Assignment 1

Team 10

Akshat Meena (CS19B052) Siddharth Singh (CS19B072)

1. The simplest and obvious top-down approach to sentence segmentation is to split the sentences on every punctuation mark such as ‘.’, ‘?’, and ‘!’.
2. The simplest top-down approach to sentence segmentation stated above doesn’t always do correct segmentation. One counter-example is the presence of abbreviations in the sentence. The given approach will segment on the periods of abbreviations (e.g., Dr., Mr., etc).
3. The Punkt Sentence Tokenizer uses a bottom-up approach for doing the task of sentence segmentation. It uses an unsupervised algorithm to build a statistical model of the sentence structure of the language. It must be trained on a large collection of plaintext in the target language before it can be used. The NLTK data package includes a pre-trained Punkt tokenizer for English.
4. Top-down method v/s Pre-trained Punkt Tokenizer
    a. The first method performs better when the new sentence after a period starts without space between the period and the first word of the sentence.

       Eg. “He is Bob.He is 5 years old.”

        - First approach output: ['He is Bob.', 'He is 5 years old.']
        - Second approach output: ['He is Bob.He is 5 years old.']

    b. The second method performs better in presence of abbreviations and decimals.

       Eg. "Average weight of men from U.S.A is 197.9 pounds.”

        - First approach output: ['Average weight of men from U.', 'S.', 'A is 197.', '9 pounds.']
        - Second approach output: ['Average weight of men from U.S.A is 197.9 pounds.']

5. The simplest top-down approach to word tokenization for English texts is to split whenever you encounter a space. But we have also added a few additional functionalities like splitting on punctuations and math operators, along with some handling of abbreviations and contractions.
6. NLTK’s Penn Treebank tokenizer uses top-down knowledge. It uses regular expressions to tokenize text as in Penn Treebank. It constructs several regular expressions and matches their occurrences in the given text to handle the different cases (For e.g., splitting standard contractions, don't → do n't and they'll → they 'll, separate periods that appear at the end of line, etc.).
7. Top-down method v/s Penn Treebank Tokenizer
    a. The first approach performs better when math operators are used.

       Eg. "Bob insisted that 2+2=5.”

        - First approach output: ['Bob', 'insisted', 'that', '2', '+', '2', '=', '5', '.']
        - Second approach output: ['Bob', 'insisted', 'that', '2+2=5', '.']

    b. The second approach performs better in the case of currency symbols.

       Eg. "I have $200.”

        - First approach output: ['I', 'have', '$200', '.']
        - Second approach output: ['I', 'have', '$', '200', '.']
8. **Stemming** is a crude heuristic process that strips off the ends of words to obtain the root form of the word. It is a faster approach but has less accuracy. It can also lead to non-words(words that don’t exist in the dictionary). For e.g., “dance” is stemmed down to “danc” which is a non-existent word.

   Lemmatization, on the other hand, uses a more sophisticated approach taking into account the vocabulary and morphological analysis of words and aims to return the base or dictionary form of a word, which is known as the *lemma.* This is a slower process compared to stemming but produces accurate results.
9. According to us, stemming will be a better technique than lemmatization simply because a search engine has to process a huge amount of files/documents for executing a search query and we need a technique that is fast. We might tolerate a little bit of inaccuracy but more time taken for search query execution will drastically reduce the user experience. Also, stemming may provide more results as it will provide less similar results too.
10. We used two different stemmers from the NLTK package, namely:

a. Porter Stemmer - This is one of the most common and effective stemming algorithms.

b. Snowball - It offers a slight improvement over the original Porter stemmer, both in logic and speed.

Observation & Conclusion: From the given dataset, it was observed that the porter stemmer removed ‘s’ at the end of some words such as was→wa, has→ha, viscous→viscou, this→thi. On the other hand, the snowball stemmer didn’t remove them and hence we have employed the snowball stemmer in our code.

11. Done
12. We can use the concept of inverse document frequency (IDF) as a bottom-up approach for stopword removal.

    Suppose there are “N” number of documents in the collection out of which “n” contain the type “t”. Then, we define

$$IDF(t) = log_2(N/n)$$

    We learn a suitable threshold by training on some corpus. Then, if a word “t” has IDF lesser than the learned threshold, we say it is a stopword.

References

- https://www.nltk.org/_modules/nltk/tokenize/punkt.html
- https://www.nltk.org/_modules/nltk/tokenize/treebank.html
- https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html
- https://towardsdatascience.com/stemming-vs-lemmatization-2daddabcb221