Tracy Sandifer

INFX 575 Assignment 2: Natural language processing

April 28, 2015

The provided set of ten patent claim files provides material for developing a process to compare the similarity of their content. I don’t know if patent evaluation processes actually include automated natural language evaluation algorithms, but it does seem to be an area that could benefit from some level of automated screening at least, given the high volume of submissions and how subject the judgement of similarity may be from one evaluator to the next. The steps required for this assignment cover the initial data preparation steps; it would be interesting to investigate what sort of analytic tests might be applied to these parsed and compiled lists to flag sets of patent claims containing language that is more similar to each other than would be expected.

To carry out these data preparation steps, I downloaded the ten patent claim files from the instructor’s S3 space using the URLs provided. All subsequent steps are handled in one Python file designed to perform a set of functions iteratively for each file.

The basic goal of these initial data preparation steps is to glean the meaningful language content from the uninformative content and to format it in a way that can be used for analysis. The first step in separating informative from uninformative language was to remove “stop words”. Given that there is no universally accepted set of stop words, and I am not experienced in this type of analysis, I was interested in finding a fairly conservative (minimal) set of stop words to err on the side of leaving extra words, knowing that I could refine and remove additional stop words in later rounds of analysis as I became more familiar with the data and the task at hand. I found a list of 429 stops words on LexTek.com (<http://www.lextek.com/manuals/onix/stopwords1.html>) that seems to serve this purpose, and saved it as a text file to access in my Python script. Prior to comparing the patent text to the stop words, I removed anticipated instances of characters sets that are unrelated to patent concepts, such document organization structures like “a), b), c)”, numbers, and punctuation. Since my list of stop words is lowercase, I made all remaining characters lowercase. Next, I split the text into component words, and compared each word against the stop word list, removing any matches from the patent claim word set.

The next step involved “stemming”, or removing word endings in an attempt to consolidate words that diverge from different parts of speech but that actually represent the same concept. Again, having no basis in experience for knowing what a great or poor stemming approach might be, I opted to use the one provided in Python’s natural language toolkit, largely for ease of use. The output from this step is a list of single, stemmed words from which stop words have been removed.

These two initial steps produce in the initial, noise-reduced word set to carry forward into an analysis of choice. In this case, we were asked to produce frequencies of n-grams of size one, two, and three. The stemming step produced a list of unigrams; bigrams are derived from iterating over the list of unigrams, storing an item and the next one into each bigram list item. Similar, trigrams were produced by iterating over the list of unigrams, storing an item and the next two into each trigram list item.

Finally, the Python collections Counter function counts the frequency of each unique list item in the unigram, bigram, and trigram lists produced. The final step in the Python loop writes the Counter key, value pairs to a local file and copies that file to the S3 space. The script also iteratively builds all-files lists of unigrams, bigrams, and trigrams with each pass over the loop, and after exiting the loop applies the Counter function, file writing, and file copy processes to the final all-file lists.