Natural Language Parsing

by Michael Trittin and Andrew McKee

Abstract

In this paper, we will discuss our natural language parsing (NLP) project, including its capabilities, a summary of its successes and failures, data structures used to store documents and queries, and ideas for improvement which we would like to achieve.

General Capabilities

Our NLP project uses the Stanford CoreNLP project in order to parse complex sentences and search them to varying degrees of accuracy for a given search query. By using an NLP library such as CoreNLP, we are able to match very long and complex sentences to varying phrases, declaring the sentence to either be a match or not. On the surface, the output of our program is simplistic, being only a 'yes' if the document given is determined to have a sentence which matches the query, or a 'no' if there is no such sentence or (more powerfully) if the statement is directly contradicted. However, with more fine tuning we believe that our project could achieve more, such as extracting statements from the search document that closely match the query and highlighting which part of the query is the most interesting. However, we are greatly limited by the speed of the Stanford CoreNLP library and its memory constraints when processing larger documents. For our examples in this document, we will primarily be discussing some smaller text documents or sections of larger documents which the parser does not have trouble handling. This being said, we did manage to run the parser on larger documents such as the Abraham Lincoln Wikipedia article by splitting the article into sections and considering them separately. That being said, were we to move on with the project we would likely start over with a different NLP library that doesn't suffer from the same problems. In terms of total words our project can handle, the issue is primarily one of time and memory: we were only willing to let the parser work on around 10-15 sentences at a time: any more, and the time/memory growth seemed to be exponential.

Successes and Failures

We found our project to be highly capable compared to our earlier attempts at rudimentary NLP using skip-bigrams and the bm25 algorithm. Unlike those approaches, this project is capable of far more flexibility with regards to sentence analysis, and in general can be used as a semi-reliable fact-checker rather than a simple word search. The power of our project comes from its ability to prioritize words according to their importance in a sentence: adjectives, for example, are almost never as important as a proper noun, and without recognizing these different parts of speech any document searching algorithm will always be lacking since it will weight each word the same. Another highly powerful part of our approach lies in the ability to determine what a pronoun might refer to, and attach descriptions of a proper noun even despite a gap of several sentences.

In order to demonstrate how this works in our project, consider the following document which we will query for validity:

```
Abraham Lincoln was the 16th president of the United States. He led
the country through its greatest period of moral, political, and
constitutional crisis. He didn't live to see the results of his
labours due to his untimely death.
```

Were we to analyze the above document using a more naive approach (i.e. bm25 and skip-bigrams), we may find some counter-intuitive results. For example, if we were to search for the phrase "He did live to see the results of his labours", we would find that bm25 and skip-bigrams rate this document relatively highly, considering that the query is contained (almost exactly) within the above document. However, the bm25 and skip-bigrams are only noting that something resembling the sentence is within the document: it does not account for the fact that the document says he did *not* live to see the results of his labours. Additionally, if we were to change our query to instead say "Abraham Lincoln did live to see the results of his labours", our bm25/skip-bigram approach would lower its confidence score in the document because it is unaware that the "He" refers to Abraham Lincoln.

Our NLP project on the other hand handles the document above rather well. Using the Stanford CoreNLP library, we were able to take the sentences and use what are called Coref Chains to link pronouns and adjectives to the nouns they refer to. Additionally, we use the graph structure produced by Stanford CoreNLP as the basis of our search, and compare the graph of the parsed query to sections of the parsed document in order to determine accurate statements. This gave us an advantage in that not only is the word "he" synonymized in our search with Abraham Lincoln, but we also recognize negation in the form of "didn't", and score the phrase appropriately. Thus search for "Abraham Lincoln did live to see the results of his labours" scores lower than "Abraham Lincoln didn't live to see the results of his labours", but both score as positive enough for our project to answer "YES" in a search since most of the words are contained in both versions.

Another advantage of our system is that it is capable of basic stemming. For the example above, the word "didn't" and "did" are both matched despite the addition of the contraction. This is due primarily to the power of Stanford CoreNLP.

In terms of failures, as we have already mentioned our project fails on large documents like a Wikipedia article. Without processing significant document preprocessing, it is in fact incapable of working without significant time and memory due to exponential time and memory complexity of the CoreNLP project. As far as we can tell, our own part of the code is not at fault for this as once the document is processed by CoreNLP, the searches are nearly instantaneous (<1s). Additionally, we require some modifying of the threshold for detection depending on how tolerant you want to be of a statement's accuracy. By default, we set the tolerance to be 0.35, which is rather low so that if something is even close to matching, the parser will say "YES". However, if you increase this tolerance, you can more accurately filter an argument to say whether the exact statement is contained.

Data Structures

In terms of data structures, we primarily translated the graphs given to us by the CoreNLP library to our own purposes. These graphs are in some ways similar to the frames produced by the IBM Watson project, but they require more processing to get the same detailed information. An example of one such graph translated into a string might be:

```
(ROOT (S (NP (PRP$ My) (NN dog)) (ADVP (RB also)) (VP (VBZ likes) (S
(VP (VBG eating) (NP (NN sausage))))) (. .)))
```

Each word in this sentence is also analyzed and linked using a Coref Chain Parser, which links prounouns and adjectives to nouns and shows the relations between them.

After CoreNLP has parsed the sentence and produced the above graph/associated Coref Chains, we reorder the graph to better suit our purposes. Specifically, we use a Comparator to sort the graph in such a way that relations are scored more significantly than governors, and governors are scored more significantly than dependents. An example of one such re-ordering can be seen below:

```
He led the country through its greatest period of moral, political,
and constitutional crisis.
-> led/VBD (root)
  -> He/PRP (nsubj)
  -> country/NN (dobj)
    -> the/DT (det)
  -> period/NN (nmod:through)
    -> through/IN (case)
    -> its/PRP$ (nmod:poss)
    -> greatest/JJS (amod)
    -> moral/JJ (nmod:of)
      -> of/IN (case)
      -> ,/, (punct)
      -> political/JJ (conj:and)
      -> ,/, (punct)
      -> and/CC (cc)
      -> crisis/NN (conj:and)
        -> constitutional/JJ (amod)
    -> political/JJ (nmod:of)
    -> crisis/NN (nmod:of)
  -> ./. (punct)
```

Ideas for Improvement

First and most importantly, we would like to transition to a different NLP library but did not find the time to do so. CoreNLP is very powerful, but also very slow when it is processing Coref Chains. The architecture of CoreNLP is also somewhat difficult to work with, but we imagine this is due to the power the library possesses.

We would also like to make our HTML parser work better on the documents we are given to better determine when to split it for processing. As it stands currently, the HTML parser simply splits a wiki article on an "h2" so that each chunk is a manageable size, but sometimes this doesn't work out. For the Abraham Lincoln wiki article, there were certain chunks that were much too large for the parser to handle, but some of this may have been due to the fact that the parser encountered difficulty with the inconsistent of the HTML.