CS6370: Natural Language Processing Assignment - 1

Shania Mitra CH18B067
Shashank Patil CH18B022

March 15, 2021

---

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

   Splitting a discourse on punctuation marks such as '.' , '?' , '!' can separate sentences.

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, it may not always lead to correct result. Suppose we have a sentence such as "Good Morning Mr. Sharma! How are you today?", it would be broken as "Good Morning Mr.","Sharma!", "How are you today?" which is incorrect. Further, if multiple exclamation marks or fullstops are used just as "Hip Hip Hurrah!!!" are used, empty string between the exclamation marks are considered as sentences.. This method also cannot handle decimals in between numbers in a sentence. Further if a code snippet is a part of the sentence it may cause problems. For example. "In python != is used to symbolise not equal to. The symbol <> may also be used" would be broken as ["In python!", "= is used to symbolise not equal to", The symbol <> may also be used"][1]

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?**

   Punkt Sentence Tokenizer is pre-trained on a large english corpus using an unsupervised algorithm and thus also has bottom-up knowledge. This approach allows it to learn abbreviations, collocations and words that start sentences. It can handle:

   (a) Fullstops in between a sentence and decimals, that do not form sentence boundaries

   (b) Sentences that do not start with a capital letter

   (c) Capital letters appearing in between a sentence

   (d) Parentheses in a discourse [1]

4. **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)**

   Since the second approach involves learning from a corpus, it may fail when the training corpus is not representative enough. Secondly it may fail when it misinterprets periods that are supposed to be sentence boundaries as abbreviations or collocations. For example, "There have been many inventions since the advent of comp. Science has progressed immensely since then." Here, comp is an abbreviation for computers and

while the period signifies an abbreviation as well as a sentence boundary, it is misinterpreted to be once sentence where comp. Science will be considered an abbreviation for Computer Science. Thus, the ideal breakup should have been “There have been many inventions since the advent of comp. (computers)” and “Since has progressed immensely since then”. However, it is treated as one sentence.

The second method performs better when there are periods that should not form sentence boundaries, such as in cases of abbreviations. It also deals with paranethesis consistently, even if they appear after sentence boundaries, consisting of many sentences. It also aptly deals with decimals and abbreviations. For example, "Mr. Bean reaches home at 5 p.m. He then drinks 1.5 L milk" when passsed through Punkt tokenizer gives [’Mr. Bean reaches home at 5 p.m.’, ’He then drinks 1.5 L milk’] whereas naive tokenizer gives [’Mr’, ’ Bean reaches home at 5 p’, ’m’, ’ He then drinks 1’, ’5 L milk’] which is not desirable.

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

   The simplest approach would be to split the sentence across spaces and punctuations followed by spaces such as ’;’ and ’,’.

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

   NLTK’s Penn Treebank tokenizer uses top-down knowledge. It uses regular expressions to tokenize text as in Penn Treebank. It takes a list of standard contractions from Robert MacIntyre’s tokenizer such as “don’t” as “do”, “n’t” and “they’ll” into “they” and “’ll”. It handles punctuation characters as separate tokens splits commas and single quotes off from words, when they are followed by whitespace, splits off periods that occur at the end of the sentence.[2]

7. **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)**

   The first method is more beneficial in case of some contractions. For eg. Treebank tokenizer splits “can’t” as “ca” and “n’t” which does not make sense and may cause the sentence to lose meaning. Further, “I’d” is split as “I” and “’d”. In such cases it would be better to let these contractions remain as it is and perform spelling corrections at a later state.

   The second method performs better in most other cases. When the user types punctuations without a space the simple tokenizer would not be able to handle it while the Treebank tokenizer can. For eg. “I need pens, pencils,colours and an eraser” would result in [’I’, ’need’, ’pens’, ’pencils’, ’colours’, ’and’, ’an’, ’eraser’][2] while in the first method, “pencil,colours” would be treated as one word.

8. **What is the difference between stemming and lemmatization?**

   Stemming is the process of reducing inflection in words to their root forms such as mapping a group of words to the same stem even if the stem itself is not a valid word in the Language.Lemmatization, unlike Stemming, reduces the inflected words properly ensuring that the root word belongs to the language. In Lemmatization root word is called Lemma. A lemma (plural lemmas or lemmata) is the canonical form, dictionary form, or citation form of a set of words.[3, 4]

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

   Although Stemming is faster than Lemmatization, most modern day machines are capable of performing Lemmatization almost as fast as Stemming, due to which we use Lemmatization since it is known to deliver

more accurate results. [5] Consider the following example: The user searches with the word “celebrities”, as in the plural of celebrity, but the search engine ends up with a stem of “celebr” (Stemming). That search could end up with false positives from other words with the same stem as “celebrations”. This is not desirable. The only way to tackle this and get accurate results is to perform more advanced morphological analysis to find the ’lemma’ of the word (Lemmatization). In our code, we use the “WordNet Lemmatizer”[6]

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

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

    A large corpus can be taken and frequency of all words can be calculated. The most frequently occuring words can be thought of as stopwords since their occurence does not provide much information and is not discriminative.

    We could also use the TF-IDF score. Words which occur very frequently across many documents will be of little help in differentiating the relevant documents from the irrelevant ones. But ones which occur very frequently (high tf or term-frequency) in only few documents (high idf or inverse document frequency) as likely to be more important in distinguishing the good documents from the bad ones.

    The term-document matrix is computed with the respective TF*IDF scores in the matrix. We then find the final TF-IDF score for each term averaged across all the documents. We then set a threshold to generate the list of stopwords.

    Another approach could be Entropy which offers us an opportunity to determine stop words based on the amount of information that a word carries.

    We calculate the entropy value (H) for word w_j as:

    $H(w_j) = \sum_{i=1}^{n} P_{x,y}\ log(\frac{1}{P_{x,y}})$

    $P_{x,y} = \frac{frequency\ of\ word\ w_j\ in\ Document\ D_i}{total\ number\ of\ words\ in\ D_i}$

    where, $P_{x,y}$ is the probability of the word w_j

    The higher the entropy a word has, the lower the information value of such a word. Therefore, the words with high entropy (above a threshold) will be extracted as the stop words.[7]

References

[1] NLTK Punkt Tokenizer Source Code

[2] NLTK Penn TreeBank Tokenizer Source Code

[3] Stemming vs Lemmatization - Stanford

[4] Stemming vs Lemmatization - DataCamp

[5] Stemming vs Lemmatization for Search Engines

[6] NLTK WordNet Lemmatizer Source Code

[7] Auto-generated Approach of Stop Words Using Aggregated Analysis