4. 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 $P_{MLE}(w_i \mid w_{i-1}) = 0$. 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 $N_1(w_{i-1}) / c_h(w_{i-1})$, 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 `lambda` is set to some value so that this probability distribution sums to 1, and $P_{backoff}(w_i)$ 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 $P_{unif}(w_i) = 1 / |V|$. 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 $P_{WB}(w_i \mid w_{i-1})$ you find that the history count $c_h(w_{i-1})$ is zero, then just take $P_{WB}(w_i \mid w_{i-1}) = P_{WB}(w_i)$. Intuitively, if a history `h` has no counts, the MLE distribution $P_{MLE}(w \mid h)$ 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 $P_{WB}(w_i \mid w_{i-2}\ w_{i-1})$ 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 2.1.

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.

Prev
Part 2: Implement "`+delta`" smoothing

Home

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