CS6730 : Natural Language Processing
Assignment #1

E Naveen **(ME16B077)** Pawan Prasad **(ME16B179)**

1. What is the simplest and obvious top-down approach to sentence segmentation for English texts?

Ans: A basic top-down approach is to search the text find where any punctuation such as **{'.', ';', '!', '?'}** occur and store the corresponding indices. These indices are where we split the text to obtain the segments.

2. Does the top-down approach (your answer to the above question) always do correct sentence segmentation? If Yes, justify. If No, substantiate it with a counter example.

Ans: No, it does not: There exists abbreviations in english and numbers with decimal places which also contain full-stops/period which get incorrectly segmented into different component sentences.
Example Text: 'I completed my B.Tech degree with a score of 95.6%. I am happy.';
Top-Down Segmentation: ['I completed my B.', 'Tech degree with a score of 95.','6%.', 'I am happy.'];
Actual/Ideal Segmentation: ['I completed my B.Tech degree with a score of 95.6%.', 'I am happy.']
A simple way to tackle this problem is to look for *punctuations followed by blankspaces*. This avoids splitting the text into segments when encountering most of the typical abbreviations present in english text which would otherwise lead to false segmentation.

3. Python NLTK is one of the most commonly used packages for Natural Language Processing. What does the Punkt Sentence Tokenizer in NLTK do differently from the simple top-down approach?

Ans: Punkt is a data-driven unsupervised approach to sentence boundary detection [1]. Punkt achieves this by a two-fold process (type-based and token-based classifier) using 3 criteria for detecting abbreviations including 1) defining them as a collocation between truncated word and period, 2) they are usually short, 3) sometimes contain internal periods.

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 method is a perfect example of a bottom-up implementation of sentence tokenization guided by top-down suppositions about the nature of abbreviations (and initials/ordinal numbers). Since the Punkt Sentence Tokenizer is already trained on a large text-database of corresponding language, all we need to do is create an instance of it and use it on our text document [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

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) For specific cases, at the sentence boundaries, where there is no blank space following a punctuation (E.g: 'My name is Sherlock.How are you?') which is by itself an error the first method segments the text properly (E.g: ['My name is Sherlock.', 'How are you?']) whereas the second method

fails to do so (E.g: ['My name is Sherlock.How are you?']).
(b) For all cases where there are abbreviations or numbers with decimal places in the text (E.g: 'The temperature today is 50.7 degrees. It is hot.') the first method falsely segments (E.g: ['The temperature today is 50.', '7 degrees.', 'It is hot.']) whereas the second method proves to segment correctly (E.g: ['The temperature today is 50.7 degrees.', 'It is hot.']).

5. What is the simplest top-down approach to word tokenization for English texts?

Ans: We can store indices where a blankspace occurs. These indices are where we split the text to obtain words. *Note:* This approach is top-down because we already know beforehand that in English, words are separated by spaces and therefore utilize this knowledge accordingly.

6. Study about NLTK's Penn Treebank tokenizer. What type of knowledge does it use - Top-down or Bottom-up?

Ans: This tokenizer uses top-down knowledge about regular expressions occurring in text. These include performing the following tasks [3]:
- split standard contractions, e.g. "don't" to "do n't" and "they'll" to "they 'll".
- treat most punctuation characters as separate tokens.
- split off commas and single quotes, when followed by whitespace.
- separate periods that appear at the end of line.

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

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
(b) Since we are only looking at blank spaces as word boundaries in the first method, scenarios where a word is followed by a punctuation such as '?', '!', etc. the punctuation becomes part of the tokenized word whereas in the second method they are correctly identified as separate tokens.
Examples: The text 'What is your name?' tokenized as ['What', 'is', 'your', 'name?'] instead of ['What', 'is', 'your', 'name', '?'].

8. What is the difference between stemming and lemmatization?

Ans: Stemming refers to a rule-based heuristic process resulting in deletion of suffixes and returning the core of a word (which may or may not have a dictionary meaning) [4]. For example, given the word 'operating' as in the query 'operating system', a stemmer would return 'oper'.

Lemmatization involves a detailed morphological analysis of the given word using vocabulary and returns the base form of the word in the dictionary called *lemma* [4]. Taking the above mentioned example query 'operating system' and performing lemmatization on 'operating' will result in 'operate', a base word having meaning in dictionary rather than a meaningless crude form like 'oper'.

Results of stemming are dependent on the set of rules the algorithm follows. For example, given the word 'delicacies', the output can be 'delicac' or 'delicaci' depending on the rules, whereas, a lemmatizer will always return the base word from the dictionary which is 'delicacy'.

9. For the search engine application, which is better? Give a proper justification to your answer.

Ans: The aim of a search engine is to retrieve the most relevant information within limited time. Consider the query 'operating system': A stemmer on 'operating' would return 'oper' and therefore

retrieve information containing the core 'oper'. This can include several undesirable results, such as, documents related to 'operations research' , 'operatives' etc. Thus it has a high recall but low precision.

A lemmatizer would return the lemma 'operate'. Still, we do not always get results relevant to 'operating system' alone because particular inflectional forms are used in particular collocations: a sentence with the words *operate* and *system* is not a good match for the query 'operating system'. Getting better value from term normalization depends more on pragmatic issues of word use than on formal issues of linguistic morphology [4].

Thus, lemmatization may result in more relevant information retrieval (higher precision) compared to stemming but not always, as explained above. 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 fit 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.

10. Perform stemming/lemmatization (as per your answer to the previous question) on the word-tokenized text.

Refer Code for implementation

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

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 methodology would be to look at the data and for each word, keep count of the number of documents in which it occurs atleast once. Rank the words based on this count from words that appear in most number of documents to words that appear in least number of documents. Based on a threshold number of words we can take the first few words as stopwords.

Another simple methodology would be to keep count of the occurences for each word in the entire dataset and take the first few words as stopwords after ranking them from highest number of occurences to the least.

REFERENCES:

[**1**] Official paper for Punkt algorithm: https://dl.acm.org/doi/10.1162/coli.2006.32.4.485
[**2**] NLTK Punkt documentation: https://www.nltk.org/_modules/nltk/tokenize/punkt.html
[**3**] NLTK Treebank documentation: https://www.nltk.org/_modules/nltk/tokenize/treebank.html
[**4**] https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html