KN-WB-Smoothing

Procedure to run the model:

1. If you want to split the corpus into train and test data and generate perplexity score for the test sentences, **comment out the code** below the commented category "SINGLE SENTENCE".
2. If you want to run the code over a single sentence and generate perplexity score for only the input sentence, **comment out the code** under the commented category "TRAIN-TEST CODE".

The input regardless of the above on the command line prompt should be-

```
python3 language_model.py <value of n> <smoothing type> <filename>
```

Here,

1. "<value of n>" : Accepts values from 1-4. For any value above 4, it sets n=4.
2. "<smoothing type>" : Accepts only 'w' for Witten-Bell and 'k' for Kneyser-Ney.
3. "<filename>" : Insert the corpus to train or test.

Implementation of Witten-Bell and Kneyser-Ney:

Smoothing is used to adjust for/manage previously unseen grammes so that the overall probability does not drop to zero just because a new token appears.

Kneyser-Ney does this by employing a technique known as interpolation by absolute discounting. Essentially, only when the count in the higher order model increases does the lower order model become more necessary/significant, and weights are applied accordingly. For counts greater than one, a predetermined discount value is deducted, and weights for various order models are assigned according to the count.

Witten Bell uses a technique known as a backoff to address this problem. We fall back to a lower order model if a higher model has no count. It tries to encapsulate the idea that the more times a context appears, the more probable a new token will exist in that context.

Common steps:

1. Preprocessing the data and tokenizing it suitably.
2. Append and account for unknown("<UNK>") words.
3. Creating default dictionaries based on thr training data for n=1,2,3,4.
4. Create Language Models for the mentioned smoothing techniques.

Different steps:

1. In Kneyser-Ney, for lower n-grams, we use discounted counts.
2. In Kneyser-Ney, for higher n-grams, we perform absolute discounting by substracting a suitable discount value from the n-gram counts.

Model Performance:

1. For $n = 4$, we see that the perplexity score on the training data is better for Witten-Bell as compared to Kneyser-Ney.
2. For $n = 4$, we see that the perplexity score on the testing data is better for Kneyser-Ney as compared to Witten-Bell.
3. Similarly for the lower n-grams of n=1,2,3, we tabulate and observe that Kneyser-Ney performs better in comparison to Witten-Bell.

It is obvious that while Kneyser-Ney performs significantly better on the data in comparison to Witten-Bell, Witten-Bell runs significantly faster in comparison to Kneyser-Ney.
