CS6370 – Natural Language Processing

Assignment 1

Teammate 1: R Narendhiran CH18B015
Teammate 2: Akiti Gunavardhan Reddy CH18B035

1.What is the simplest and obvious top-down approach to sentence segmentation for English texts?
Ans. The simplest top-down approach which we can think of to segment sentences is by spliting the given discourse at periods **{'.'}** and other punctuation marks such as **{'?', '!', ';'}**

2.Does the top-down approach (your answer to the above question) always do correct sen-tence segmentation? If Yes, justify. If No, substantiate it with a counter example.
Ans. No, It doesn't. In general English texts have many abbreviations, commonly used short forms and numbers with decimals which have periods and the top-down approach mentioned above incorrectly segments into different sentences.
Example Text :
'"Did he get 95.6 marks in his B.Tech Project?" asked Prof. Rakesh. Who was surprised'
From Top-Down approach split is:
['"Did he get 95.', '6 marks in his B.', 'Tech Project?', '"asked Prof.', 'Rakesh.', 'Who was surprised']
But Ideal split would have been:
['"Did he get 95.6 marks in his B.Tech Project?" asked Prof. Rakesh.', 'Who was surprised']

3. Python NLTK is one of the most commonly used packages for Natural Language Pro-cessing. What does the Punkt Sentence Tokenizer in NLTK do differently from the simpletop-down approach? You can read about the tokenizer here.
Ans. Punkt Sentence Tokenizer uses an unsupervised algorithm to build a model which can identify shortcomings of topdown approach which are abbreviation words, also collocations and words that start sentences.
Punkt achieves this by two-fold process (Type-based and Token-based classifier).
The type-based classifier initially distinguishes an abbreviation (or initials/ordinal numbers) from normal words. The token-based classifier is more of a refining stage (for example, abbreviations which occur at end of sentences are rightly identified). Both the classifiers employ log-likelihood ratios to determine the collocational bond between an abbreviation and a period[1].
This bottom-up approach which is already trained on a large text-database of corresponding language is used in our search engine application.[2]

4. Perform sentence segmentation on the documents in the Cranfield dataset using:
(a) The top-down method stated above
(b) The pre-trained Punkt Tokenizer for English
Ans. Refer Code for implementation

State a possible scenario along with an example where:
(a) the first method performs better than the second one (if any)
(b) the second method performs better than the first one (if any)
Ans. (a) There are times when our top down approach performs better than Punkt Sentence Tokenizer, as the Texts cannot always be perfect there might be some spacing error like for example text: 'I know you.We studied in same school'. This is split by our top-down approach as ['I know you.','We studies in same school'] but split by Punkt as ['I know you.We studied in same school'] where it failed to grasp the error.

(b)Here as well there are many cases where second method outperforms the first in case of all abbreviations for example text: 'I scored 99.9 percentile in I.I.T exam' is split by second method as ['I scored

99.9 percentile in I.I.T exam'] but is wrongly split by our second method as ['I scored 99.', '9 percentile in I.','I.', 'T exam']

5. What is the simplest top-down approach to word tokenization for English texts?
Ans. For the top-down approach we can use the knowledge that in general words are separated by blankspaces. So, we can split the words in the sentences separated by blankspace to tokenize words in the sentence.

6. Study about NLTK's Penn Treebank tokenizer here. What type of knowledge does it use- Top-down or Bottom-up?
Ans. Penn Treebank tokenizer is a Top-down model, as it uses knowledge based on expressions which occur regularly in English texts. Based on which it performs the following four operations.
i) Spliting of standard contractions such as "couldn't" to "could n't", "I'll" to "I 'll"
ii) Treating most punctuation characters as separate tokens.
iii) Spliting commas and single quotes when followed by whitespace.
iv) Separating periods which appear at the end of the line.[3]

7. Perform word tokenization of the sentence-segmented documents using
(a) The simple method stated above
(b) Penn Treebank Tokenizer
Ans. Refer Code for implementation

State a possible scenario along with an example where:
(a) the first method performs better than the second one (if any)
(b) the second method performs better than the first one (if any)
Ans.(a) None as both initially segregates the words based on commas but second method does more checks for contractions.

(b) In our first method we are only taking words separated by blankspace and selected punctuations, which in itself is huge assumption that all things separated by blankspace are words. For example take the sentence 'He finally answered (after taking five minutes to think) that he did not understand the question.' our first method separates the tokenizes as ['He', 'finally', 'answered', '(after', 'taking', 'five', 'minutes', 'to', 'think)', 'that', 'he', 'did', 'not', 'understand', 'the', 'question.'] but '(after' is not a word nor is 'think)' but this is taken care by our second method, said example by second method is tokenized as ['He', 'finally', 'answered', 'after', 'taking', 'five', 'minutes', 'to', 'think', 'that', 'he', 'did', 'not', 'understand', 'the', 'question.'].
There other cases as well such as contractions example text "I don't wanna go", from second method its split as ['I', 'do', 'n't', 'wanna', 'go'] which is correct but from first method we get ['I', 'don't', 'wanna', 'go'] where do + not, don't isn't split.

8. What is the difference between stemming and lemmatization?
Ans. Stemming algorithms work by cutting off the end or the beginning of the word, taking into account a list of common prefixes and suffixes that can be found in an inflected word. This indiscriminate cutting can be successful in some occasions, but not always, and that is why we affirm that this approach presents some limitations. And the stemmed word need not always be in dictionary
Lemmatization, on the other hand, takes into consideration the morphological analysis of the words. To do so, it is necessary to have detailed dictionaries which the algorithm can look through to link the form back to its lemma.[4]
Also, stemming has different algorithms and stemmed word depends on the rules the algorithm follows. For example, given the word 'legacies', the stemmed word can be 'legac' or 'legaci' based on the rules. But lemmatization always returns the base form found in dictionary 'legacy'.

9. For the search engine application, which is better? Give a proper justification to your answer. This is a good reference on stemming and lemmatization.
Ans. The aim of a search engine is to retrieve the most relevant information within limited time. Suppose

the input words are 'machine learning': A stemmer on 'machine' would return 'mach'/'mac'(depending on rules used in the stemming algorithm) and therefore retrieve information containing the core 'mach'. This would be several undesirable results, such as documents related to apple company products. Thus it has a high recall but low precision. A lemmatizer would return the lemma 'machine'.
Hence, leading to better results. Thus, lemmatization may result in more relevant information retrieval (higher precision) compared to stemming (but not always). Moreover performing morphological analysis involves more complexity compared to a rule-based approach like stemming which beats the purpose for a faster search engine. Thus, lemmatization is a good for small-scale applications (such as the one we build in the course). Otherwise, owing to faster retrieval, stemming is the better method to implement for bigger search engines.[5]

10. Perform stemming/lemmatization (as per your answer to the previous question) on theword-tokenized text.
Ans. Refer Code for implementation

11. Remove stopwords from the tokenized documents using a curated list of stopwords (for example, the NLTK stopwords list).
Ans. Refer Code for implementation

12. In the above question, the list of stopwords denotes top-down knowledge. Can you think of a bottom-up approach for stopword removal?
Ans. One way is to compute TF-IDF score for each word and eliminate those words which fall below some threshold value which we can experiment on.

Another most simplest way is using the knowledge that stopwords occur more often than regular words, we can find the frequency of occurences of each word in entire dataset and select the top few words and label them as stopwords.

REFERENCES:

[1] Punkt algorithm's Official paper:

https://dl.acm.org/doi/10.1162/coli.2006.32.4.485

[2] NLTK Punkt Sentence Tokenizer documentation:

https://www.nltk.org/_modules/nltk/tokenize/punkt.html

[3] NLTK Penn Treebank Tokenizer documentation:

https://www.nltk.org/_modules/nltk/tokenize/treebank.html

[4] Stemming and Lemmatization reference 1:

https://blog.bitext.com/what-is-the-difference-between-stemming-and-lemmatization/

[5] Stemming and Lemmatization reference 2:

https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html