6.864, Fall 2005: Problem Set 2

Total points: 160 regular points
Due date: 5pm, 18th October 2005
Late policy: 5 points off for every day late, 0 points if handed in after 5pm on October 22nd 2005

Question 1 (15 points)

In the absolute discounting model of smoothing, all non-zero MLE frequencies are discounted by a constant amount δ where $0 < \delta < 1$:
Absolute discounting: If $C(w_n|w_1 \dots w_{n-1}) = r$,

$$P_{abs}(w_n|w_1 \dots w_{n-1}) = \begin{cases} \frac{(r-\delta)}{N} & r > 0 \\ \frac{(V-N_0)\delta}{N_0 * N} & otherwise \end{cases}$$

(Here $C(w_n|w_1 \dots w_{n-1})$ is the number of times $w_1 \dots w_n$ has been seen, P_{abs} is the absolute discounting estimate, V is the size of the vocabulary, N is the total number of times $w_1 \dots w_{n-1}$ has been seen, and N_0 is the number of word types that were unseen after this context.)

Under linear discounting the estimated count of seen words is discounted by a certain fraction, defined by a constant α where $0 < \alpha < 1$.
Linear discounting: If $C(w_n|w_1, \dots, w_n) = r$,

$$P_{lin}(w_n|w_1 \dots w_{n-1}) = \begin{cases} \frac{(1-\alpha)r}{N} & r > 0 \\ \frac{\alpha}{N_0} & otherwise \end{cases}$$

1. Show that absolute discounting yields a probability distribution for any context $w_1 \dots w_{n-1}$.

1. Show that linear discounting yields a probability distribution for any context $w_1 \dots w_{n-1}$.

Question 2 (15 points)

Say we have a vocabulary $\mathcal{V}$, i.e., a set of possible words. We'd like to estimate a unigram distribution $P(w)$ over $w \in \mathcal{V}$. We observe n sample points, $w_1, w_2, \dots w_n$ (note that this sample may not include all members of $\mathcal{V}$, particularly if n is small compared to $|\mathcal{V}|$.) For any word seen r times in the training sample, the Good-Turing estimate of its count is

$$GT(r) = (r+1) * \frac{N_{r+1}}{N_r},$$

where N_r is the number of members of $\mathcal{V}$ which are seen r times in the corpus. For any w which is observed in the training corpus, we make the estimate $P(w) = GT(C(w))/n$, where $C(w)$ is the number of times w is seen in the sample.

1. Can you see any problem with this estimation method for words with large values for $C(w)$?

2. Prove that under this definition $\sum_{w \in \mathcal{V}'} P(w) \leq 1$, where $\mathcal{V}'$ is the subset of $\mathcal{V}$ seen in the training corpus. If the "missing" probability mass $1 - \sum_{w \in \mathcal{V}'} P(w)$ is divided evenly amongst the words not seen in the corpus, show that $P(w)$ for any word not in the corpus is $N_1/(n \times N_0)$ where N_0 is $|\mathcal{V}| - |\mathcal{V}'|$, and N_1 as before is the number of members of $\mathcal{V}$ seen exactly once in the corpus.

Question 3 (15 points)

Recall the "topic modeling" example in the lecture. In this model, the training sample $x^1, x^2, \ldots x^m$ is a sequence of m documents. We will take each document x^i to consist of n words, $x_1^i, x_2^i, \ldots x_n^i$. The hidden variables y can take one of K values, $1, 2, \ldots K$. The model is defined as follows:

$$P(x, y|\Theta) = P(y) \prod_{j=1}^{n} P(x_j|y)$$

Thus if $\mathcal{V}$ is the vocabulary—the set of possible words in any document—the parameters in the model are:

- $P(y)$ for $y = 1 \ldots K$
- $P(w|y)$ for $y = 1 \ldots K$ and $w \in \mathcal{V}$

Our aim in this question will be to derive EM updates which optimize the log-likelihood of the data:

$$L(\Theta) = \sum_{i=1}^{m} \log P(x^i|\Theta) = \sum_{i=1}^{m} \log \sum_{y} P(x^i, y|\Theta)$$

Give pseudo-code showing how to derive an updated parameter vector Θ^t from a previous parameter vector Θ^{t-1}. I.e., show pseudo-code that takes as input parameter estimates $P^{t-1}(y)$ for all y and $P^{t-1}(w|y)$ for all w, y, and as output provides updated parameter estimates $P^t(y)$ and $P^t(w|y)$ using EM. Use the notation $C(w, x)$ to denote the number of times word w is seen in document x.

Question 4 (15 points)

Recall the "word clustering" example in the lecture on EM on September 27th. In this model, the training sample $x^1, x^2, \ldots x^m$ is a sequence of m bigrams of the following form: each x^i is of the form w_1^i, w_2^i where w_1^i, w_2^i are words, and w_2^i is seen following w_1^i in the corpus. The hidden variables y can take one of K values, $1, 2, \ldots K$. The model is defined as follows:

$$P(w_2, y|w_1, \Theta) = P(y|w_1)P(w_2|y)$$

Thus if $\mathcal{V}$ is the vocabulary—the set of possible words in any document—the parameters in the model are:

- $P(y|w)$ for $y = 1 \ldots K$, for $w \in \mathcal{V}$
- $P(w|y)$ for $y = 1 \ldots K$ and $w \in \mathcal{V}$

Our aim in this question will be to derive EM updates which optimize the log-likelihood of the data:

$$L(\Theta) = \sum_{i=1}^{m} \log P(w_2^i|w_1^i, \Theta) = \sum_{i=1}^{m} \log \sum_{y} P(w_2^i|y)P(y|w_1^i)$$

Give pseudo-code showing how to derive an updated parameter vector Θ^t from a previous parameter vector Θ^{t-1}. I.e., show pseudo-code that takes as input parameter estimates $P^{t-1}(y|w)$ for all y, w and $P^{t-1}(w|y)$ for all w, y, and as output provides updated parameter estimates $P^t(y|w)$ and $P^t(w|y)$ using EM.

Question 5 (15 points)

In lecture (see also the accompanying note on EM) we saw how the forward-backward algorithm could be used to efficiently calculate probabilities of the following form for an HMM:

$$P(y_j = p|x, \Theta) = \sum_{y:y_j=p} P(y|x, \Theta)$$

and

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

where x is some sequence of output symbols, and Θ are the parameters of the model (i.e., parameters of the form π_i, $a_{j,k}$ and $b_j(o)$ as defined in the lecture). Here y_j is the j'th state in a state sequence y, and p, q are integers in the range $1 \ldots N-1$ assuming an N state HMM.

Question 5(a) (5 points) State how the following quantity can be calculated in terms of the forward-backward probabilities, and some of the parameters in the model:

$$P(y_2 = 1, y_3 = 2, y_4 = 1|x, \Theta)$$

(we assume that the sequence x is of length at least 4)

Question 5(b) (5 points) State how the following quantity can be calculated in terms of the forward-backward probabilities, and some of the parameters in the model:

$$P(y_2 = 1, y_5 = 1|x, \Theta)$$

(we assume that the sequence x is of length at least 5. Don't worry too much about the efficiency of your solution: we **do** expect you to use forward and backward terms, but we **don't** expect you to calculate any other quantities using dynamic programming.)

Question 5(c) (5 points) Say that we now wanted to calculate probabilities for an HMM such as the following:

$$\max_{y:y_j=p} P(y|x, \Theta)$$

so this is the maximum probability of any state sequence underlying x, with the constraint that the j'th label y_j is equal to p.

How would you modify the definition of the forward and backward terms—i.e., the recursive method for calculating them—to support this kind of calculation? How would you then calculate

$$\max_{y:y_3=1} P(y|x, \Theta)$$

assuming that the input sequence x is of length at least 3?

Question 6 (10 points)

We will now consider a slightly modified definition of an HMM, where each state emits a *pair* of symbols at each step, rather than a single symbol. In the new definition, a hidden Markov model $(N, \Sigma_1, \Sigma_2, \Theta)$ consists of the following elements:

- N is a positive integer specifying the number of states in the model. Without loss of generality, we will take the N'th state to be a special state, the *final* or *stop* state.
- Σ_1 is a first set of output symbols, for example $\Sigma_1 = \{a, b\}$
- Σ_2 is a second set of output symbols, for example $\Sigma_2 = \{c, d\}$
- Θ is a vector of parameters. It contains three types of parameters:
  - π_j for $j = 1 \ldots N$ is the probability of choosing state j as an initial state. Note that $\sum_{j=1}^{N} \pi_j = 1$.
  - $a_{j,k}$ for $j = 1 \ldots (N-1)$, $k = 1 \ldots N$, is the probability of transitioning from state j to state k. Note that for all j, $\sum_{k=1}^{N} a_{j,k} = 1$.
  - $b_j(o)$ for $j = 1 \ldots (N-1)$, and $o \in \Sigma_1$, is the probability of emitting symbol o as the first symbol from state j. Note that for all j, $\sum_{o \in \Sigma} b_j(o) = 1$.
  - $c_j(o)$ for $j = 1 \ldots (N-1)$, and $o \in \Sigma_2$, is the probability of emitting symbol o as the second symbol from state j. Note that for all j, $\sum_{o \in \Sigma} c_j(o) = 1$.

A regular HMM assigns a probability $P(x, y|\Theta)$ to any sequence of symbols x (such as `a a b b a`) paired with an underlying sequence of states y (such as `1 2 2 1 1`). The new form of HMM assigns a probability $P(x, y|\Theta)$ to any sequence of symbol *pairs* x paired with an underlying sequence of states y. For example, with

$$x = \langle \texttt{a}, \texttt{c} \rangle \langle \texttt{a}, \texttt{d} \rangle \langle \texttt{b}, \texttt{d} \rangle \langle \texttt{b}, \texttt{c} \rangle \langle \texttt{a}, \texttt{d} \rangle$$

where for example $\langle a, c \rangle$ is the ordered pair consisting of `a` and `c`, and $y =$ `12211`, we would have

$$P(x, y|\Theta) = \pi_1 a_{1,2} a_{2,2} a_{2,1} a_{1,1} a_{1,3} b_1(a) c_1(c) b_2(a) c_2(d) b_2(b) c_2(d) b_1(b) c_1(c) b_1(a) c_1(d)$$

Question (10 points): Describe how you would modify the definitions of the forward and backward terms so that they can be applied to this new form of HMM.

Question 7 (75 points)

Question 7(a) In the file `counts.gz`, you will find the counts for unigrams, bigrams, and trigrams, which were extracted from roughly 38,000 sentences from the Wall Street Journal. For example, the first few lines of the file are

```
66 !
2 ! !
1 ! ! #END#
```

This means that the unigram "!" was seen 66 times, the bigram "! !" was seen twice, and the trigram "! ! #END#" was seen once.

The first part of this question is to implement a language model. Your code should read the counts from `counts.gz` into some data structure, most likely a hash table. Your code should then provide a function of the form `trigram-prob(w1, w2, w3)` which returns an estimate $P(w_3|w_1, w_2)$ for any triple of words. You should use a **Katz back-off model (trigrams)** to define $P(w_3|w_1, w_2)$. See slides 26/27 from the 15th September lecture slides for a definition of this type of model. The Katz models require a definition of discounted counts. For these definitions, use

$$Count^*(w_1, w_2, w_3) = Count(w_1, w_2, w_3) - 0.5$$
$$Count^*(w_1, w_2) = Count(w_1, w_2) - 0.5$$

for the trigram and bigram cases, where $Count$ denotes a count taken from `counts.gz`, and $Count^*$ is the discounted count used in the Katz 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 trigram per line. For example, the file might contain

the dog sleeps
colorless green ideas

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

0.054
0.032

assuming that $P(sleeps|the, dog) = 0.054$**, and** $P(ideas|colorless, green) = 0.032$ **under your model.**

Some notes about the counts in `counts.gz`:

- The symbol `#END#` was added to the end of every sentence in the WSJ corpus. The symbols `#START1#` `#START2#` were added at the start of each sentence. For example, the sentence

  Shares fell by 225 points .

  would be processed to give

  #START1# #START2# Shares fell by 225 points . #END#

  The trigrams extracted from this sentence would then be

  #START1# #START2# Shares
  #START2# Shares fell
  Shares fell by
  fell by 225
  by 225 points
  225 points .
  points . #END#

- Sometimes you will see the token *#LOW#* in the file. In the original WSJ corpus, any word seen less than 10 times was replaced by the token *#LOW#*.

Question 7(b) The next step will be to implement a function that returns the probability of *an entire sentence* under the trigram language model. The function should read a sentence $w_1, w_2, \ldots w_n$ from a file (note that w_n will *not* be the symbol *#END#*, this end marker has not been added). It should calculate the log probability

$$\log P(w_1, w_2, \ldots w_n) = \log P(\text{\#END\#}|w_{n-1}, w_n) + \sum_{i=1}^{n} \log P(w_i|w_{i-1}, w_{i-2})$$

where terms such as $P(w_i|w_{i-1}, w_{i-2})$ are calculated using the trigram model you constructed in 7(a). Note that we always define $w_{-1} =$ #START1# and $w_0 =$ #START2#.

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. As output, the code should write the *log* probability for each sentence in turn under your model.

Question 7(c) We'll now use your code to construct a simple spell–checker, which attempts to spot cases where a writer has used *their* instead of *there*, or vice-versa. In the file `theirthere.test`, you will find 559 sentences. Each sentence contains a token `THERE-OR-THEIR`, where there was either the word *their* or *there*. For example, the first sentence

" #LOW# is written more for #LOW# , not THERE-OR-THEIR daughters , " said Mr. Lang .

was produced from the sentence

" #LOW# is written more for #LOW# , not their daughters , " said Mr. Lang .

(note that, as before, infrequent words have been replaced with the token *#LOW#*.)

For each sentence in `theirthere.test`, you should calculate two probabilities: one being the probability of the sentence under your trigram model where `THERE-OR-THEIR` is replaced by *their*, one where `THERE-OR-THEIR` is replaced by *there*. For example, for the first sentence you should calculate the probability of both

" #LOW# is written more for #LOW# , not their daughters , " said Mr. Lang .

and

" #LOW# is written more for #LOW# , not there daughters , " said Mr. Lang .

and then see which sentence has the higher probability.

Note: make sure your code has the following functionality. The code should be able to read the examples from `theirthere.test` one by one. As output, it should write (a) the *log* probability of the sentence with `THERE-OR-THEIR` replaced with *their*; (b) the *log* probability of the sentence with `THERE-OR-THEIR` replaced with *there*.