Due Friday, April 5, 2019

N-Grams

Part 1 - Building NGram Models

Level 1 - may use NLTK **Levels 2/3 - may not use NLTK**

Write a script called **build_ngram_model.py**, that takes in an input file and outputs a file with the probabilities for each unigram, bigram, and trigram of the input text.

The script should run with the following command:

```
./build_ngram_model.py <input_file> <output_file>
```

The input file format is 1 sentence per line, with spaces between every word. (The files are pre-tokenized).

Add beginning of sentence (<s>) and end of sentence tags (</s>) and make everything lowercase.

For example, if a sentence is:

```
Hello , my cat !
```

You will count as if the sentence is written:

```
<s> hello , my cat ! </s>
```

The output file should have the following format: (see sample file: dickens_model.txt)

```
\data\

ngram 1: types=<# of unique unigram types> tokens=<total # of unigram tokens>

ngram 2: types=<# of unique bigram types> tokens=<total # of bigram tokens>

ngram 3: types=<# of unique trigram types> tokens=<total # of trigram tokens>


\1-grams:

<list of unigrams>


\2-grams:
```

<list of bigrams>

\3-grams:

<list of trigrams>

The lists should include the following, in order, separated by spaces:

Count of n-gram

Probability of n-gram (P(wn | wn-1) for bigram and P(wn | wn-2 wn-1) for trigram)

Log-prob of n-gram (take the log base 10 of the probability above)

n-gram

Do not use smoothing for this! Only include n-grams that exist in the training text.

Part 2 - Using NGram Models for Text Generation

Write a script, **generate_from_ngram.py,** that takes an ngram language model (output from the previous part, written to a file as described above), and outputs to a file 5 sentences generated from the unigram, bigram, and trigram models.

The script should run with the following command:

```
./generate_from_ngram.py <input_file> <output_file>
```

You will need to import **random** for this script.

Unigram:

Generate a random number from 0.0 to 1.0, and begin to count up the probabilities for the unigrams. When you reach the unigram whose probability sends the probability total above the random number, add that unigram to the sentence. Repeat.

Sentences should begin with <s> and end with </s>, and not have any <s>s or </s>s between the start and end.

Bigram:

Start with <s>. Generate a random number from 0.0 to 1.0, and begin to count up the probabilities of the second word, given <s>. When you reach the word whose probability sends the probability total above the random number, add that word to the sentence, and repeat with the bigrams that start with the word you generated.

Sentences should begin with <s> and end with </s>, and not have any <s>s or </s>s between the start and end.

Trigram:

Same idea as above, adapted to trigrams. Use the bigram generator to find the first word after the <s> of the sentence.

Part 3 - Perplexity - Level 3

Calculating the perplexity on a test set is one way of evaluating the effectiveness of the language model. Write a script called **ngram_perplexity.py**, that takes in an ngram language model as input (output of Part 1), lambda values, a test file and and output file and calculates the perplexity of the test file.

The script should run with the following command:

```
./ngram_perplexity.py <input_file> <lambda1> <lambda2> <lambda3>
<test_file> <output_file>
```

Perplexity can be calculated like this:

```
for each sentence in the test data file:
    add the number of words in the sentence (excluding <s> and </s>) to the total number of words
    for each word(i) in the sentence (excluding <s>, but including </s>:
        if the word(i) is unknown, increment an unknown word counter and continue
        Calculate the **interpolated** log-probability of the trigram as below:
            log( P(word(i) | word(i-2) word (i-1)) )
        Add this log-prob to a running total
divide the negative sum of the log-probs by the total number of words added to the number of sentences minus the number of unknown words.
Raise this value to the power of 10
```

Build a model using dickens_training.txt. Calculate the perplexity for the following lambda values on dickens_test.txt:

lambda 1 (unigram weight)	**lambda 2 (bigram weight)**	**lambda 3 (trigram weight)**	**Perplexity**
0.1	0.1	0.8	
0.2	0.5	0.3	
0.2	0.7	0.1	
0.2	0.8	0	
1.0	0	0	

Submissions!

You submissions should include the following files, named precisely as outlined here:

Level 1	Level 2	Level 3
build_ngram_model.py	build_ngram_model.py	build_ngram_model.py
generate_from_ngram.py	generate_from_ngram.py	generate_from_ngram.py
		ngram_perplexity.py
shakespeare_model.txt	shakespeare_model.txt	shakespeare_model.txt
dickens_generated.txt	dickens_generated.txt	dickens_generated.txt
readme.txt	readme.txt	readme.txt

readme.txt should include your favorite sentences from development, any general observations. Also, commentary on where you got stuck/got help/gave up/ what you found easy! And the perplexity calculations for level 3

I have provided dickens_model.txt so if you get stuck on Part 1, you can use this model to move on to Parts 2/3.