Hints for A1

Instructor: Yejin Choi *CSE 517/447 Win 24*

1 N-gram Models

1.1 What do N-gram models do?

The N-gram models should give probabilities of the next token over the entire vocabulary given a sequence to condition on. Your N-gram models should be case-sensitive (e.g., treating upper and lower cases differently).

1.2 UNK Tokens

Remember, you build the vocabulary with only the training data. Thus, for words in the dev/test set, if they're not in your vocab (that you built from training data), you should convert them into UNK. Once you make the conversion of all UNK tokens, you treat UNK as a normal token.

1.3 START Tokens

START tokens are not part of your vocabulary, but your N-gram models should be able to handle N-grams containing START.

Why "START tokens are not part of your vocabulary"? Your vocab contains all the tokens you may choose from when predicting the next token. The probability of all tokens in your vocab should sum to 1. Since you'll never expect the model to predict the START token as the next token (as it only appears at the beginning of the sentence), you should not take the START token into account when normalizing the probability.

How to handle START token in the unigram model? You don't need to do anything special to handle the START token in the unigram model, as the next token's probability depends only on itself and has nothing to do with the START token.

How to handle START token in the bigram model? In addition to give the probability of any regular bigrams, e.g., ("Paul", "Allen"), the bigram model should also give probabilities of bigrams that start with the START token, e.g., (START, "Paul").

How to handle START token in the trigram model? In addition to give the probability of any regular trigram, e.g., ("Paul", "Allen", "School"), the bigram model should also give probabilities of bigrams that start with the START tokens, e.g., (START, "Paul", "Allen"), (START, START, "Paul").

1.4 Perplexity

In this assignment, you will treat each line in the data file as a separate document. You won't count the n-grams that span multiple lines and will have a STOP token at the end of each line.

When computing perplexity over the entire file, you can compute perplexity for each line separately and then take the average across all lines. Alternatively, you can take the weighted average over all lines (discounted by the number of words in each line). **Both implementations are acceptable for this assignment as long as you explain them clearly in your write-up.**

However, in practice, to estimate the perplexity of the overall document, taking the weighted average will better estimate the overall document's perplexity with sentence-level perplexities. This way, the final document-level perplexity will not be biased too much by shorter sentences.

1.4.1 Without Smoothing

Given a sequence, S, perplexity is the inverse probability of the test sequence, normalized by the number of words in the sequence.

$$PPL(S) = P(S)^{-\frac{1}{|S|}} \tag{1}$$

However, you might encounter numerical underflow as you are multiplying a lot of probability together when computing $P(S)$. This can be improved with log transformation. How do we formulate $PPL(S)$ with log probability? Let's apply log to $P(S)$ (remember $log(a)+log(b) = log(a*b)$, and you are using the former to avoid multiplying probabilities together) then revert it back to $P(S)$ by exponentiate it. (Here we assume natural logarithm.)

$$P(S) = e^{\log P(S)} \tag{2}$$

Plug the expanded expression of $P(S)$ into Eq. 1:

$$PPL(S) = (e^{\log P(S)})^{-\frac{1}{|S|}} \tag{3}$$

$$PPL(S) = e^{-\frac{1}{|S|}\log P(S)} \tag{4}$$

Note that $-\frac{1}{|S|}\log P(S)$ is exactly cross-entropy. Therefore, another formulation of perplexity is the exponentiated cross entropy, $PPL(S) = e^{H(S)}$.

To expand $\log P(S)$:

$$\log P(S) = \sum_{i=1}^{|S|} \log P(s_i|s_1, ..., s_{i-1}) \tag{5}$$

The final expression of perplexity with log probability,

$$PPL(S) = e^{-\frac{1}{|S|}\sum_{i=1}^{|S|}\log P(s_i|s_1,...,s_{i-1})} \tag{6}$$

Note that $\log P(s_i|s_1, ..., s_{i-1})$ is the generic expression of log probability of s_i given $s_1, ..., s_{i-1}$ in language modeling. For unigram, bigram, and rigram models, you will have different ways to compute $\log P(s_i|s_1, ..., s_{i-1})$.

1.4.2 Laplace-smoothing (Add-one smoothing)

Note that many N-grams are rare under your N-gram model (i.e., have 0 count). As a result, the distribution for the next token given a sequence can be spiky, with few frequent tokens taking the most probability mass. Thus, we want to employ regularization methods to improve the data sparsity issue and make the model generalize better. Laplace-smoothing (also called add-one smoothing) is the easiest regularization method. How do we calculate perplexity under Laplace-smoothing for our unigram, bigram, and trigram models? The key is to pretend that each token in your vocab has been seen **one more time** than it actually is.

Unigram:

$$P(s_i) = \frac{Count(s_i) + 1}{\sum_{s_j \in V} (Count(s_j) + 1)} = \frac{Count(s_i) + 1}{\sum_{s_j \in V} Count(s_j) + |V|} \tag{7}$$

Bigram:

$$P(s_i | s_{i-1}) = \frac{Count(s_{i-1} s_i) + 1}{\sum_{s_j \in V} (Count(s_{i-1} s_j) + 1)} = \frac{Count(s_{i-1} s_i) + 1}{\sum_{s_j \in V} Count(s_{i-1} s_j) + |V|} \tag{8}$$

Trigram:

$$P(s_i | s_{i-2} s_{i-1}) = \frac{Count(s_{i-2} s_{i-1} s_i) + 1}{\sum_{s_j \in V} (Count(s_{i-2} s_{i-1} s_j) + 1)} = \frac{Count(s_{i-2} s_{i-1} s_i) + 1}{\sum_{s_j \in V} Count(s_{i-2} s_{i-1} s_j) + |V|} \tag{9}$$

Acknowledgement

This document is written by Liwei Jiang and Muru Zhang.