One-count smoothing (minor revision from the version by Professor Jason Eisner@JHU)

For full credit, use the following nice type of smoothing. It is basically just add-λ smoothing with backoff, but λ is set higher in contexts with a lot of "singletons" – words that have only occurred once – because such contexts are likely to have novel words in test data. This is called "one-count" smoothing.

First let us define our backoff estimates:

- Let

$$p_{\text{tt-backoff}}(t_i \mid t_{i-1}) = p_{\text{t-unsmoothed}}(t_i) = \frac{c(t_i)}{n}$$

- Let

$$p_{\text{tw-backoff}}(w_i \mid t_i) = p_{\text{w-addone}}(w_i) = \frac{c(w_i)+1}{n+V}$$

This backoff estimate uses add-one-smoothing. n and V denotes the number of words tokens and types, respectively, that were observed in training data. In addition, V includes an *OOV* (Out-Of-Vocabulary) type (same as the <UNK> symbol we used before.)

Notice that according to this formula, any novel word has count 0 and backoff probability $p_{\text{w-addone}} = \frac{1}{n+V}$.

In effect, we are treating all novel words as if they had been replaced in the input by a single special word *OOV*. That way we can pretend that the vocabulary is limited to exactly V types, one of which is the unobserved *OOV*. Your definition of V should include all types that were observed in **train U raw (Union of these two sets)**. This vocabulary set should be used for all your computations.

Now for the smoothed estimates:

- Define a function *sing* that counts singletons. Let

$$\begin{aligned} sing_{\text{tt}}(\cdot \mid t_{i-1}) &= \text{number of tag types } t \text{ such that } c(t_{i-1}, t) = 1 \\ sing_{\text{tw}}(\cdot \mid t_i) &= \text{number of word types } w \text{ such that } c(t_i, w) = 1 \end{aligned}$$

There is an easy way to accumulate these singleton counts during training. Whenever you increment c(t, w) or c(t, t), check whether it is now 1 or 2. If it is now 1, you have just found a new singleton and you should increment the appropriate singleton count. If it is now 2, you have just lost a singleton and you should decrement the appropriate singleton count.

- Notice that $sing_{tw}(\cdot|N)$ will be high because many nouns only appeared once. This suggests that the class of nouns is open to accepting new members and it is reasonable to tag new words with N too. By contrast, $sing_{tw}(\cdot|D)$ will be 0 or very small because the class of determiners is pretty much closed – suggesting that novel words should not be tagged with D. we will now take advantage of these suggestions.
- Let

$$p_{\text{tt}}(t_i \mid t_{i-1}) = \frac{c(t_{i-1}, t_i) + \lambda \cdot p_{\text{tt-backoff}}(t_i \mid t_{i-1})}{c(t_{i-1}) + \lambda} \text{ where } \lambda = 1 + sing_{\text{tt}}(\cdot \mid t_{i-1})$$

$$p_{\text{tw}}(w_i \mid t_i) = \frac{c(t_i, w_i) + \lambda \cdot p_{\text{tw-backoff}}(w_i \mid t_i)}{c(t_i) + \lambda} \text{ where } \lambda = 1 + sing_{\text{tw}}(\cdot \mid t_i)$$

Note that λ will be higher for $p_{tw}(\cdot|N)$ than for $p_{tw}(\cdot|D)$. Hence $p_{tw}(\cdot|N)$ allows more backoff, other things equal, and so assign a higher probability to novel words.

If one doesn't pay respect to the difference between open and closed classes (http://www.ucl.ac.uk/internet-grammar/wordclas/open.htm), then novel words will often get tagged as D (for example) in order to make neighboring words happy. Such a tagger does worse than the baseline tagger (which simply tags all novel words with the most common singleton tag, N)!

You can test your code on the *ic* dataset which happen to have no singletons at all (neither novel words), you'll always have λ = 1 (equivalent to add-one smoothing with backoff).

```
Tagging accuracy (Viterbi decoding): 90.91%   (known: 90.91%  novel: 0.00%)
```

To allow a more detailed test of whether you counted singletons correctly in your one-count smoothing, the ic data set is modified to contain singletons. Here is the test result on the new ic data set (ic2train, ic2test):

```
Tagging accuracy (Viterbi decoding): 90.91%   (known: 90.32%  novel: 100.00%)
```