CSCE 5543: Homework 3

April Walker
adw027@uark.edu

University of Arkansas,
Fayetteville, AR, 72701, USA

Language Modeling from a Shakespearean Corpus

Introduction

In this assignment I explore language modeling by assigning probabilities to new sentences given a corpus and generating new sentences given a starting bigram.

I utilize code from both my previous assignments, while expanding on some functions. For example, `get_ngrams()` remained identical while `filter_bigrams()` was altered significantly for a specific action, and `clean_corpus()` was altered to identify sentence endings and cut out informational sections regarding copyright. Stop words were not removed as they seemed important to sentence structure.

Sentence Probability

For this section I developed a functionality to calculate the probability of a sentence or a phrase. If `call_sentence_probability()` was used, whether or not the user included punctuation, it was assumed to be a complete sentence. And thus the probability for a sentence S containing n words w could be calculated as:

$$P(S) = P(w_1| < s >) \times P(w_2|w_1) \times ... \times P(< /s > |w_n)$$

If instead, the user called `call_phrase_probability()`, the probability could be calculated as:

$$P(S) = P(w_2|w_1) \times P(w_3|w_2) \times ... \times P(w_n|w_{n-1})$$

Both results were documented for the test phrases given.

Now, for this portion of the assignement, I again wanted to reuse my code `get_ngrams()`. For a given input, the sentence of phrase was broken down into individual bigrams and put into a hashtable giving their counts. That is, the phrase "to be or not to be" would become a hashtable of "to be" "be or" "or not" and "not to" with each having a count of one except "to be" with a count of two. Looking at my previous equations, this would simply mean the conditional probability of "to be" should be multiplied twice, or more generally the conditional probability should be raised to the power equivalent to the counts indicated in the hashtable. Since we instead used log probability, the following equation was used:

$$\log(P(S)) = c(w_1, w_2) \cdot \log(P(w_2|w_1)) \times ... \times c(w_{n-1}, w_n) \cdot \log(P(w_n|w_{n-1}))$$

In this case $c(w_{i-1}, wi)$ refers to the counts in the *input statement* while $P(w_n|w_{n-1})$ refers to the conditional probability in the *entire corpus*. While this is given by the homework outline, I will clarify this conditional probability is calculated by normalizing the bigram counts table against the unigrams hashtable with the following formula:

$$P(w_n|w_{n-1}) = \frac{c(w_{i-1}, wi)}{c(wi)}$$

The final probabilities were found by taking the exponential of the log probability. Considering each of the following to be a phrase, the probability of "All the world's a stage" was calculated to be 1.514e^{-10}, for "to be or not to be" - 4.368e^{-10}, and for "astronaut Internet telephone - 8.315e^{-07}. Our relatively high probability for the later is simply due to our cap on conditional probability of e^{-7} for a word which does not exist. For the sentence probabilities, please reference `test.log`.

Shannon Visualization

For this portion of the assignment, given a bigram, an additional 8 words will be added to form a pseudo-Shakespearean phrase or sentence. To avoid a rather boring loop, the game picks the subset of corpus bigrams in which the first word matches the second input word. From this, the top 10% (based on conditional probability) are considered "contenders" from which was is picked via random number generator. If a word did not exist, I instead used the top 10% of unigrams to be the next word.

For rather minimal processing, and little concern for sentence structure (tags and etc.) the results were decent... (with some more decent than others). Below are some examples of my results:

> to sleep a villain be at his power to marry
> to sleep a pretty wrongs that life is his mistress
> to sleep a stranger of youth and mine ear with
>
> love is more worth than is of caesar agrippa dolabella
> love is this it in his sword to see this
> love is no no further pleasure she will in such
>
> love twitter home to whom we be for one is
> love twitter top of them i take her lips to
> love twitter caesar with other citizens menenius and make thy

Additionally, here are a few more radom ones messing around with the number of additional words $n = 8$ qualifier...

> love is it for which we see your city we must attend
>
> love twitter live in mind that all i be thou mightst have
>
> to sleep and hath made him as any tricks which makes a womans part of nature hath brought
>
> love is done and thy sight of laughter let me my husband to milford haven

Function Timing

The most time consuming portion of the code was in normalizing the bigrams to get the conditional probabilities. This took approximately 13.3 seconds. This, however, is cut in half if bigrams are not filtered to disclude sentence endings. This component was added in to allow either phrase of sentence likelihood calculation. Following much farther behind, preprocessing took about 0.81 seconds, developing the bigram hashtable took approximately 0.15 seconds, the unigram hashtable 0.10 seconds, sentence probability calculations were negligible (<0.01 seconds), and the Shannon Visualization took approximately 0.23 seconds per call.