Working with n-grams in SRILM

Linguistics 165, Professor Roger Levy

13 February 2015

1. **Recap:** an n-gram model estimates the probability of a length-N sentence $\mathbf{w}$ as

$$P(\mathbf{w}) \approx \prod_{i=1}^{N+1} P(w_i | w_{i-n+1} \dots w_{i-1}) \tag{1}$$

   where w_{N+1} and w_j for $j < 1$ are defined as special "sentence-boundary" tokens.

2. I gave you a "starter" bigram model in Python last week. This is useful for small-scale exploration of how to build a model and do things like generate from it, but won't scale up well

3. Fortunately there is state-of-the-art open-source software for n-gram modeling: the SRI Language Modeling Toolkit (SRILM; `http://www.speech.sri.com/projects/srilm/`)!

4. **Getting started:** You can download a text representation of the "toy" corpus I used in the lecture on Monday from `http://idiom.ucsd.edu/~rlevy/teaching/2015winter/lign165/lectures/lecture13/toy-example/corpus.txt`. You can save it in a text file on your lab workstation and then use `FileZilla` to transfer it to your instructional server account, or you can log into the instructional server and then use the `wget` command to fetch it directly from the command line: the command

```
wget <url>
```

   fetches the contents of the webpage at address `<url>` directly.

   The workhorse program for estimating n-gram models in SRILM is `ngram-count`. The simplest thing that this program does is simply to count n-grams in a text file. We can try this for counting bigrams in our toy corpus:

```
ngram-count -text corpus.txt -order 2 -write corpus.count
```

   This writes a set of bigram counts in the file `corpus.count`. You can use the `cat` command (which is used to concatenate files and print to the screen) to see its contents:

```
cat corpus.count
```

I'm not showing the complete output here, but you can see that the counts look just like the ones I showed you in the previous lecture. (Note that SRILM uses the `<s>` symbol as "beginning of sentence".)

Adding the `-lm` option, together with a filename, to the invocation of `ngram-count` causes a language model to be estimated and its parameters (probabilities) to be written to the filename specified. By default, SRILM computes a *smoothed* model, so we have to specify that we want an unsmoothed model. One way of doing this is to include `-addsmooth 0` as an option. (You'll see a bit later on why this works.)

```
ngram-count -text corpus.txt -order 2 -addsmooth 0 -lm corpus.lm
```

If you look at the contents of the resulting file `corpus.lm`, by running (for example)

```
cat corpus.lm
```

you will see (1) a header that tells you how many unique n-gram types were observed of each order n up to the maximum order of the model:

```
\data\
ngram 1=10
ngram 2=18
```

and later you will see a list of each of those n-grams, with each n-gram preceded by the log (base-10) of the conditional probability of the word given the preceding $n-1$ words (some n-grams are also followed by numbers; you can ignore these following numbers at this point):

```
\2-grams:
-0.544068       <s> cats
-0.243038       <s> dogs
-0.845098       <s> the
0       bark </s>
...
```

This says that the probability of the first word in a sentence being *dogs* is $10^{-0.243038} = 0.571$, the probability given the word *bark* that the next thing that happens is that the sentence ends is $10^0 = 1.0$, and so forth. If you check the handout from the previous lecture you'll see that these match the relative frequency estimates we calculated by hand. More complete information about how model parameters are recorded in language-model files can be found in the manual page for `ngram-format`—you can find it online at `http://www.speech.sri.com/projects/srilm/manpages/ngram-format.5.html`.

5. **Calculating model perplexity with SRILM.** Once you have a language model written to a file, you can calculate its perplexity on a new dataset using SRILM's `ngram` command, using the `-lm` option to specify the language model file and the

`-ppl` option to specify the test-set file. I've provided a text-file version of the test-set file at `http://idiom.ucsd.edu/~rlevy/teaching/2015winter/lign165/lectures/lecture13/toy-example/test_corpus.txt` which you can fetch via `wget`. If you run the command

```
ngram -lm corpus.lm -ppl test_corpus.txt
```

you get the output

```
file test_corpus.txt: 2 sentences, 6 words, 0 OOVs
0 zeroprobs, logprob= -2.59329 ppl= 2.10941 ppl1= 2.70529
```

and if you check the notes from the previous lecture, you can see that the same perplexity measure we hand-calculated, of 2.109, is obtained.

As an added bonus, by using the `-debug` option, you can even get word-by-word n-gram probabilities. Try this with

```
ngram -lm corpus.lm -ppl test_corpus.txt -debug 2
```

for example!

6. **Why we need smoothing.** It's very easy to "break" our language model. For example, if we just add the sentence `birds chirp` to our test set, we will get a zero-probability event and this means that our entire dataset gets probability zero, and thus has infinite perplexity! You can use the `cp` (copy) command to create a copy of the test set, and then the `cat` command to add this sentence to it:

```
cp test_corpus.txt test_corpus2.txt
cat >> test_corpus2.txt
birds chirp
```

(The `cat` command above means "append whatever I type next to the file `test_corpus2.txt`", and it will accept multi-line input. Press Control-D after adding the line `birds chirp` to finish the input that you're appending to the file.) If you now run

```
ngram -lm corpus.lm -ppl test_corpus2.txt -debug 2
```

you will see a zero-probability event. In its overall summary at the end of the output, SRILM actually sets aside the zero-probability event (notice it tells you `1 zeroprobs`) and gives you a perplexity, but it's just being polite: really you have infinite perplexity because of just that one zero-probability event!

The problem arises from the fact that in our training set we never saw the word `birds` at the beginning of the sentence, and that we used relative-frequency estimation to estimate our bigram probabilities:

$$\widehat{P}_{RFE}(w_i|w_{i-1}) = \frac{\text{Count}(w_{i-1}w_i)}{\text{Count}(w_{i-1})}$$

where $\widehat{P}_{RFE}$ means the relative frequency estimate. Note that relative frequency estimation always assigns zero probability to any event we've never seen in our training data. This is rather fool-hardy when it comes to human language! What we want to do instead is to SMOOTH our language model so that it puts non-zero probability on events that it's never seen in the training data. These probabilities should still generally be smaller than the probabilities of what we have seen, but they shouldn't be zero.

7. **Laplace smoothing.** The simplest smoothing technique is LAPLACE, or ADD-ONE smoothing. This technique says that, if we have V words in our vocabulary, we'll estimate our n-gram probabilities as

$$\widehat{P}_{\text{Laplace}}(w_i|w_{i-n+1}\dots w_{i-1}) = \frac{\text{Count}(w_{i-n+1}\dots w_{i-1}w_i)+1}{\text{Count}(w_{i-n+1}\dots w_{i-1})+V}$$

We can think of this as implicitly doing relative frequency estimation on an "augmented" version of our training set in which we've not only got our actual training data, but we've effectively "observed" every possible n-gram in the language exactly once. Sometimes we say that every n-gram gets an additional PSEUDO-COUNT of 1.

In SRILM we can use the `-addsmooth 1` option to `ngram-count` for Laplace smoothing:

```
ngram-count -text corpus.txt -order 2 -addsmooth 1 \
  -write corpus_laplace.count -lm corpus_laplace.lm
```

(Note that the \ at the end of line 1 is a convenience that allows you to split command-line input over multiple lines, for readability. You could also just get rid of it and put both lines of input into one line of shell input.) This command simultaneously writes the n-gram counts into the file `corpus_laplace.count`, due to the `-write` option, and writes the language model probabilities into the `corpus_laplace.lm` file, due to the `-lm` option. If you use `cat` to inspect the original and new counts files (`corpus.count` and `corpus_laplace.count`), you'll find that they're identical. However, the `corpus_laplace.lm` file looks different from the original, relative-frequency estimated `corpus.lm` file. This is because the new file has probabilities that reflect Laplace smoothing. Here is one example: the entry for *dogs* beginning a sentence in the new file can be found at the top of the bigrams part of `corpus_laplace.lm`:

```
\2-grams:
-0.7269987 <s> cats
-0.50515 <s> dogs
```

The conditional probability $P(\text{dogs}|\texttt{<s>})$ is $10^{-0.50515} = 0.3125$, which is lower than the original relative frequency estimate of $\frac{4}{7} = 0.571$. Unless told otherwise, SRILM assumes that the only "in-vocabulary" words are those that appear in the training set; there nine unique word types in our training set, so SRILM assumes that $V = 9$. So the Laplace estimate should be $\frac{4+1}{7+9} = \frac{5}{16}$, and indeed this is 0.3125. If we check the perplexity of the original testset with this new model:

```
ngram -lm corpus_laplace.lm -ppl test_corpus.txt
```

we see that it has higher perplexity than the older model. This is because the Laplace-smoothed model has held out probability mass for all possible bigram events (including ridiculous ones like `the</s>`!). There are 81 of them (why?), and so the total pseudo-counts are much larger than the size of our training set!

On the other hand, if we check the perplexity of the second test set with the new model:

```
ngram -lm corpus_laplace.lm -ppl test_corpus2.txt
```

we see that there are no more zero-probability events! This is good!

8. **Scaling up to larger datasets.** Unlike `nltk`, SRILM is carefully optimized to handle large datasets, which inevitably become important with n-gram models because there are *so many* possible events in language. In `/home/linux/ieng6/ln165w/public/data` I have put the following files:

`wsj_1994_tokenized.txt`	All the text from the 1994 Wall Street Journal, tokenized
`wsj_1994_train.txt`	A training set of the first 120,000 sentences from this dataset
`wsj_1994_dev.txt`	A development set of the next 10,000 sentences
`wsj_1994_test.txt`	A test set of the final 9,443 sentences
`wsj_1994_lexicon.txt`	The set of all words occurring in this dataset

The shell command `wc` counts lines, words, and characters of files. Executing

```
wc wsj_1994_lexicon.txt
```

shows us that there are 88,982 unique word types in this dataset! (How many word *types* are there in the dataset?)

If we call

```
ngram-count -text /home/linux/ieng6/ln165w/public/data/wsj_1994_train.txt \
  -vocab /home/linux/ieng6/ln165w/public/data/wsj_1994_lexicon.txt \
  -order 2 -addsmooth 1 -lm wsj2.lm
```

then we train a bigram model with add-one smoothing. **Task:** calculate the perplexity of this model on the development set.

9. **Additive smoothing more generally.** Although add-one smoothing is easy, it is not a great way of doing language modeling. There are $8,892^2 = 79,067,664$ possible bigrams, which is an order of magnitude higher than the size of our training data! We can generalize additive smoothing to make the pseudo-counts play a smaller role by adding some constant α instead of 1 to all counts:

$$\widehat{P}_\alpha(w_i|w_{i-n+1}\dots w_{i-1}) = \frac{\text{Count}(w_{i-n+1}\dots w_{i-1}w_i)+\alpha}{\text{Count}(w_{i-n+1}\dots w_{i-1})+\alpha V}$$

In SRILM, we do this by changing the ARGUMENT of the `-addsmooth` option to some value other than 1. You've tried $\alpha = 1$ already; now try $\alpha = 0.1$ and $\alpha = 0.01$. Which gives the lowest development-set perplexity?

10. **Linear interpolation.** One limitation of the additively-smoothed bigram model is that we back off to a state of severe ignorance when we're dealing with rare contexts. For example, the word *devoured* occurs only four times in all the training data: twice followed by *by*, once by *the*, and once by *Galen.* With additive smoothing, this means that $P(\text{a}|\text{devoured})$ is the same as, for example, $P(\text{pistachio}|\text{devoured})$—both of which are plausible but the former of which surely should be more probable. One source of information we could use to fix this problem is lower-order, UNIGRAM probabilities—*a* is simple a more common word than *devour.* We can define an INTERPOLATED model that combines unigram and bigram probabilities to bring in this source of information:

$$P_{\text{Interpolated}}(w_i|w_{i-1}) = \lambda P(w_i|w_{i-1}) + (1-\lambda)P(w_i)$$

where $0 \leq \lambda \leq 1$ is an interpolation WEIGHT. The larger λ, the more the model will behave like a bigram model; the smaller λ, the more the model will behave like a unigram model.

In SRILM we do interpolation first by training a unigram model to go with our bigram model; we do this with `-order 1` whereas we'd used `-order 2` for bigram models:

```
ngram-count -text /home/linux/ieng6/ln165w/public/data/wsj_1994_train.txt \
  -vocab /home/linux/ieng6/ln165w/public/data/wsj_1994_lexicon.txt \
  -order 1 -addsmooth 0.0001 -lm wsj1.lm
```

Note that we don't need a very large additive smoothing parameter for a unigram model, because the data are effectively much less dense!

And now we use the `-mix-lm` and `-lambda` options for `ngram` to interpolate in calculating perplexity:

```
ngram -lm wsj2.lm -mix-lm wsj1.lm -lambda 0.5 \
  -ppl /home/linux/ieng6/ln165w/public/data/wsj_1994_dev.txt
```

TASK: play around with different choices of the λ and bigram-α parameters and see what does best on the development set. Then, choose one final set of values, and compute perplexity on the test set! We'll see how everyone does, and what the best set of values across the entire class was!

11. **Shell commands review.** In addition to the SRILM commands we've used in this session, I've used a number of general shell commands and other programs that are worth reviewing. These include `cat`, `cp`, `wc`, and `wget`. Also important is paging backwards and forwards through your command history with the up and down arrows in the shell. Review all these things at home or in the lab outside of class.