Lab 3: Language Modeling Fever

ELEN E6884/COMS 86884: Speech Recognition

Due: November 7, 2005 at 12:01am

0 Overview

The goal of this assignment is for you, the student, to implement basic algorithms for n-gram language modeling. This lab will involve counting n-grams and doing basic n-gram smoothing. For this lab, we will be working with *Switchboard* data. The Switchboard corpus is a collection of recordings of telephone conversations; participants were told to have a discussion on one of seventy topics (*e.g.*, pollution, gun control).

The lab consists of the following parts, all of which are required:

- **Part 1: Implement n-gram counting** — Given some text, collect the counts of all n-grams needed in building a trigram language model.
- **Part 2: Implement "$+\delta$" smoothing** — Write code to compute LM probabilities for a trigram model smoothed with "$+\delta$" smoothing.
- **Part 3: Implement Witten-Bell smoothing** — Write code to compute LM probabilities for a trigram model smoothed with Witten-Bell smoothing.
- **Part 4: Evaluate various n-gram models on the task of N-best list rescoring** — See how n-gram order and smoothing affects WER when doing N-best list rescoring for Switchboard.

All of the files needed for the lab can be found in the directory `~stanchen/e6884/lab3/`. Before starting the lab, please read the file `lab3.txt`; this includes all of the questions you will have to answer while doing the lab. Questions about the lab can be posted on Courseworks (`https://courseworks.columbia.edu/`); a discussion topic will be created for each lab. **Note:** The hyperlinks in this document are enclosed in square brackets; you need an online version of this document to find out where they point to.

1 Part 1: Implement n-gram counting

1.1 The Big Picture

For this lab, we will be compiling the code you write into the program EVALLMLAB3. Here is an outline of what this program does:

- training phase
  - reset all n-gram counts to 0
  - for each sentence in the training data
    - update n-gram counts (A)
- evaluation phase
  - for each sentence to be evaluated
    - for each n-gram in the sentence
      - call smoothing routine to evaluate probability of n-gram given training counts (B)
  - compute overall perplexity of evaluation data from n-gram probabilities

In the first part of the lab, you'll be writing the code that does step (A). In the next two parts of the lab, you'll be writing step (B) for two different smoothing algorithms.

1.2 This Part

In this part, you will be writing code to collect all of the n-gram counts needed in building a trigram model given some text. For example, consider trying to compute the probability of the word KING following the words OF THE. The maximum likelihood estimate of this trigram probability is:

$$P_{\text{MLE}}(\text{KING} \mid \text{OF THE}) = \frac{\text{count}(\text{OF THE KING})}{\sum_w \text{count}(\text{OF THE } w)} = \frac{\text{count}(\text{OF THE KING})}{\text{count}_{\text{hist}}(\text{OF THE})}$$

Thus, to compute this probability we need to collect the count of the trigram OF THE KING in the training data as well as the count of the bigram history OF THE. (The *history* is whatever words in the past we are conditioning on.) When building *smoothed* trigram LM's, we also need to compute bigram and unigram probabilities and thus also need to collect the relevant counts for these lower-order distributions.

Before we continue, let us clarify some terminology. Consider the maximum likelihood estimate for the bigram probability of the word THE following OF:

$$P_{\text{MLE}}(\text{THE} \mid \text{OF}) = \frac{\text{count}(\text{OF THE})}{\sum_w \text{count}(\text{OF } w)} = \frac{\text{count}(\text{OF THE})}{\text{count}_{\text{hist}}(\text{OF})}$$

Notice the term *count*(OF THE) in this equation and the term *count*$_{\text{hist}}$(OF THE) in the last equation. We refer to the former count as a regular bigram count and the latter count as a bigram *history* count. While these two counts will be the same for most pairs of words, they won't be the same for all pairs and so we distinguish between the two. Specifically, the history count is used for normalization, and so is defined as

$$\text{count}_{\text{hist}}(\text{OF THE}) \equiv \sum_{w} \text{count}(\text{OF THE } w)$$

A related point that is worth mentioning is that it is useful to have the concept of a *0-gram* history. Just as we use unigram history counts in computing bigram probabilities, we use 0-gram history counts in computing unigram probabilities. We use the notation $\text{count}_{\text{hist}}(\epsilon)$ to denote the 0-gram history count, and it is defined similarly as above, *i.e.*,

$$\text{count}_{\text{hist}}(\epsilon) \equiv \sum_{w} \text{count}(w)$$

In practice, instead of working directly with strings when collecting counts, all words are first converted to a unique integer index; *e.g.*, the words OF, THE, and KING might be encoded as the integers 1, 2, and 3, respectively. In this lab, the words in the training data have been converted to integers for you. To see the mapping from words to integers, check out the file `vocab.map` in the directory `~stanchen/e6884/lab3/`. As mentioned in lecture, in practice it is much easier to fix the set of words that the LM assigns (nonzero) probabilities to beforehand (rather than allowing any possible word spelling); this set of words is called the *vocabulary*. When encountering a word outside the vocabulary, one typically maps this word to a distinguished word, the *unknown token*, which we call <UNK> in this lab. The unknown token is treated like any other word in the vocabulary, and the probability assigned to predicting the unknown token (in some context) can be interpreted as the sum of the probabilities of predicting any word not in the vocabulary (in that context).

To prepare for the exercise, create the relevant subdirectory and copy over the needed files:

```
mkdir -p ~/e6884/lab3/
cd ~/e6884/lab3/
cp ~stanchen/e6884/lab3/Lab3_LM.C .
cp ~stanchen/e6884/lab3/.mk_chain .
```

Your job in this part is to fill in the sections between the markers `BEGIN_LAB` and `END_LAB` in the function `count_sentence()` in the file `Lab3_LM.C`. Read this file to see what input and output structures need to be accessed. This routine corresponds to step (A) in the pseudocode listed in Section 1.1. In this function, you will be passed a sentence (expressed as an array of integer word indices) and will need to update all relevant regular n-gram counts (trigram, bigram, and unigram) and all relevant history n-gram counts (bigram, unigram, and 0-gram). All of these counts will be initialized to zero for you.

In addition, for Witten-Bell smoothing (to be implemented in Part 3), you will also need to compute how many unique words follow each bigram/unigram/0-gram history. We refer to this as a "1+" count, since this is the number of words with one or more counts following a history.

It is a little tricky to figure out exactly which n-grams to count in a sentence, namely at the sentence begins and ends. For more details, refer to slide 39 (entitled "Technical Details: Sentence Begin and Ends") in the week 5 language modeling slides. The trigram counts to update correspond one-to-one to the trigram probabilities used in computing the trigram probability of a sentence. Bigram history counts can be defined in terms of trigram counts using the equation described earlier. How to do counting for lower-order models is defined analogously.

1.3 Compiling and testing

Your code will be compiled into the program EVALLMLAB3, which constructs an n-gram language model from training data and then uses this LM to evaluate the probability and perplexity of some test data. To compile this program with your code, type

```
smk EvalLMLab3
```

in the directory containing your source files (which must also contain the file `.mk_chain`).

To run this program (training on 100 Switchboard sentences and evaluating on 10 other sentences), run

```
lab3p1a.sh
```

(This script can be found in `~stanchen/pub/exec/`, which should be on your path from Lab 0.) This shell script starts the executable EVALLMLAB3 located in the current directory with the appropriate flags. To start up EVALLMLAB3 in the debugger, add the flag `-debug` to the above line. The "correct" output can be found in the file `p1a.out` in `~stanchen/e6884/lab3/`; your output should match the correct output exactly. For each trigram being evaluated in the evaluation set, the program is set up to output all of the relevant training counts for that trigram. (The count following the label `hist types` is the "1+" count.) The final cross-entropy/perplexity output will be bogus, since the code for computing LM probabilities won't be filled in until later in the lab. To see the training text used by this script, check out the file `minitrain.txt` in the directory `~stanchen/e6884/lab3/`.

The instructions in `lab3.txt` will ask you to run the script `lab3p1b.sh`, which does the same thing as `lab3p1a.sh` except on a different 10-sentence test set.

2 Part 2: Implement "$+\delta$" smoothing

In this part, you will write code to compute LM probabilities for a trigram model smoothed with "$+\delta$" smoothing. This is just like "add-one" smoothing in the readings, except instead of adding one count to each trigram, we will add δ counts to each trigram for some small δ (*e.g.*, $\delta = 0.0001$ in this lab). This is just about the simplest smoothing algorithm around, and this can actually work acceptably in some situations (though not in large-vocabulary ASR). To estimate the probability of a trigram $P_{+\delta}(w_i|w_{i-2}w_{i-1})$ with this smoothing, we take

$$P_{+\delta}(w_i|w_{i-2}w_{i-1}) = \frac{c(w_{i-2}w_{i-1}w_i) + \delta}{c_h(w_{i-2}w_{i-1}) + \delta \times |V|}$$

where $|V|$ is the size of the vocabulary. (Note: in the above equation and the rest of the document, we abbreviate count$(\cdot)$ as $c(\cdot)$ and count$_{\text{hist}}(\cdot)$ as $c_h(\cdot)$.)

Your job in this part is to fill in the function `get_prob_plus_delta()`. This function should return the value $P_{+\delta}(w_i|w_{i-2}w_{i-1})$ given a trigram $w_{i-2}w_{i-1}w_i$. You will be provided with the count of the trigram as well as the count of the bigram history (which you computed for Part 1), in addition to the vocabulary size. This routine corresponds to step (B) in the pseudocode listed in Section 1.1.

Your code will again be compiled into the program EVALLMLAB3. To compile this program with your code, type

```
smk EvalLMLab3
```

To run this program on the same Switchboard training and test set used in Part 1, run

```
lab3p2a.sh
```

The "correct" output can be found in the file `p2a.out` in `~stanchen/e6884/lab3/`. Again, you should be able to match this output just about exactly. In this script, the program is set up to print the smoothed probability you compute as well as the trigram and bigram history count for each trigram in the evaluation data.

The instructions in `lab3.txt` will ask you to run the script `lab3p2b.sh`, which does the same thing as `lab3p2a.sh` except on a different test set.

3 Part 3: Implement Witten-Bell smoothing

Witten-Bell smoothing is this smoothing algorithm that was invented by some dude named Moffat, but dudes named Witten and Bell have generally gotten credit for it. It is significant in the field of text compression and is relatively easy to implement, and that's good enough for us.

Here's a rough motivation for this smoothing algorithm: One of the central problems in smoothing is how to estimate the probability of n-grams with zero count. For example, let's say we're building a bigram model and the bigram $w_{i-1}w_i$ has zero count, so $P_{\text{MLE}}(w_i|w_{i-1}) = 0$. According to the Good-Turing estimate, the total mass of counts belonging to things with zero count in a distribution is the number of things with exactly one count. In other words, the probability mass assigned to the backoff distribution should be around $\frac{N_1(w_{i-1})}{c_h(w_{i-1})}$, where $N_1(w_{i-1})$ is the number of words w' following w_{i-1} exactly once in the training data (*i.e.*, the number of bigrams $w_{i-1}w'$ with exactly one count). This suggests the following smoothing algorithm

$$P_{\text{WB}}(w_i|w_{i-1}) \stackrel{?}{=} \lambda P_{\text{MLE}}(w_i|w_{i-1}) + \frac{N_1(w_{i-1})}{c_h(w_{i-1})} P_{\text{backoff}}(w_i)$$

where λ is set to some value so that this probability distribution sums to 1, and $P_{\text{backoff}}(w_i)$ is some unigram distribution that we can backoff to.

However, $N_1(w_{i-1})$ is kind of a finicky value; *e.g.*, it can be zero even for distributions with lots of counts. Thus, we replace it with $N_{1+}(w_{i-1})$, the number of words following w_{i-1} at least once (rather than exactly once), and we fiddle with some of the other terms. Long story short, we get

$$P_{\text{WB}}(w_i|w_{i-1}) = \frac{c_h(w_{i-1})}{c_h(w_{i-1}) + N_{1+}(w_{i-1})} P_{\text{MLE}}(w_i|w_{i-1}) + \frac{N_{1+}(w_{i-1})}{c_h(w_{i-1}) + N_{1+}(w_{i-1})} P_{\text{backoff}}(w_i)$$

For the backoff distribution, we can use an analogous equation:

$$P_{\text{backoff}}(w_i) = P_{\text{WB}}(w_i) = \frac{c_h(\epsilon)}{c_h(\epsilon) + N_{1+}(\epsilon)} P_{\text{MLE}}(w_i) + \frac{N_{1+}(\epsilon)}{c_h(\epsilon) + N_{1+}(\epsilon)} \frac{1}{|V|}$$

The term $c_h(\epsilon)$ is the 0-gram history count defined earlier, and $N_{1+}(\epsilon)$ is the number of different words with at least one count. For the backoff distribution for the unigram model, we use the uniform distribution $P_{\text{unif}}(w_i) = \frac{1}{|V|}$. Trigram models are defined analogously.

If a particular distribution has no history counts, then just use the backoff distribution directly. For example, if when computing $P_{\text{WB}}(w_i|w_{i-1})$ you find that the history count $c_h(w_{i-1})$ is zero, then just take $P_{\text{WB}}(w_i|w_{i-1}) = P_{\text{WB}}(w_i)$. Intuitively, if a history h has no counts, the MLE distribution $P_{\text{MLE}}(w|h)$ is not meaningful and should be ignored.

Your job in this part is to fill in the function `get_prob_witten_bell()`. This function should return the value $P_{\text{WB}}(w_i|w_{i-2}w_{i-1})$ given a trigram $w_{i-2}w_{i-1}w_i$. You will be provided with all of the relevant counts (which you computed for Part 1). Again, this routine corresponds to step (B) in the pseudocode listed in Section 1.1.

Your code will again be compiled into the program EVALLMLAB3. To compile this program with your code, type

```
smk EvalLMLab3
```

To run this program with the same training and test set as before, run

```
lab3p3a.sh
```

The "correct" output can be found in the file `p3a.out` in `~stanchen/e6884/lab3/`. Again, you should be able to match the values in this output just about exactly. This script is set up to print the smoothed probability you compute for each trigram in the test set. The file `p3a.out` also contains the results of some intermediate computations that may help you with debugging, but which you do not need to replicate.

The instructions in `lab3.txt` will ask you to run the script `lab3p3b.sh`, which does the same thing as `lab3p3a.sh` except on a different test set.

4 Part 4: Evaluate various n-gram models on the task of N-best list rescoring

In this section, we use the code you wrote in the earlier parts of this lab to build various language models on the full original Switchboard training set (about 3 million words). We will investigate how n-gram order (*i.e.*, the value of n) and smoothing affect WER's using the paradigm of *N-best list rescoring*.

In ASR, it is sometimes convenient to do recognition in a two-pass process. In the first pass, we may use a relatively small LM (to simplify the decoding process) and for each utterance output the N best-scoring hypotheses, where N is typically around 100 or larger. Then, we can use a more complex LM to replace the LM scores for these hypotheses (retaining the acoustic scores) to compute a new best-scoring hypothesis for each utterance. To see an example N-best list, see the file `~stanchen/e6884/lab3/nbest.txt`. The correct transcript for this utterance is DARN; each line contains a hypothesis word sequence and an acoustic logprob at the end (*i.e.*, $\log P(\mathbf{x}|\omega)$).

To give a little more detail, recall the fundamental equation of speech recognition

$$\text{class}(\mathbf{x}) \approx \arg\max_{\omega} \; P(\omega)^{\alpha} P(\mathbf{x}|\omega) = \arg\max_{\omega} \; [\alpha \log P(\omega) + \log P(\mathbf{x}|\omega)]$$

where $\mathbf{x}$ is the acoustic feature vector, ω is a word sequence, and α is the language model weight. In N-best list rescoring, for each hypotheses ω in an N-best list, we compute $\log P(\omega)$ for our new language model and combine it with the acoustic model score $\log P(\mathbf{x}|\omega)$ computed earlier. Then, we compute the above argmax over the hypotheses in the N-best list to produce a new best-scoring hypothesis.

For this part of the lab, we have created 100-best lists for each of 100 utterances of a Switchboard test set, and we will calculate the WER over these utterances when rescoring using various language models. Because the LM used in creating the 100-best lists prevents really bad hypotheses (from an LM perspective) from making it onto the lists, WER differences between good and bad LM's will be muted when doing N-best list rescoring as compared to when using the LM's directly

in one-pass decoding. However, N-best list rescoring is very easy and cheap to do so we use it here.

First, let us see how WER compares between unigram, bigram, and trigram models. Run the following scripts:

```
lab3p4.1.sh
lab3p4.2.sh
lab3p4.3.sh
```

These scripts call EVALLMLAB3 to build unigram, bigram, and trigram models, respectively, on the full Switchboard corpus and then do N-best list rescoring on the test set. In particular, these scripts create a data set consisting of all of the hypotheses in the N-best lists for each utterance. Then, EVALLMLAB3 is run using this data set as its evaluation set, which produces the total LM probability for each hypothesis. We combine these LM scores with the acoustic model scores already in the N-best lists to compute a new highest-scoring hypothesis for each utterance, and then compute the WER of these hypotheses. (To get your code to return bigram or unigram probabilities, we just zero out the counts for all trigrams or bigrams+trigrams, respectively.)

Now, let us see how smoothing affects WER. Run the following scripts:

```
lab3p4.mle.sh
lab3p4.delta.sh
lab3p4.wb.sh
```

These do N-best list rescoring with trigram models with different smoothing. For the MLE trigram model, we assign a small nonzero floor probability to trigram probabilities that have an MLE of zero. (This will make some conditional distributions sum to slightly more than 1, but we don't care in N-best list rescoring.)

Finally, let us see how training data size affects WER (with Witten-Bell trigram models). Instead of using the full Switchboard corpus as the LM training set, we use subsets of different sizes. Run the following scripts:

```
lab3p4.2000.sh
lab3p4.20000.sh
lab3p4.200000.sh
```

The number in the script name is the number of sentences in the training set; there are about 13 words per sentence on average.

5 What is to be handed in

Make a copy of the ASCII file `lab3.txt` from the directory `~stanchen/e6884/lab3/`. Fill in all of the fields in this file and E-mail the contents of the file to `stanchen@watson.ibm.com`. (Please paste this file into the main body of the E-mail; *i.e.*, don't include this file as an attachment.)

Incidentally, if you find that your forehead is becoming warm as you do this assignment, do not be alarmed: you probably have *language modeling fever*. It should recede by itself within a day, but if it does not, go see a doctor and tell them that you have language modeling fever; they'll know what to do.