LING 450 - Assignment 1
Prof. Maite Taboada
Mar 1, 2024

Group Member	Student Id	Contribution 1	Contribution 2
Monisha Mohandas	301384925	Approach 1 coding & write up	New Approach
Mithula Barua	301217522	Approach 2 coding & write up	Comparison of Approaches
Olivia Calton	301291735	Approach 3 Gender Gap Tracker Implementation by ATAP	Formatting & Editing

APPROACH 1: REGULAR EXPRESSIONS

Describe the approach. What does the code do?
The code utilizes regular expressions to extract text within double quotes (" "). It reads text files, searches for text enclosed in double quotes, and extracts those phrases as quotes.

Describe the output. Does it do what you expected?
The output consists of all the direct quotes extracted from multiple text files. It lists the extracted quotes along with their respective file paths. The output aligns perfectly with expectations, as it displays all the quotes found within the text files.

Describe the output given the task. Does it capture all the direct quotes in the text? Why or why not? Does it capture things that are not direct quotes? Why or why not?
Yes, the output captures text enclosed within double quotes, which generally represents direct quotes in the text. However, it may not capture all direct quotes, especially if the text contains unconventional formatting or variations in quotation styles (e.g., single quotes instead of double quotes). Additionally, it may capture non-direct quotes that happen to be enclosed within double quotes, such as headings, titles, or other types of text.

If not all the direct quotes are captured, what is missing and how could you fix it?
If not all direct quotes are captured, it could be due to variations in formatting or quotation styles. To improve the extraction process, the regular expression pattern could be modified to account for different quotation styles or additional formatting variations. Additionally, refining the pattern to exclude non-quote text enclosed within double quotes can help improve the accuracy of quote extraction. Regular expression patterns can be adjusted to capture specific text patterns more effectively. Another way to evaluate the accuracy of the extracted quotes is by comparing them to the original text. Additionally, you could assess the completeness of the extracted quotes by verifying that all relevant direct quotes from the text files are indeed included in the output. If the extracted quotes accurately represent the direct quotes from the text files without missing any, then the approach can be deemed successful in its task.

APPROACH 2: SpaCy Matcher for pattern

Describe the approach. What does the code do?
Approach 2 uses spaCy's Matcher to look for patterns. The code is processing the text of our file with spaCy by defining the matcher pattern (we assigned it to a matcher object). The matching pattern looks for quotation marks. It will look for matcher.add that looks for the longest pattern match. After the pattern is implemented into the matcher with our doc it will give us the result for the matched quotations for our specific pattern. The last code will find you the loop result of the first 10 matches. It will extract only the 10 matches to our pattern.

Describe the output. Does it do what you expected?
The result of the SpaCy matcher with a simple pattern to extract things in single quotes was 0 in all of the 5 selected text files. Finding the quotes with all 0 results is not what I expected because the 5 files selected had quotes and punctuations in the text files.

Describe the output given the task. Does it capture all the direct quotes in the text? Why or why not? Does it capture things that are not direct quotes? Why or why not?
Looking at the given output of the SpaCy matcher the code is supposed to capture the defined pattern in pattern_q. The given task was to look for double quotations with {'ORTH': ' " '} and 'ORTH': ' " '}]. However, when we look into the defined pattern for quotation, I notice that it does not capture all direct quotes in the text because the quotations in the text files did not match our defined pattern. When you

look into the files, you do notice that the quotation mark of a text is shown “ instead of " . For example, text file (5c1452701e67d78e276ee126) “I was clear when I was mayor, I don’t support Uber at all”. The quotation marks do not match our matched pattern. Therefore, it did not capture all the direct quotes from the text file. It did not capture things that are not direct quotes because the pattern only looked for quotation marks that are in the loop of 10 first searches.

If not all the direct quotes are captured, what is missing and how could you fix it?
The direct quotes were not all captured but we can fix it by redefining the SpaCy matcher pattern. We would implement a pattern that would address the change in quotation marks. As shown in approach 2 section of the codes. A redo of all the five selected text files were redefined in the new pattern with the corrected quotation marks that were found in each of the text files. The redo of the pattern would be pattern_q = [{'ORTH': '“'}, {'IS_ALPHA': True, "OP": "+"}, {'IS_PUNCT': True, "OP": "*"}, {'ORTH': '”'}]. Also, we can still use the previous Spacy Matched pattern and could add the redefined pattern as a long code and that would give us all the results with the open and closed quotation marks that are varied in all text files with the old and new pattern combined and the first 10 loop of all combination would also give us the SpaCy matched pattern.

APPROACH 3: GENDER GAP TRACKER TOOL IMPLEMENTATION

Describe the Approach. What Does the Code Do?
The approach employs a Jupyter Notebook titled QuotationTool for extracting quotes from textual documents using natural language processing (NLP) techniques facilitated by the spaCy library. It systematically processes texts to identify and extract quotations, speakers, and related contextual information such as named entities and their positions within the text. The tool stages include initialization, data loading, quote extraction based on linguistic patterns, and the visualization or saving of results. This structured methodology enables efficient analysis of textual data for research, journalism, or data analysis purposes by automating the otherwise labor-intensive task of quote identification and extraction.

Describe the Output. Does It Do What You Expected?
The output from this tool is a detailed compilation of extracted quotes presented in a pandas DataFrame, which includes information about the quotes, speakers, their locations within the text, and any named entities associated with them. Additionally, it offers a visualization component for better understanding the relationship between quotes, speakers, and entities. This outcome meets expectations for a tool designed to parse and analyze textual data, providing a clear and organized way to review and utilize extracted quotes for further analysis, reporting, or academic study.

Describe the Output Given the Task. Does It Capture All the Direct Quotes in the Text? Why or Why Not? Does It Capture Things That Are Not Direct Quotes? Why or Why Not?
While the tool is designed to capture direct quotes from the text efficiently, its performance might vary depending on the complexity of the text structure and the consistency of quoting practices used in the documents. It's likely that most direct quotes are captured accurately, but there may be instances where the tool misses quotes due to ambiguous linguistic patterns or captures text that isn't a direct quote, mistaking certain expressions or idiomatic uses of quotation marks for quotes. These limitations are inherent in automated NLP-based tools, which must navigate the nuanced and variable nature of human language.

If Not All the Direct Quotes Are Captured, What Is Missing and How Could You Fix It?
If the tool fails to capture all direct quotes, it's likely missing quotes embedded in complex sentence structures or those that don't conform to the patterns the tool is trained to recognize. Improving its

performance could involve refining the NLP model to better understand and interpret a wider range of linguistic cues for quoting. Additionally, incorporating more sophisticated rules or algorithms that can account for the variability in quoting practices across different text types and languages might help. Regular updates and training with diverse datasets can enhance the tool's accuracy in identifying quotes, reducing the likelihood of missing direct quotes or incorrectly including non-quote text.

COMPARISON OF THE APPROACHES

The first approach focused on the regular expressions to extract text within double quotes (" "). It reads text files, searches for text enclosed in double quotes, and extracts those phrases as quotes.The output consists of all the direct quotes extracted from multiple text files. It lists the extracted quotes along with their respective file paths. Whereas, approach 2 uses spaCy's Matcher to look for patterns.The code is processing the text of our file with spaCy by defining the matcher pattern (we assigned it to a matcher object). The matching pattern looks for quotation marks and punctuations that are within the loop for the first 10 searches. In approach 3, we are extracting quotes from textual documents using natural language processing (NLP) techniques facilitated by the spaCy library. It systematically processes texts to identify and extract quotations, speakers, and related contextual information such as named entities and their positions within the text. Hence, all three approaches use different techniques to extract quotes. Approach 1 and 2 uses two different ways of extracting simple quotes from text using the spaCy language model. Approach 1 with regular expression (finding text within text) and Approach 2 with Spacy matcher Whereas, Approach 3 focuses on extracting quotes and implementing it using the Gender Gap Tracker by the Australian Text Analytics Platform.

Describe what it would take to implement indirect quotes (reported speech). Refer to the parsing module and use NLP concepts and terminology. Use at most 1 more page.

Implementing indirect quotes, or reported speech, in natural language processing involves several key steps and concepts. First, it requires the use of dependency parsing techniques to analyze the grammatical structure of sentences and identify the relationships between words. Central to this process is the recognition of communication verbs, such as "said" or "mentioned," which serve as indicators of reported speech. These verbs anchor the extraction process by signaling the presence of reported content. Next, identifying reported speech structures entails recognizing syntactic patterns commonly associated with indirect quotes, typically involving a reporting verb followed by a subordinate clause introduced by a conjunction like "that." Extracting the reported content involves parsing the subordinate clause and retrieving the relevant phrases or clauses that constitute the reported speech. Moreover, handling complex sentences with multiple clauses and addressing ambiguity in linguistic expressions are essential aspects of implementing indirect quotes effectively. Integration with natural language processing tools, such as spaCy or NLTK, provides access to pre-trained models and APIs for dependency parsing, facilitating the analysis of grammatical structures and the extraction of indirect quotes from text data. Overall, implementing indirect quotes requires a combination of linguistic knowledge, syntactic parsing techniques, and the utilization of NLP tools to accurately identify and extract reported speech structures.