CS6370 : Natural Language Processing Assignment #1

Roll No: CS14B017 Full Name: Manohar Mulle

Roll No: CS14B043 Full Name: Harshal Gawai

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

   Solution: An obvious top-down approach to sentence segmentation for English texts is to split the text based on some specific punctuation marks ie.. periods(.) , exclamation marks(!) and question marks(?)

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.

   Solution: Top-down approach as done based on (.),(!) and (?) doesn't guarantee correct sentence segmentation always. eg. an abbreviation used in sentence could have full-stops after each character which will split each of those characters during sentence segmentation as separate sentence. eg. A.S.A.P - as soon as possible Also in case of titles and short forms eg. (Dr.), (Dept.) , (Prof.)

   The following is an example with informal English text:
   $ He said he ate a thousand (!) fishes. I don't believe him.

   Clearly, the output of our sentence segmentator here will be,
   $ He said he ate a thousand (!
   $) fishes.
   $ I don't believe him.

   While the correct output should be,
   $ He said he ate a thousand (!) fishes.
   $ I don't believe him.

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?

   Solution: Punkt Tokenizer, in addition to the the above specified simple top-down approach, tries to learn the abbreviations from a training data using an unsupervised algorithm. While the design of the algorithm is a top-down choice, the learning of the abbreviations from the training data is clearly a bottom-up one.

4. After perform sentence segmentation on the documents in the Cranfield dataset, state a possible scenario along with an example where:
(a) The top-down method stated above performs better than the second one
(b) The pre-trained Punkt Tokenizer for English performs better

Solution: (a) Consider,
$ He only goes by M. No one knows his real name.

The top-down method that we implemented split above text into
$ He only goes by M.
$ What is your name?
Which is correct considering M is a name of a person.

Whereas Punkt tokenizer doesn't split text. Because it saw M. as short form or an abbreviation., it just outputs,
$ He only goes by M. No one knows his real name.

(b) Now, consider,
$ That's Mr. Ramesh.
Since, the punkt tokenizer is trained on a large collection of plain-text before it's usage. It understands Mr. is an abbreviation and hence didn't split text during sentence-Segmentation and outputs,
$ That's Mr. Ramesh.

While our top-down method simply split text into two sentences at punctuation mark (.) and outputs,
$ That's Mr.
$ Ramesh.

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

Solution: Tokenization is a common task in Natural Language Processing (NLP). It is done after Sentence Segmentation. Tokenization is a way of separating a piece of text into smaller units called tokens. This tokens can be either words, characters, or subwords.
The simplest top-down approach to word tokenization would be to by splitting the input whenever a white space in encountered.

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

Solution: The Penn Treebank tokenizer uses regular expressions to tokenize text. Just like in compiler design it chooses LA lexical analyser on sentence. Its a rule based approach with fixed set of rules. Penn Treebank tokenizer approach is Top-down.

7. Perform word tokenization of the sentence-segmented documents using Naive approach and pennTreeBank. Then 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)

Solution:
(a) Consider a written article about a person Mr.Black. There will me many instances throughout this article where "Mr.Black" will come in sentences throughout article.
Penn Treebank tokenizer will split this string into two tokens(i.e. "Mr." and "Black") instead of [Mr.Black] as a single token. Here our first method, the naive approach where we make tokens on cue of whitespace will give single token. Which is what we want.

(b) Consider a sentence "It costs $1000 for an iphone."
A treebank Word Tokenizer will give this tokens: ['It', 'costs', '$1000', 'for', 'an', 'iphone', '.']
Here, $ symbol and its value are split in two separate tokens which is what we ideally want.
Whereas naive approach that was built (as above) will consider '$1000' as one single token. Here second method performs better than first one.

8. What is the difference between stemming and lemmatization?

Solution:
Stemming is the process of producing morphological variants of a root/base word. Stemming algorithms work by cutting off the end or the beginning of the word, taking into account a list of common prefixes and suffixes that can be found in an inflected word. **e.g.**

Form	Suffix	stem
studies	-es	studi
studying	-ing	study

Lemmatization looks beyond word reduction and considers a language's full vocabulary to apply a morphological analysis to words. Here it makes use of a detailed dictionaries to link the form back to its root/base and form lemma. **e.g.**

Form	Lemma
studies	study
studying	study

As we can see, stemming is a top down approach while lemmatization involves a lot of bottom up knowledge

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

Solution: 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. **e.g.** From above:
Studies $\xrightarrow{\text{Stemming}}$ studi
Studies $\xrightarrow{\text{lemmatization}}$ study
Lemmatization, uses a corpus to supply lemma which makes it slower than stemming. While doing query search on search engine application it is important that word doesn't change meaning while we find stem or root form of words.
For search engine application , lemmatization is preferred.

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

Solution: (This is implemented in code.)

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

Solution: (This is 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?

Solution: A possible Bottom-up approach for stopword removal can be:

1. Tokenize all documents.
2. Find tokens that are common in sufficient number of documents among all, if not all documents.

3. These tokens can be considered as our stopwords
4. In other words, tokens that have idf(inverse document frequency) values close to zero(threshold can be subjective but usually very close to zero) can be considered as stopwords