Author: Ryan Timbrook
UW Net ID: timbrr
Project: Ling 473 Project 2
Date: August 16, 2016

Description:

This project is to write a program to clean a corpus and tally the words in it. The resulting output is called a unigram language model. The files being analyzed are part of the AQUAINT corpus of English newswire. The sample directory contains 249 files containing New York Times news articles in SGML format.

Approach:

For this assignment I took a procedural programming approach using Python. It appeared apparent after various attempts at cleaning the SGML formatted text using regular expressions that the cleaning process would need to be achieved through a series of steps using multiple regular expression transactions.

The first step uses a custom function to clean all SGML tags from the document. Its input parameter is a reference to the data file being processes. This function uses a regular expression pattern (r'<.*?>') to find all tags then replace these tag elements with an empty character (''), returning a single string Object representing all elements of the document delimited by a single white space character (' ').

The second step in the cleaning processes uses a custom function to remove non-word elements from the document. This was done through a two-step process. The first step used a regular expression to find and replace, with an empty string, elements which did not fit the specification to only accept a word as a contiguous occurrence of one or more acceptable characters (i.e. Capital letters A-Z, Lower-case letters a-z, and the straight apostrophe) . Note that this specification was not able to differentiate and clean non-lexical elements which fell into the above specification. A couple examples include spelling errors, and single character elements which are not words but could be part of the SGML header or other technology specific specifiers. The substitution regular expression used was:

r"([a-zA-Z]+[0-9]+)|([0-9]+[a-zA-Z]+)|(&[a-zA-Z]+)|(\w+(-\w+)+)|([a-zA-Z])(\.[a-zA-Z])+\.?"

- *([a-zA-Z]+[0-9]+)*
  - *Find all elements that are alpha numeric in that sequence*
- *([0-9]+[a-zA-Z]+)*
  - *Find all elements that are numeric alpha in that sequence*
- *(&[a-zA-Z]+)*
  - *Find all elements which are non-words that start with &*
- *(\w+(-\w+)+)*
  - *Find all hyphenated words*
- *([a-zA-Z])(\.[a-zA-Z])+\.?*
  - *Find all abbreviations such as N.J.*

The second step in this cleaning process was to find all elements including those with an internal apostrophe. The regular expression used was: r"[a-zA-Z]+'?[a-zA-Z]+". This expression matches on one or more alphas followed by an optional apostrophe followed by one or more alphas. The last step in this function was to convert all acceptable elements to lowercase and return these elements in a list Object.

The final step tallied each instance of the elements returned by the clean functions and stored it to an in-memory dictionary collections Object. The procedure using a dictionary object to store these tallies after each document was cleaned resulted from a few trial and errors in my procedural designs. I had first attempted to join each flatten document to a single String Object in the main procedure then use the collection module's Counter class to tally all of the elements. This procedure resulted in critical "Memory Error" terminating the program at approximately 180 documents processed. Clearly memory and speed are two critical conditions which have to be taken into consideration when processing large data sets.