Assignment 3:
Environment sensitivity

Assigned 6/30/19
Due 7/9/19

Paul Crowley

1 Introduction: context matters

The first assignment was primarily focused on determining the frequency of certain words that we care about in a text. This can be useful but it is very simplistic in that it does not take into account the environment in which words that we care about appear.

Imagine that we care about the use of words expressing positivity and negativity in a text. Now, consider the single-sentence texts text1 and text2. Both texts have a positive word count of 1 and negative word count of 0—the only such polar word is 'good' which is positive—but one is clearly positive and one is clearly negative.

text1 = 'The results where good.'
text2 = 'The results were not good.'

If we are focusing on positive and negative words as an indicator of overall positivity and negativity of a text, then it is not that we can only care about frequency of words but we must care about their environments as well. In text2 the positive word 'good' appears in a context that reverses the semantics of 'good', created by 'not'.

If we had thousands of lines of elicited text from a candidate for an open position in a company, then the total frequencies of positivity and negativity may no be thrown off too much with cases like text2, given a simplistic word-counting program. However, given that eliciting thousands of lines of text from a candidate is not in the realm of possibility, we will only have a relatively small amount of text to feed our NLP system. This means that it is important for what we build to be sensitive to the environments in which key elements are featured in order to make the results as accurate as possible.

2 The task

In order to complete the following task, you should read NLTK chapters 4-8. Keep the task in mind while you read so you can detect any useful knowledge, features of the NLTK API, etc.

2.1 Part 1: positive and negative words

Compile a set of 250 common generally positive words (glad, exciting, useful) and a set of 250 common generally negative words (fail, frustrate, boring).

You can make use of any resources that you find online to compile the sets, or do it manually yourself with brute force mental power (probably don't do this), or a combination of the two (this is better).

2.2 Part 2: programming environment sensitivity

Write a program that can recognize and track positive and negative *sentences*, not just positive and negative words.

You must determine what the computer should do when it confronts...

- a positive or negative word that is not in a reversal context
- a positive or negative word that is in a reversal context

...in order to accurately determine overall positivity and negativity of the text.

In order to do this you must determine what a reversal context is and determine how to make the computer sensitive to such contexts in Python.

This will involve initially determining a collection of important functional elements that will reverse the positive or negative effect of positive and negative words respectively, when they co-occur with these words.

- *not* is one such element, as illustrated above, but there are others.

Additionally, you must determine what relations these functional elements must be in with respect to the positive and negative words in order for them to have the reversal effect—is it sensitive to hierarchical syntactic structure or is it a linear relation in the phonological string?

- you will learn about hierarchical syntactic structures in the NLTK book

Your work should be informed by this short theoretical paper which discusses important formal logical properties of such functional reversal elements, that will help you determine which elements your program should care about.

- `http://rdues.bcu.ac.uk/repulsion/negcontexts.pdf`

Your program should capture simple cases like text2 but think about other cases that would be relevant here. Think of other types of sentences that feature one or more reversal elements in different positions.

If you want your program to be sensitive to syntax, you can use the syntactic parser introduced in NLTK or, alternatively, you can download and run the Stanford parser provided here, which is a good alternative and yields a slightly easier to read phase structure output (in my opinion).

- `https://nlp.stanford.edu/software/lex-parser.shtml`

If you find another open source parser that you like (there are many—there are even various Stanford parsers) then feel free to use that and share whatever resources you found with the group.

Test the program that you write against five chapters from five different books from five different authors that you can find in simple text format. The books should preferably be of different types, e.g. a biography, three novels and a text book. Let everyone know if you find a good resource for books in simple text.

This task is a creative and open-ended one. There will be a variety of different ways to implement a system that achieves the target goal here. Feel free to find any resources that you want on to help you find the/a solution (you very likely won't find a solution anywhere, sorry).

Let me know if you have questions, find any resources that you like, etc.