CSPE73- Natural Language Processing

Assignment 0 – Language Models

Date: 27/08/2022

In this assignment, your primary goal is to implement unigram and bigram language models and evaluate their performance. You'll use the equations from chapter 4; in particular you will implement maximum likelihood estimation with add-k smoothing, as well as a perplexity calculation to test your models.
The skeleton code for this assignment is available in this assignment folder.
There is a small interface given so you can test your program by running:

```
python language_modeling.py
```

And it will run your model on some data accompanying the assignment (specifically, Sam I Am) and report its performance.

Your Jobs

There are a number of ways to build a language model; I've set up a basic interface which I'd like you to stick with for the basic submission, but you can change things around wildly in any extensions.

For this assignment the main things you need to do are to write
the `train`, `predict_unigram`, `predict_bigram`, and `test_perplexity` functions.

`train`

The way it's set up, the `train` function need only accumulate counts, the resulting probabilities can be calculated at test time in the `predict_*` functions. This is slower at test time though, so one simple extension is to modify this (see below).

You can expect the training corpus to contain one sentence per line, already tokenized, so you can split it up on whitespace (e.g., `sentence.split()`). Important reminder that you must add `<s>` tokens to the beginning and `</s>` tokens to the end of every sentence.

`predict_unigram`, `predict_bigram`

These functions should take a sentence (as a string), split it into tokens on whitespace, add start and end tokens, and then calculate the probability of that sentence sequence using a unigram or bigram language model, respectively.

Notes:

- **Use add-k smoothing** in this calculation. This is very similar to maximum likelihood estimation, but adding `k` to the numerator and `k * vocab_size` to the denominator.
- **Return log probabilities!** As talked about in class, we want to do these calculations in log-space because of floating point underflow problems. Do this per word! Even a long sentence could

easily get us to an underflow. So create a float that starts at 0.0, and for each word where you get the probability, do `+= math.log(prob)` onto that float variable. Return this sum in the end.

test_perplexity

This function takes the path to a new corpus as input and calculates its perplexity (normalized total log-likelihood) relative to a new test corpus.

The basic gist here is quite simple - use your `predict_*` functions to calculate sentence-level log probabilities and sum them up, then convert to perplexity by doing the following:

```
math.exp(-1 * total_log_prob / N)
```

Where `N` is the total number of words seen. There are some additional details given in the function docstring in the skeleton code.