CS6370 (NLP): Assignment 1

Manikandan Sritharan (EE19B038) & Snehan J (EE19B027)

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

The simplest approach would be to segment the text based on standard english language delimiters like ('.', '!', '?'). In our code, any sequence of text between the delimiters will be categorized as a sentence.

2 Does the top-down approach (the above answer) always do correct sentence segmentation?

No, the above approach will not always result in correct segmentation of sentences.
Consider the following text, "Hello, my name is Monkey D. Luffy. Nice to meet you."

- The above text will be segmented as ["Hello, my name is Monkey D.", "Luffy.", "Nice to meet you."]
- The correct segmentation should have been ["Hello, my name is Monkey D. Luffy.", "Nice to meet you."]

3 What does the Punkt Sentence Tokenizer in NLTK do differently from the simple top-down approach?

Python NLTK is one of the most commonly used packages for Natural Language Processing. As we saw earlier, top-down approach fails in places where abbreviations are present. The Punkt Sentence Tokenizer in NTK is trained with an unsupervised algorithm that identifies possible abbreviations, collocations in the text.

The Punkt Tokenizer uses a two-stage approach to detect sentence boundaries:

- Type based classification stage
- Token based classification stage

4 Perform sentence segmentation on the documents in the Cranfield dataset using:

4.1 Naive outperforms Punkt:

Being a model trained on text with proper english semantics and grammar, the Punkt Segmenter underperforms on real world data with improper structure.

Consider the following example, (with no spaces between sentences)

"Hi, fellow NLP enthusiast.Hope you are doing well."

The Punkt model's output would be ["Hi, fellow NLP enthusiast.Hope you are doing well."], whereas our naive model's output would be ["Hi, fellow NLP enthusiast.", "Hope you are doing well."].

4.2 Punkt outperforms Naive:

The Punkt model is very adept at handling abbreviations and Names.

Consider the example mentioned in Q2, “Hello, my name is Monkey D. Luffy. Nice to meet you.” The Punkt model’s output would be [“Hello, my name is Monkey D. Luffy.”, “Nice to meet you.”], whereas our naive model’s output would be [“Hello, my name is Monkey D.”, “Luffy.”, “Nice to meet you.”].

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

The simplest top-down approach for word tokenization is spitting at spaces, commas, exclamatory marks, slashes, hyphens and brackets.

6 What type of knowledge does NLTK’s Penn Treebank tokenizer use: Top-down or Bottom-up?

The Penn Treebank tokenizer is a top-down approach for word tokenization which uses regular expressions as defined in Penn Treebank to tokenize text. It uses a direct set of rules for what constitutes a word. This tokenizer assumes that text has already been segmented into sentences.

The Penn Treebank tokenizer performs some of these steps:

- Split standard contractions. Ex: ”Don’t” to ”Do” and ”n’t”
- Treat most punctuation characters as separate tokens

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

7.1 Naive outperforms Penn Treebank:

The Naive algorithm will ignore most (if not all) special characters unlike Penn Treebank. Contrary to expectations, this is better in the case of Information Retrieval system.

Consider the following word, ”old-fashioned”. It is better to split it as ”old” and ”fashioned”. If not, processing on hyphenated tokens can get quite muddy while performing lemmatization.

7.2 Penn Treebank outperforms Naive:

As mentioned in Q5, Penn Treebank handles standard contractions better. Ex: ”Don’t” to ”Do” and ”n’t”, while the naive model splits it as ”Don” and ”t”.

8 What is the difference between stemming and lemmatization?

Both stemming and lemmatization include the process of reducing words into a base form. Stemming is a heuristic way of this process where it tries to remove derivational affixes, whereas lemmatization uses vocabulary and morphological knowledge to remove affixes and also to convert them to a base form.

The major difference is that stemmers use language-specifc rules, but they require lesser knowledge than a lemmatizer, which needs a complete understanding of the vocabulary to correctly lemmatize words. After stemming, the result words may not be meaningful words but lemmatization will result in a correct dictionary word.

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

For an Information Retrieval System, stemming would be the better choice as stemming improves the **recall** of the result, while lemmatization improves the **precision** of the result.

While stemming might result in meaningless words (by truncating the ends), they are better when matching queries and documents as this will result in more relevant matches.

Out of the many choices of stemmers available, we have decided to use the SnowballStemmer() from nltk due to it's superior empirical performance with the respect to the other stemmers.

10 Perform stemming/lemmatization on the word-tokenized text.

Implemented in code.

11 Remove stopwords from the tokenized documents using a curated list of stopwords

Implemented in code.

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 most intuitive bottom-up approach for determining a stop list is to sort all the words in the corpus by collection frequency (the total number of times each word appears in the document collection), and then to take the most frequent words as stop words.

A more involved way of doing this is with the help of **Inverse document frequency (IDF)**. IDF gives us measure of whether a term is common or rare in given document corpus. We can sort the IDF values and remove the most frequently occuring words (low IDF).

$$\text{IDF (x)} = \log(N/(n+1))$$

N = total number of documents, n is the number of documents where term x appears.

13 References

- https://www.nltk.org/api/nltk.tokenize.html
- https://docs.python.org/3/library/re.html
- https://nlp.standford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html
- https://towardsdatascience.com/stemming-lemmatization-what-ba782b7c0bd8