or $\langle 1, 1 \rangle$

We define two features in the model:

$$f_1(x, \langle y_1, y_2 \rangle) = \begin{cases} 1 & \text{if } y_1 = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(x, \langle y_1, y_2 \rangle) = \begin{cases} 1 & \text{if } y_2 = 1 \\ 0 & \text{otherwise} \end{cases}$$

The model then defines a probability distribution

$$P(\langle y_1, y_2\rangle|x) = \frac{e^{f_1(x,\langle y_1,y_2\rangle)\theta_1 + f_2(x,\langle y_1,y_2\rangle)\theta_2}}{\sum_{\langle y_1',y_2'\rangle \in \mathcal{Y}} e^{f_1(x,\langle y_1',y_2'\rangle)\theta_1 + f_2(x,\langle y_1',y_2'\rangle)\theta_2}}$$

where $\mathcal{Y}$ is the set $\{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,0\rangle, \langle 1,1\rangle\}$

Show that it is possible to write

$$P(\langle y_1, y_2\rangle|x) = P_1(y_1|x) \times P_2(y_2|x)$$

where P_1 and P_2 are themselves log-linear models, with a single feature each.

Question 5

Nathan L. Pedant generates (x, y) pairs as follows. Take $\mathcal{V}$ to be set of possible words (vocabulary), e.g., $\mathcal{V}$ = {`the, cat, dog, happy, ...`}. Take $\mathcal{V}'$ to be the set of all words in $\mathcal{V}$, **plus** the reversed string of each word, e.g., $\mathcal{V}'$ = {`the, eht, cat, tac, dog, god, happy, yppah, ...`}.

For each x, Nathan chooses a word from some vocabulary $\mathcal{V}$. He then does the following:

- With 0.4 probability, he chooses y to be identical to x.
- With 0.3 probability, he chooses y to be the reversed string of x.
- With 0.3 probability, he chooses y to be some string that is neither x nor the reverse of x. In this case he chooses y from the uniform distribution over words in $\mathcal{V}'$ that are neither x nor the reverse of x.

Question 5(a) (10 points)

Define a log-linear model that can model this distribution $P(y|x)$ perfectly (Note: you may assume that there are no palindromes in the vocabulary, i.e., no words like `eye` which stay the same when reversed.) Your model should make use of as few parameters as possible (we will give you 10 points for a correct model with 2 parameters, 8 points for a correct model with 3 parameters, 5 points for a correct model with more than 3 parameters.)

Question 5(b) (10 points)

Write an expression for each of the probabilities

$$P(\text{the}|\text{the})$$
$$P(\text{eht}|\text{the})$$
$$P(\text{dog}|\text{the})$$

as a function of the parameters in your model.

Question 5(c) (10 points)

What value do the parameters in your model take to give the distribution described above?

Question 6 (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).

Part 4: Extending the Approach

Thus far (in part 2) we've used a relatively simplistic approach for smoothing the $P(t_i|t_{i-2}, t_{i-1})$ parameters. In the final part of this project, you should implement a more sophisticated form of smoothing. You could use any one of the methods described in the lecture on language modeling. In your write-up you should describe the method you've implemented, and describe how well it performs compared to the simple form of smoothing developed in part 2. To test the accuracy of the two approaches, we've provided the file `wsj.19-21.withtags` which has the correct tag sequences for the test sentences.