Natural Language Processing:

Assignment 3: Follow my Words

Jordan Boyd-Graber

Out: **2. September 2014**
Due: **26. September 2014**

Introduction

As always, check out the Github repository with the course homework templates:

git://github.com/ezubaric/cl1-hw.git

The code for this homework is in the `hw3` directory.

1 Preparing Data (15 points)

We will use the Brown corpus (nltk.corpus.brown) as our training set and the Treebank (nltk.corpus.treebank) as our test set. Eventually, we'll want to build a language model from the Brown corpus and apply it on the Treebank corpus. First, however, we need to prepare our corpus.

1. First, we need to collect word counts so that we have a vocabulary. This is done by the `train_seen` function. Modify this function so that it will keep track of all of the tokens in the training corpus and their counts.

2. After that is done, you can complete the `vocab_lookup` function. This should return a unique identifier for a word, or a common "unknown" identifier for words that do not meet the `unk_cutoff` threshold. You can use strings as your identifier (e.g., leaving inputs unchanged if they pass the threshold) or you can replace strings with integers (this will lead to a more efficient implementation). The unit tests are engineered to accept both options.

3. After you do this, then the finalize and censor functions should work (but you don't need to do anything). But check that the appropriate unit tests are working correctly.

2 Estimation (45 points)

After you've finalized the vocabulary, then you need to add training data to the model. This is the most important step! Modify the `add_train` function so that given a sentence it keeps track of the necessary counts you'll need for the probability functions later. You will probably want to use default dictionaries or probability distributions. Finally, given the counts that you've stored in `add_train`, you'll need to implement probability estimates for contexts. There are four required probability estimates you'll need to implement:

- 5 `mle`: Simple division of counts for that observation by total counts for the context
- 5 `laplace`: Add one to all counts
- 5 `dirichlet`: Add a specified parameter > 0 to all counts
- 10 `Jelinek-Mercer`: Interpolate between probability distributions with parameter λ
- 20 `Kneser-Ney`: Use discounting and prefixes with discount parameter δ

Now if you run the main section of the `language_model` file, you'll get per-sentence reports of perplexity. Take a look at what sentences are particularly hard or easy (you don't need to turn anything in here, however).

3 Exploration (10 points)

Try finding sentences from the test dataset that get really low perplexities for each of the estimation schemes (you may want to write some code to do this). Can you find any patterns? Turn in your findings and discussion as `discussion.txt`.

Extra Credit

Extra Credit (make sure they don't screw up required code / functions that will be run by the autograder):

- 3 Implement Good-Turing Backoff (function called `good_turing`)
- 5 Implement a function to produce English-looking output (return an iterator or list) from your language model (function called sample)

<10 Make the code really efficient for reading in sequences of characters

5 Modify the code to accept an arbitrary n-gram length history (create a new class in a new file called `ngram_model.py` that takes the order as an argument)