NLP Write-up

Stanford NLP API Properties

For this project, we used Stanford's CoreNLP API. The method of operation is annotating a provided text. The annotations correspond to different parts of speech and their position in the overall sentence structure. Additional annotators can recognize if a number is part of a date, proper nouns for people and locations, temporal words, and coreference between pronouns and their corresponding nouns in preceding sentences. We chose not to use the second set of annotators due to memory constraints. The program would run out of memory after parsing roughly 10% of the document with the additional annotators included.

Other modifications made include limiting the length of each sentence parsed to 70 words and to interpret two consecutive newline characters as the end of a sentence. These changes were made to account for the paragraph breaks and tables present in the text, as well as further reduce memory and time used. After all these modifications, the program took 13 minutes to execute with the whole Lincoln document so we tested it using chunks of text.

Internal representation

The internal representation of the text is an object that represents each sentence. This object has fields corresponding to different parts of speech in the sentence (subject, object, verb, preposition, and modifier). These fields are determined by the tag that the parser annotates the word with in the parse tree. Clauses are assumed to have a noun phrase and a verb phrase. This caused some sentence types to not parse correctly.

Search procedure

First, the query and document text is parsed and their internal representations are produced. The parser does a roughly good job of breaking down the sentences, however there is room for a lot of improvement to deal with the intricacies of language. Only one sentence can be input as the query. The query's internal representation is compared to all data internal representations produced using a comparison function.

If the query contains the word, "lincoln" or "Lincoln" but doesn't contain the word, "he", then "he" is added to the query's internal representation (subject) because the wikipedia entry uses the word, "he" very often to refer to Lincoln. The coreference annotator that could identify who "he" refers to was responsible for most of the memory usage since it scaled exponentially with document length so it was a much more efficient to use the above technique since we are only concerned with the Lincoln document in this assignment.

Comparison Function

While comparison, porter's algorithm is used to assure that the root words are compared. The porter's algorithm is used while comparison rather than while saving the internal representations so that when the scoring function is bettered in the future, it can rank exactly matching words higher than words with matching roots.

The comparison function produces a score for each internal representation corresponding to the document's sentences, by checking for similar words the document internal representation and the query internal representation. The score is incremented more if the matching words are of the same type (subject-subject) and lesser depending on the matching types (e.g. subject-object match does not increment the score at all because the sentences might be saying different things if they have a different subject and object but a modifier-verb match can lead to incrementing the score a little).

The yes-no score threshold is set according to the query size since bigger queries are more likely to have bigger scores. The threshold function is currently a little strict and has been arbitrarily determined. The threshold function could be modified by inputting expected answers, running them with some queries and tweaking the function to respond best (Q-learning). However it would probably make more sense to write a better parser for the internal representations first.

Comparison with BM25 and Skip-Bigrams

The BM25 and skip-bigrams were used to check which documents were most likely to have information relating to a query. This project needed us to give a yes-no answer to a statement. The criteria for the two assignments were somewhat different. We could not use BM25 and skip-bigrams to find if a statement is correct or not because they can only identify if some words exist in a document. They cannot check if the words are in the same sentence or section (skip bigrams can do that but in a very restricted way). Also, we could add code to the NLP assignment to check for negations, but we can't do that with BM25 or skip bigrams. NLP is more powerful in checking for specific information in the document though it takes a lot of memory and time wheres BM25 and skip bigrams are better to determine if the needed information is roughly related to the document, and are quicker than NLP in doing this job.

Example Sentences with correct answer, yes.

1. Lincoln was assassinated.
   Note: Changing the above sentences from statement to question form, "Lincoln was assassinated" to "Was Lincoln assassinated?" for example, had no effect on the result.
2. Lincoln was president.
3. Lincoln was president from March 1861 to April 1865.
4. Lincoln died April 15.

Example sentences with correct answer, no.

1. Lincoln was president from March 1862 to April 1866.

Example sentences with wrong answer, no.

1. Lincoln was born in Kentucky.
   Reason: We make internal representations assuming that any clause (S tag) has a noun phrase (NP tag) branch and a verb phrase (VP tag) branch, so for e.g. the sentence below from Wikipedia, the "Born in Hodgenville, Kentucky" phrase is not processed and saved in any internal representation.
   "Born in Hodgenville, Kentucky, Lincoln grew up on the western frontier in Kentucky and Indiana."