CS6370: Natural Language Processing

Team details:

Team number: **16**
Team members:

1) N Kausik (CS21M037)
2) Karthikeyan S (CS21M028)
3) Kavya Rengaraj (CS21Z018)

Assignment 1

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

Answer:

Most written languages including English have punctuation marks at sentence boundaries**.** Thus the most simple and obvious top-down approach would be making use of the punctuations to determine the sentence boundaries.

"In most NLP applications, the only sentence boundary punctuation marks considered are the **period, question mark, and exclamation point**, and the definition of what constitutes a sentence is limited to the text-sentence, as defined by (Nunberg, 1990)".

So the approach we are going to follow is to parse the sentences from the start and segment the sentences as and when we encounter either one of the punctuation marks (**period(.), question mark(?), and exclamation point(!)**).

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.

Answer:

No. In English, even though the sentences are bound by punctuation marks, the rules for punctuation are not always coherently defined and thus we need to disambiguate all instances of punctuation that may delimit sentences.

For instance, the approach we used fails to segment the sentences properly when the text contains:

- Abbreviations like **I.I.T, Ph.D, Prof., Dr., Er.**
- Latin abbreviations like **etc., i.e., e.g., c., and et al.**
- Names with initials containing periods like **H.P. Lovecraft**
- Decimal numbers like 1.2, 3.5

Example:
Input text: " I am pursuing my M.Tech in CSE at I.I.T Madras where there are multiple professors who have completed their Ph.D's. I have enrolled for a subject called NLP which is offered by Prof. Dr. Sutanu Chakraborti. I am expected to attend 4.0 hrs of NLP class every week."

Output:

```
[' I am pursuing my M',
 'Tech in CSE at I',
'I',
'T Madras where there are multiple professors who have
completed',
 'their Ph',
'D's',
'I have enrolled for a subject called NLP which is offered by
Prof',
'Dr',
'Sutanu Chakraborti',
'I am expected to attend 4',
'0 hrs of NLP class every week']
```

Actual output should have been:

```
[' I am pursuing my M.Tech in CSE at I.I.T Madras where there
are multiple professors who have completed their Ph.D's',
 'I have enrolled for a subject called NLP which is offered by
Prof. Dr. Sutanu Chakraborti',
 'I am expected to attend 4.0 hrs of NLP class every week']
```

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.

"*This tokenizer divides a text into a list of sentences by using an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences. It must be trained on a large collection of plaintext in the target language before it can be used.*" -- cited from NLTK documentation.

In the naive approach we simply segment the sentences based on punctuations but punkt does not follow that instinctive approach but is trained to differentiate punctuations which are part of abbreviations, decimal numbers, etc. Thus punkt is more robust in the task of segmentation when compared to naive models. We can add custom abbreviations to punkt by making use of **punkt._params.abbrev_types.add()** method.

We can refer to question2 as the typical top-down approach did not perform as expected on the text we gave, but the punkt model will yield better results as expected.

Output of Segmentation with punkt:

```
[' I am pursuing my M.Tech in CSE at I.I.T Madras where there
are multiple professors who have completed their Ph.D's',
 'I have enrolled for a subject called NLP which is offered by
Prof. Dr. Sutanu Chakraborti',
 'I am expected to attend 4.0 hrs of NLP class every week']
```

4. Perform sentence segmentation on the documents in the Cranfield dataset using:
(a) The top-down method stated above
Refer sentenceSegmentation.py in zip file.

(b) The pre-trained Punkt Tokenizer for English
Refer sentenceSegmentation.py in zip file.

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)

Answer:

a) Naive approach performs better than Punkt method in terms of execution time, at cases where there are no abbreviations involved or in cases where there is no ambiguity in punctuations involved.

Because in the Punkt model in terms of time taken for execution, it takes additional time to import or download external packages like NLTK and also it has to take care of characters before and after punctuation to decide whether the punctuation mark segments a sentence or not. Whereas the Naive method just uses punctuation marks as delimiters and performs segmentation very fast.
Example: I ate an apple in the morning.

b) Punkt approach performs better than Naive method in terms of accuracy as punkt model uses trained knowledge about the context and differentiates between punctuations that delimits sentences and abbreviations, decimal numbers, etc.

Example:
Text 1: I.I.T is the number one engineering institute in India

Naive approach: `['I', 'I', 'T is the number one engineering institute in India']`

Punkt Model: `['I.I.T is the number one engineering institute in India']`

5. What is the simplest top-down approach to word tokenization for English texts?
Answer:
The simplest top-down approach is to use split() function that separates words in a sentence using a delimiter(mostly whitespace)

6. Study about NLTK's Penn Treebank tokenizer here. What type of knowledge does it use
- Top-down or Bottom-up?
Answer:
Penn treebank tokenizer uses Regex patterns for tokenizing texts.
It has the following features.

- Separates periods that appear at the end of the line
- Split standard contractions, e.g. ``don't`` -> ``do n't`` and ``they'll`` -> ``they 'll``
    - treat most punctuation characters as separate tokens
    - split off commas and single quotes, when followed by whitespace

The penn treebank word tokenizer assumes that the input text is already segmented into sentences using the sentence tokenizer.
The genre of training text included in Penn Treebank is from wall street journal news stories and some spoken conversations
Penn treebank tokenizer uses a top-down approach as it uses regex patterns to tokenize the input text.

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

State a possible scenario along with an example where:
(a) the first method performs better than the second one (if any)
Answer:
When both (1) and (2) are true

1. No more patterns other than white-space based splitting are required in the input data i.e. works well for simple cases.
2. Speed is a concern

(b) the second method performs better than the first one (if any)
Answer:
when the input text contains more punctuations.
Example :
Say we want to find out what Ria and Dia has in common(fruits)
input_1 = 'Ria likes to eat apples and bananas.'
input_2 = 'Dia likes to eat apples,bananas and oranges.'
print(input_1.split())
print(input_2.split())

Using whitespace based splitting gives the following results
['Ria', 'likes', 'to', 'eat', 'apples', 'and', 'bananas.']
['Dia', 'likes', 'to', 'eat', **'apples,bananas'**, 'and', 'oranges.']
where apples and apples,bananas because it doesn't treat punctuations.

Comparatively, **penn treebank tokenizer does a good work here**
from nltk.tokenize import TreebankWordTokenizer
tokenizer = TreebankWordTokenizer()

```
input_1 = 'Ria likes to eat apples and bananas.'
input_2 = 'Dia likes to eat apples,bananas and oranges.'
print(tokenizer.tokenize(input_1))
print(tokenizer.tokenize(input_2))
```

['Ria', 'likes', 'to', 'eat', 'apples', 'and', 'bananas', '.']
['Dia', 'likes', 'to', 'eat', **'apples', ',', 'bananas',** 'and', 'oranges', '.']

So use penn treebank tokenizer when more patterns/rules are needed for splitting the input data and when punctuations are included.

White space tokenization - Split()	PennTreebank
Uses delimiter to split the text in to tokens	Uses predefined regex patterns
Doesn't treat punctuation as separate tokens	Treats most punctuation characters as separate tokens Can convert brackets in to PTB symbols for further analysis
Time taken for tokenizing the following sentence input = "Don't forget to bring your passport, your driver's license, and your proof of address." 0.0009958744049072266	Time taken for parsing the following sentence input = "Don't forget to bring your passport, your driver's license, and your proof of address." 0.0054628849029541016

8. What is the difference between stemming and lemmatization?

Answer:

Stemming is a process by which we produce morphological variants of a root word to aid in better searching. Eg. If we take a root word as "run", we can relate words like "running", "runner", etc to the root word and hence can get better search results by reducing those words into a common root word.

Lemmatization is a more appropriate method for morphological analysis since it doesn't simply check if a word is reducible to another word but also considers the vocabulary of the language. It uses the part of speech tags of a word like whether it is a verb, noun, etc. for better analysis.

Eg. The lemma of “saw” can either be “saw” itself or “see” depending on whether the word is used as a noun or verb.

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.
Answer:

For Search Engines, lemmatization will give better and more accurate results than stemming. This is due to the fact that in many cases, the stemmed morphological root/base of a word may not be meaningfully related to the original word. Since lemmatization takes care of this by reducing the word while considering the vocabulary and not just the letters in the word, it works better.

Eg. For the input query word “caring”, stemming produces the reduced word “car” whereas lemmatization produces the reduced word “care”.

Here, using “car” would produce wrong results whereas “care” would produce proper search results.

10. Perform stemming/lemmatization (as per your answer to the previous question) on the word-tokenized text.
Refer inflectionReduction.py in the zip file.

11. Remove stopwords from the tokenized documents using a curated list of stopwords (for example, the NLTK stopwords list).
Refer stopwordRemoval.py in the zip file.

12. In the above question, the list of stopwords denotes top-down knowledge. Can you think of a bottom-up approach for stopword removal?
Answer:
The nltk stopwords list is pre-existing knowledge and hence that method is a form of top-down approach for stop-word removal.
For a bottom-up approach to stop-word removal, we cannot use any such pre-existing knowledge. Hence the stop-words must be identified from the input dataset itself. Since stop-words are words which are not considered important and are filtered out, we can consider words with **less information content** in the dataset as stop-words.
Hence we can use any measure of information of words in a dataset like TF-IDF or Entropy. Using this measure we can rank the words in the dataset in descending order of information

brought to the dataset by that word. Then we can consider the least ‘N’ words in the ranking as stop-words and remove these words from the input sentences to perform stop-word removal. Since we do not have any pre-existing knowledge about stop-words, this method is a bottom-up approach.