Assignment 2
Autocorrect: Correcting Spelling Errors

In this assignment you will be training a language model to build a spell checker. Specifically, you will be implementing part of a noisy-channel model for spelling correction. We will give the likelihood term, or **edit model**, and your job is to make a **language model**, the prior distribution in the noisy channel model. At test time you will be given a sentence with exactly one typing error. We then select the correction which gets highest likelihood under the noisy-channel model, using your language model as the prior. Your language models will be evaluated for accuracy, the number of valid corrections, divided by the number of test sentences.

You will be using the writings of secondary-school children, collected by David Holbrook. The training data is located in the data/ directory. A summary of the contents:

- **holbrook-tagged-train.dat:** the corpus to train your language models
- **holbrook-tagged-dev.dat:** a corpus of spelling errors for development
- **count_1edit.txt :** a table listing counts of edits $x|w$, taken from Wikipedia. You don't need to modify any code which uses this.

Note that the data files do not contain <s> and </s> markers, but the code which reads in the data adds them.

Your assignment is to implement the following language models:

- **Laplace Unigram Language Model:** a unigram model with add-one smoothing. Treat out-of-vocabulary items as a word which was seen zero times in training.
- **Laplace Bigram Language Model:** a bigram model with add-one smoothing.
- **Stupid Backoff Language Model:** use an unsmoothed bigram model combined with backoff to an add-one smoothed unigram model
- **Kneser-Ney Language Model:** use an interpolated Kneser-Ney model with backoff to an add-one smoothed unigram model.

You can get inspired from the language models already provided: UniformLanguageModel.py and UnigramLanguageModel.py.

To implement a language model you need to implement two functions:

- **train(HolbrookCorpus corpus):** takes a corpus and trains your language model. Compute any counts or other corpus statistics in this function. See the example UniformLanguageModel for how to access sentences in the corpus and the words in those sentences.
- **score(List words):** takes a list of strings as argument and returns the numerical score, which should be the log-probability of the sentence using your language model. Use whatever data you computed in train() here.

Evaluation

Your language models will be evaluated on the development data set. The expected performance of each language model on the development set is listed below:

- **Laplace Unigram Language Model:** 0.110403
- **Laplace Bigram Language Model:** 0.135881
- **Stupid Backoff Language Model:** 0.180467
- **Kneser-Ney Language Model:** 0.182590.

Meeting the stated thresholds is not a guarantee of the correctness of the code. Please refer to the Guidelines document for specific hints on how to implement each of the language models.

Given Code

The rest of the scaffolding has been provided (reading corpora, computed edit probabilities, computing the argmax correction). A short summary of the remaining files:

- **SpellCorrect.py**: Computes the most likely correction given a language model and edit model. The main() function here will load all of your language models and print performance on the development data. It may be useful to comment out some of the tests in main() when developing.
- **EditModel.py**: Reads the count_1edit.txt file and computes the probability of corrections. The candidate corrections are all strings within Damerau-Levenshtein edit distance 1. The probability of no correction is set

at .9 ($P(x|x)=.9$). Note that the EditModel isn't great, but your language models will be evaluated using this model, so it won't affect your grade.

- **HolbrookCorpus.py**: Reads in the corpus and generates test cases from misspellings.
- **Sentence.py**: Holds the data for a given sentence, which is a list of Datums. Contains helper functions for generating the correct sentence and the sentence with the spelling error.
- **Datum.py**: Contains two strings, word and error. The word is the corrected word, and error contains the spelling error. For tokens which are spelled correctly in the corpus, error = ''.