CS6370: Natural Language Processing

Assignment 1

Shubham Biren Patel, ME19B170
Bagul Amit Sunil, MM19B023

---

1. The simplest top-down approach would be to take into account the most common sentence ending punctuation marks such as full stop, question mark and exclamation mark.

2. No. The approach mentioned above works incorrectly when a full stop is used for abbreviations like Mr. or C.N.N.

3. *Punkt Sentence Tokenizer* in NLTK is trained on abbreviation words, collocations, and words that start sentences and hence a model is built. It understands that Mr. does not end a sentence. Also that some sentences need not begin with capital letters.

4. Code Implementation: refer to `sentenceSegmentation.py`
   (*a*) The method mentioned above may work better than the *Punkt Sentence Tokenizer* in case the text has ill-formed English sentences. For example text 'valid any day after January 1. not valid on federal holidays, including February 14, or with other in-house events, specials, or happy hour.' has two sentences but *Punkt Tokenizer* does not recognize end of sentence after 'January 1.'. But above mentioned method does recognize end of sentence at full stop and give two sentences. same with example 'I was working on a project on 2^{nd} Feb of 2021. l was a good variable name that I used for length of strings.'
   (*b*) The *Punkt Sentence Tokenizer* works better than approach/method mentioned above in case text has abbreviations and question mark, exclamation mark that do not represent end of sentence. For example text, 'If I asked my friends questions like, "With whom are you seeing a movie?" or "About what are you talking?" I would get teased mercilessly. This way of communicating is, in a word, formal—which means it's out of place in casual and friendly conversations.' has "?" mark that is not end of sentence.

5. The simplest tokenization approach would be to segment the input based on the spaces and the most common punctuations like fullstops, question marks, exclamation marks, commas, colons, semicolons and quotation marks that occur between words.

6. NLTK's *Penn Treebank Tokenizer* uses top-down knowledge of the way punctuation marks separate words, begin or end sentences or simply occur in abbreviations. This is accomplished by using regular expressions to account for different cases. This correctly facilitates tokenization.

7. Code Implementation: refer to `tokenization.py`
   (*a*) The simple approach only uses the occurence of common punctuations like [" ", ".", "?", "!", ",", ";", ":", "", "'] to tokenize words from a given list of sentences. Whereas, *Penn Treebank Tokenizer* uses a bunch of regular expressions to account for all the cases to implement tokenization. It can also identify parentheses, standard contractions, etc. Thus, in the case of simple text having sentences without many contractions, parentheses, the simple approach may perform better than *Penn Treebank Tokenizer* in terms of time complexity.
   (*b*) *Penn Treebank Tokenizer* is better than the simple approach in case the text has decimal numbers like '6.58'. This will be tokenized as 6.58 by the former and ['6', '.', '58'] by the latter. While, english contractions like 'can't' will be tokenized as ['ca', 'n't'] by the former and ['can', '', 't'] by the latter.

8. Both stemming and lemmatization are text normalization techniques, used for reducing inflectional forms and derivationally related words to common base word form. Stemming chops off affixes from word to get the base word/stem. Stem may not be a meaningful word. In contrast, Lemmatization reduces words using full morphological analysis to identify the lemma, a meaningful word. So,

building a lemmatizer is harder than developing a stemmer. Stemming is used when the dataset is large but may give incorrect meanings and spellings. Lemmatization can overcome this but it can be computationally expensive as it needs to scan the entire corpus.

9. For the search engine application, lemmatization technique is better. If word 'celebrities' is searched, stemming results in 'celebr'. This gives false results for the same stem but a different word 'celebration'. This can be avoided using lemmatization resulting in lemma 'celebrity' instead of 'celebr' for searched word 'celebrities'.

10. Code Implementation: refer to `inflectionReduction.py`

11. Code Implementation: refer to `stopwordRemoval.py`

12. A bottom-up approach for stopword removal could involve a search through the corpus to find the top $\boldsymbol{k}$ most frequent words. Hence, based on the hyperparameter $\boldsymbol{k}$ of choice and the nature of the task at hand, required text reduction can be achieved.