CS6370 Natural Language Processing
Assignment 1

Team number : 41
Team members:
Vardhman Anand uikey CS21M069
Abhinav Anurag CS19M007
Ramakrishnan NA18B030

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

The most simplest and obvious top-down approach to sentence segmentation in English is to use sentence ending punctuations like “.”, “?”, “!” or other end-mark punctuation
Thus any sequence of words found between these these end-mark punctuations can be treated as a separate sentence.

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.

No, the top-down approach does not always produce accurate sentence segmentation.
One example could be sentences with “.” to represent an abbreviation of a word
For example: “Dr. Ram is a cardiologist” is segmented as [“Dr”, “Ram is a cardiologist”]
And the above sentence segmentation is not ideal as the context of the sentence is lost and the split is meaningless.

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? You can read about the tokenizer here.

Punkt sentence tokenizer is a top-down approach. Punkt sentence tokenizer uses an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences; and then uses that model to find sentence boundaries.

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
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)

a.) Top-down method:

- We split the input text by delimiters ".!?"
- The regex module splits the document into separate sentences
- The whitspaces are removed and the sentence list is returned

b.) pre-trained Punkt Tokenizer for English:

- We import the pre-trained punktsentencetokenizer from nltk.tokenize
- The document is segmented into sentences using punktsentencetokenizer
- The list of sentences is returned.

a.) example where the naive method performs better than punktsentencetokenizer:

- The naive method is faster and is better for implementation if the document had no abbreviation.
- The naive method is better when the grammar or spacing is not exactly followed in the sentences.
- For example, the sentence "Good morning.We are having a meeting at 12:00.Please call"
- Naive approach : ['Good morning', 'We are having a meeting at 12:00', 'Please call']
- Punktsentencetokenizer : ['Good morning.We are having a meeting at 12:00.Please call']

b.) example where the punktsentencetokenizer performs better than naive method:

- Punkt tokenizer is better at handling abbreviation in the middle of the sentence
- For example: "Dr. Ram is a cardiologist"
- Naive method : ['Dr', ' Ram is a cardiologist']
- Punktsentencetokenizer : ['Dr. Ram is a cardiologist']

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

The beginning and end of words are usually indicated in the English language script by spaces or word punctuation marks. The comma (,), hyphen (-), and front slash (/) are examples of word punctuation marks. As a result, the most straightforward top-down method to word tokenization would be to split words based on spaces and punctuation marks. As a result, a word is the portion of text that is between the immediate spaces or word punctuations.

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

Top-down knowledge is used to segment words in NLTK's Penn Treebank Tokenizer. It contains a collection of expressions and tokenization solutions that it utilizes to tokenize words, which overcomes some of the drawbacks of the straightforward technique of segmentation based on spaces and word punctuations.
Following are the steps taken by this tokenizer:

- Split ordinary contractions, such as don't to "do" and "n't" , they'll to "they" and "'ll"
- Most punctuation characters should be treated as individual tokens.
- When followed by whitespace, break off commas and single quotes.
- Periods at the end of each line must be separated.

7. Perform word tokenization of the sentence-segmented documents using

(a) The simple method stated above

- Here, we do punctuation-based tokenization.
- We begin by looking for an input.
- After that, the provided text is divided based on the punctuation marks selected.
- The result is returned after the edge cases (black words, symbols) are deleted.

(b) Penn Treebank Tokenizer:

- The input text is passed onto the TreebankWordTokenizer function which returns a lsit of tokenised words

a.) example where the first method performs better than the second one:

- The naive method is faster and performs better when the tokens in the middle of the sentence has "-" which
- For example : ["the pre-covid times"]
- Naive approach : [['the', 'pre', 'covid', 'times']]
- PennTreebankTokenizer : [['the', 'pre-covid', 'times']]
- Thus naive method is better because "pre-covid" is less useful than "pre" and "covid" tokens

b.) example where the second method performs better than the first one:

- The PennTreebankTokenizer is better at handling words with contractions
- For example : "I don't drink"
- Naive approach : [['I', 'don', 't', 'drink']]
- PennTreebankTokenizer : [['I', 'do', "n't", 'drink']]
- The PennTreebankTokenizer is better because the information regarding the negation of the token "don't" is preserved

8.What is the difference between stemming and lemmatization?

Stemming:

- Stemming is the process of producing morphological variants of a root/base word.
- Stemming programs are commonly referred to as stemming algorithms or stemmers.Often when searching text for a certain keyword, it helps if the search returns variations of the word.
- For instance, searching for “boat” might also return “boats” and “boating”. Here, “boat” would be the stem for [boat, boater, boating, boats].
- Stemming is a somewhat crude method for cataloging related words; it essentially chops off letters from the end until the stem is reached. This works fairly well in most cases, but unfortunately English has many exceptions where a more sophisticated process is required.

Lemmatization:

- Lemmatization looks beyond word reduction and considers a language’s full vocabulary to apply a morphological analysis to words. The lemma of ‘was’ is ‘be’ and the lemma of ‘mice’ is ‘mouse’.
- Lemmatization is typically seen as much more informative than simple stemming.
- Lemmatization looks at surrounding text to determine a given word’s part of speech, it does not categorize phrases.

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

- One thing to note about lemmatization is that it is harder to create a lemmatizer in a new language than it is a stemming algorithm because we require a lot more knowledge about structure of a language in lemmatizers.
- Stemming and Lemmatization both generate the foundation sort of the inflected words and therefore the only difference is that stem may not be an actual word whereas lemma is an actual language word.
- Stemming follows an algorithm with steps to perform on the words which makes it faster. Whereas, in lemmatization, you used a corpus also to supply lemma which makes it slower than stemming. You furthermore might had to define a parts-of-speech to get the proper lemma.
- The above points show that if speed is concentrated then stemming should be used since lemmatizers scan a corpus which consumes time and processing. It depends on the problem you’re working on that decides if stemmers should be used or lemmatizers.
- As a conclusion, we can say developing a stemmer is far simpler than building a lemmatizer. In the latter, deep linguistics knowledge is required to create the dictionaries that allow the algorithm to look for the proper form of the word. Once this is done, the noise will be reduced and the results provided on the information retrieval process will be more accurate.

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

- Stemming:

form	suffix	stem
studies	-es	studi
studying	-ing	study
cats	-s	cat
caresses	-es	caress

- Lemmatization:

I saw 18 mice today

I	PRON	I
saw	VERB	see
eighteen	NUM	eighteen
mice	NOUN	mouse
today	NOUN	today
!	PUNCT	!

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

Steps to remove stop-words:

- A list of pre-defined stopwords is loaded from nltk library
- If the lemmatized set of words has any intersection with the stopwords, we remove those words from the list

12. In the above question, the list of stopwords denotes top-down knowledge. Can you think of a bottom-up approach for stopword removal?

The stopwords are those words which occur frequently across documents and have very less discriminating power.
We can identify those words using inverse document frequency
Every word above a certain threshold of IDF value can be considered as stopword

$$IDF = \log(n/N)$$

Where n is the number of documents in which word w1 occurs
N is the total number of documents

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://www.geeksforgeeks.org/removing-stop-words-nltk-python/