The goal of the assignment is to build a search engine from scratch, which is an example of an information retrieval system. In the class, we have seen the various modules that serve as the building blocks of a search engine. We will be progressively building the same as the course progresses. The first part of this assignment is to build a basic text processing module that implements sentence segmentation, tokenization, stemming/lemmatization and stopword removal. The Cranfield dataset will be used for this purpose, which has been uploaded separately on Moodle.

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

ANS: Sentence segmentation refers to the process of breaking down a piece of written text into its constituent sentences. A top-down approach [1] to parsing a given piece of written text works on the assumption that we have some pre-existing idea about the domain to which the text belongs. Having this idea and by appropriately applying it, one then goes on to break down the piece of text until the desired smallest units (in this case sentences) are obtained.

For English texts, the most obvious top-down approach to sentence segmentation would be to make use of popular and widely used markers [2] indicating the end of sentences. For example, **‘.’** (period/full-stop), **‘?’** (question mark), **‘!’** (exclamation mark). This näive top-down approach could be used to segment sentences as follows:

Text: “*Peter lived in Queens****.*** *He was bit by a spider****?*** *What a story****!***”

Segmented Sentences: “*Peter lived in Queens****.***”, “*He was bit by a spider****?***”, “*What a story****!***”

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.**

ANS: No, the näive top-down approach discussed above in the previous question does not always produce correct sentence segmentation. English texts have a popular prevalence of periods as part of abbreviations. For instance, **e.g.** → the short form for example, **Dr.** → abbreviation for doctor, **St.** → short for Saint as a title conferred to a person or the name of a place like St. Petersburg, **A.M.** → as in ante meridiem (10 A.M.).

With such occurrences of the period in a given piece of text, the näive top-down approach would end up producing erroneous and meaningless segments. An example:

Text: “*Dr****.*** *Strange will arrive at St****.*** *Petersburg****.*** *Remember, tomorrow by 10 A****.****M****.***”

Segmented Sentences: “*Dr****.***”, “*Strange will arrive at St****.***”, “*Petersburg****.***”, “*Remember, tomorrow by 10 A****.***”, “*M****.***”

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.**

ANS: The Punkt Sentence Tokenizer uses a bottom-up approach for sentence segmentation. A bottom-up approach [1] to parsing a given piece of text works by first recognizing smaller portions of text, identifying their syntactic classes followed by combining results obtained for the smaller objects to identify bigger objects composed of those smaller objects until some kind of language entity (sentences in this case) is recognized.

The Punkt Sentence Tokenizer performs sentence segmentation by dividing a piece of text into a list of sentences by using an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences. [3] It must be trained on a large collection of plaintext in the target language before it can be used. [3] The large collection of paintext can either be from a NLTK provided corpus or user supplied raw text to produce a custom sentence tokenizer. [4] The specific technique used by the Punkt Sentence Tokenizer is called sentence boundary detection and it works by counting punctuation and tokens that commonly end a sentence, such as a period or newline, then using the resulting frequencies to decide what the sentence boundaries should actually look like. [4]

In the light of the above mentioned information, the sentence segmentation for the example discussed in Q2 using the Punkt Sentence Tokenizer would be as follows:

Text: “*Dr****.*** *Strange will arrive at St****.*** *Petersburg****.*** *Remember, tomorrow by 10 A****.****M****.***”

Segmented Sentences: “*Dr****.*** *Strange will arrive at St****.*** *Petersburg****.***”, “*Remember, tomorrow by 10 A****.****M****.***”

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

**(a) The top-down method stated above**

ANS: The Code has been submitted along with this report.

**(b) The pre-trained Punkt Tokenizer for English**

ANS: The Code has been submitted along with this report.

**State a possible scenario along with an example where:**

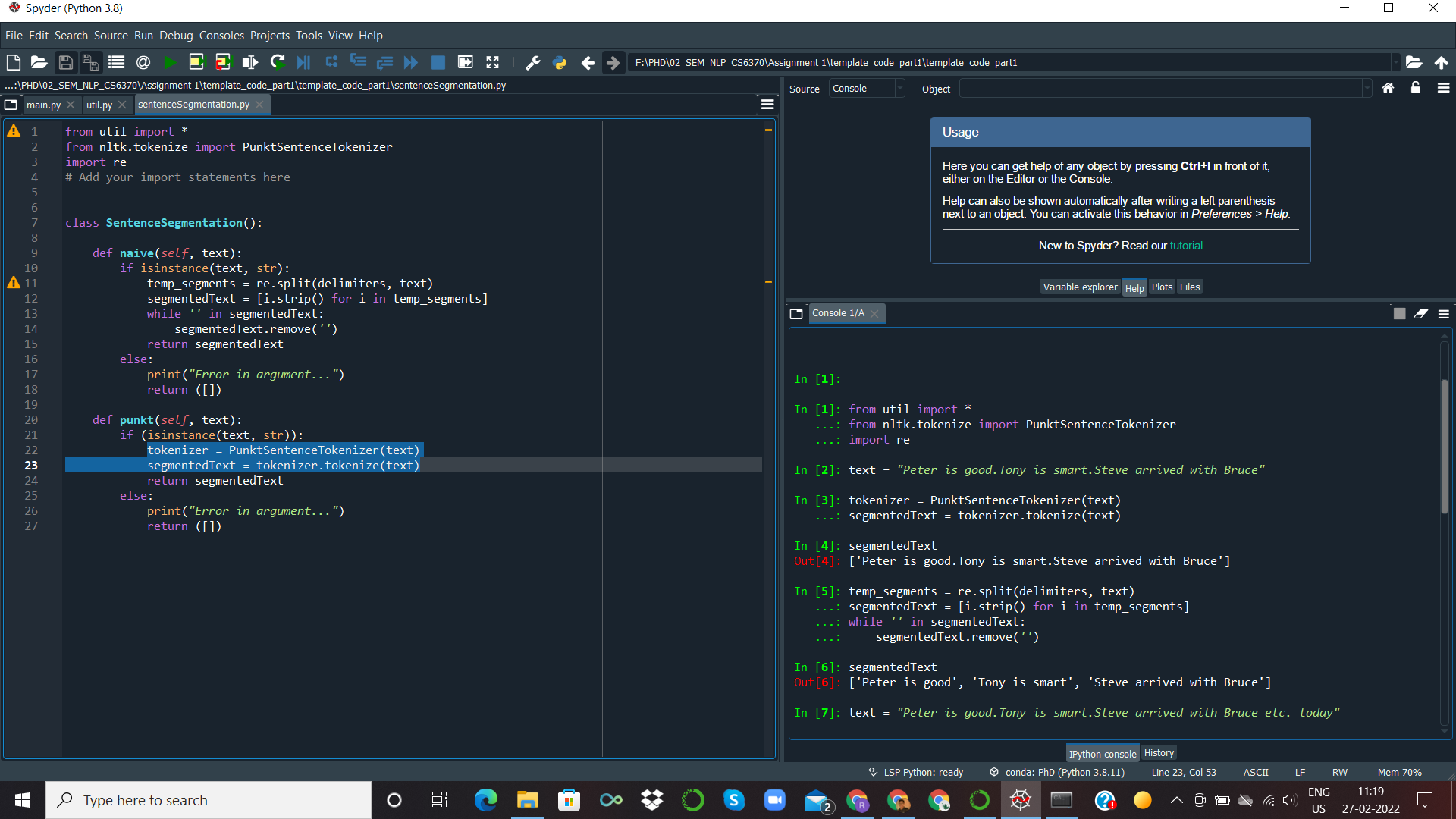
**(a) the first method performs better than the second one (if any)**

ANS: The näive top-down approach for sentence segmentation essentially splits the given piece of text wherever it detects the presence of characters like **‘.’** (period/full-stop), **‘?’** (question mark), **‘!’** (exclamation mark). Whereas, the Punkt Sentence Tokenizer has to be first trained on a sufficiently large collection of text before it can effectively identify abbreviation words, collocations, and words that start or end sentences. Thus the näive top-down approach is reasonably faster than the Punkt Sentence Tokenizer.

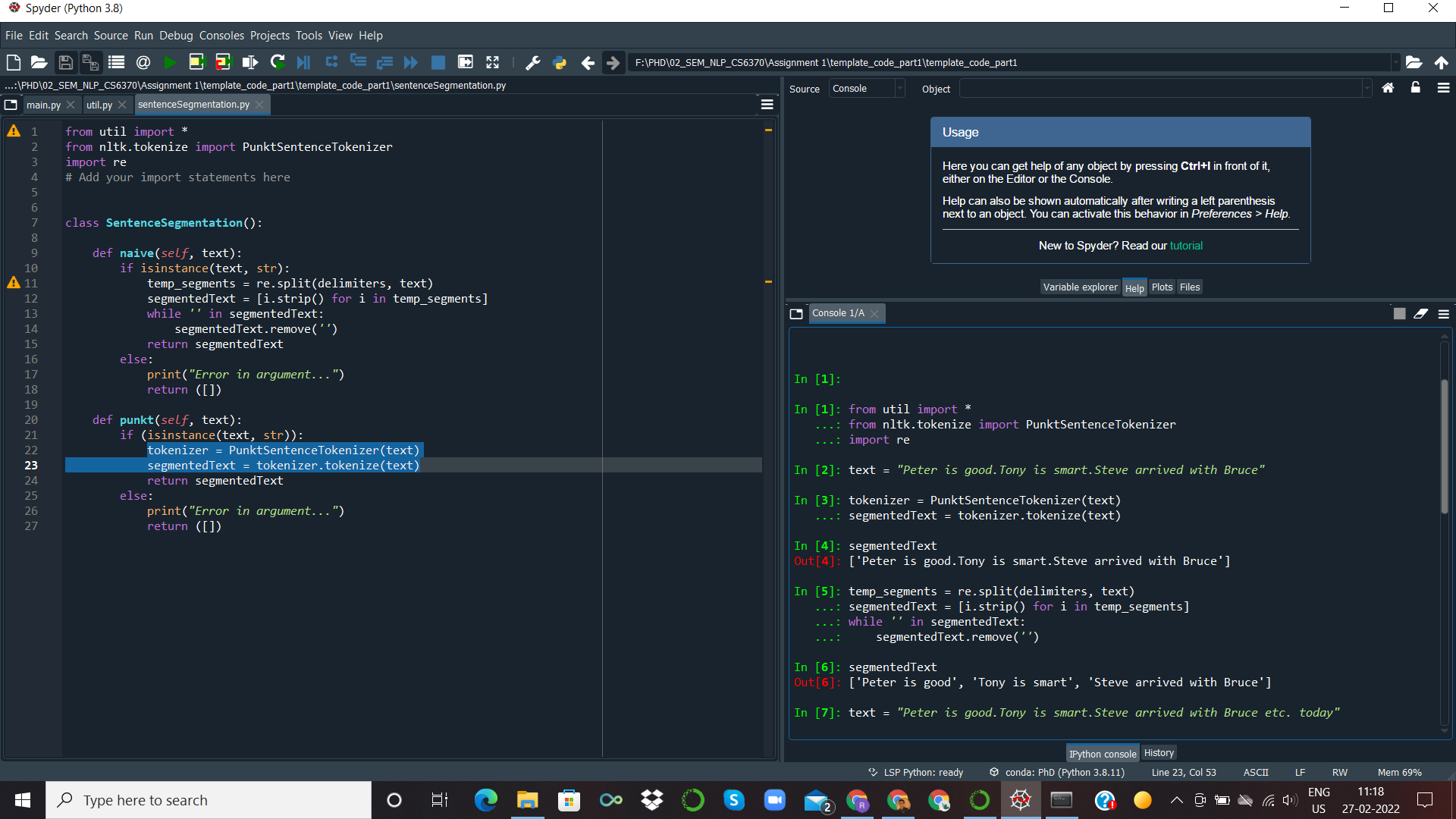
A possible scenario where the näive top-down approach would perform better than Punkt Sentence Tokenizer could be when the text is not properly punctuated as per standard practices in written English. Often while typing in a hurry, people tend to miss spaces between full stops (or similar delimiters). With a sufficient number of occurrences of such mishaps in the text, the Punkt Sentence Tokenizer can end up learning erroneous abbreviations. The näive top-down approach however can easily overcome such mishaps.

An Example:

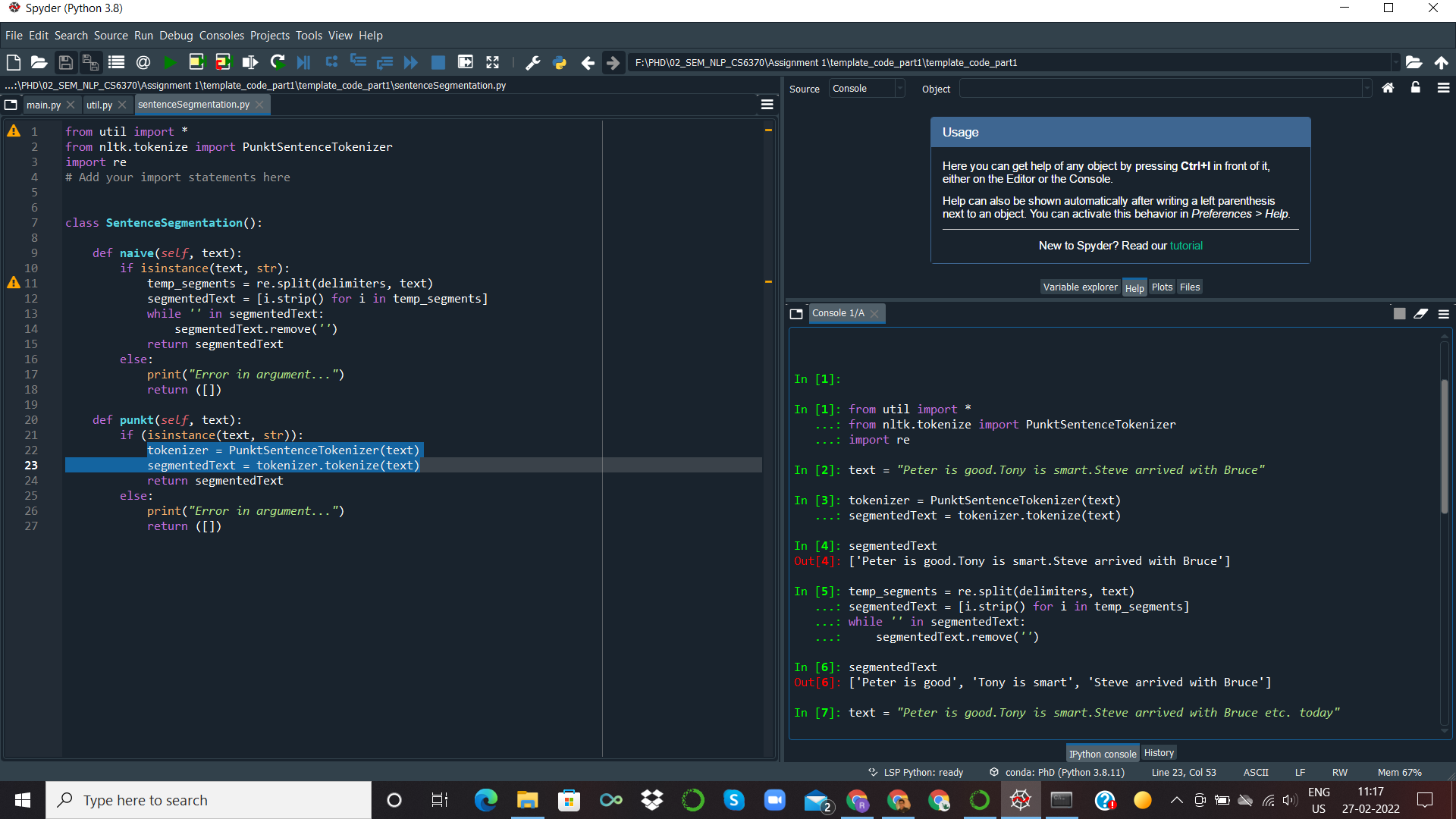
Text: “*Peter is good.Tony is smart.Steve arrived with Bruce*”



Näive top-down approach: “*Peter is good*”, “*Tony is smart*”, “*Steve arrived with Bruce*”



Punkt Sentence Tokenizer: “*Peter is good.Tony is smart.Steve arrived with Bruce*”



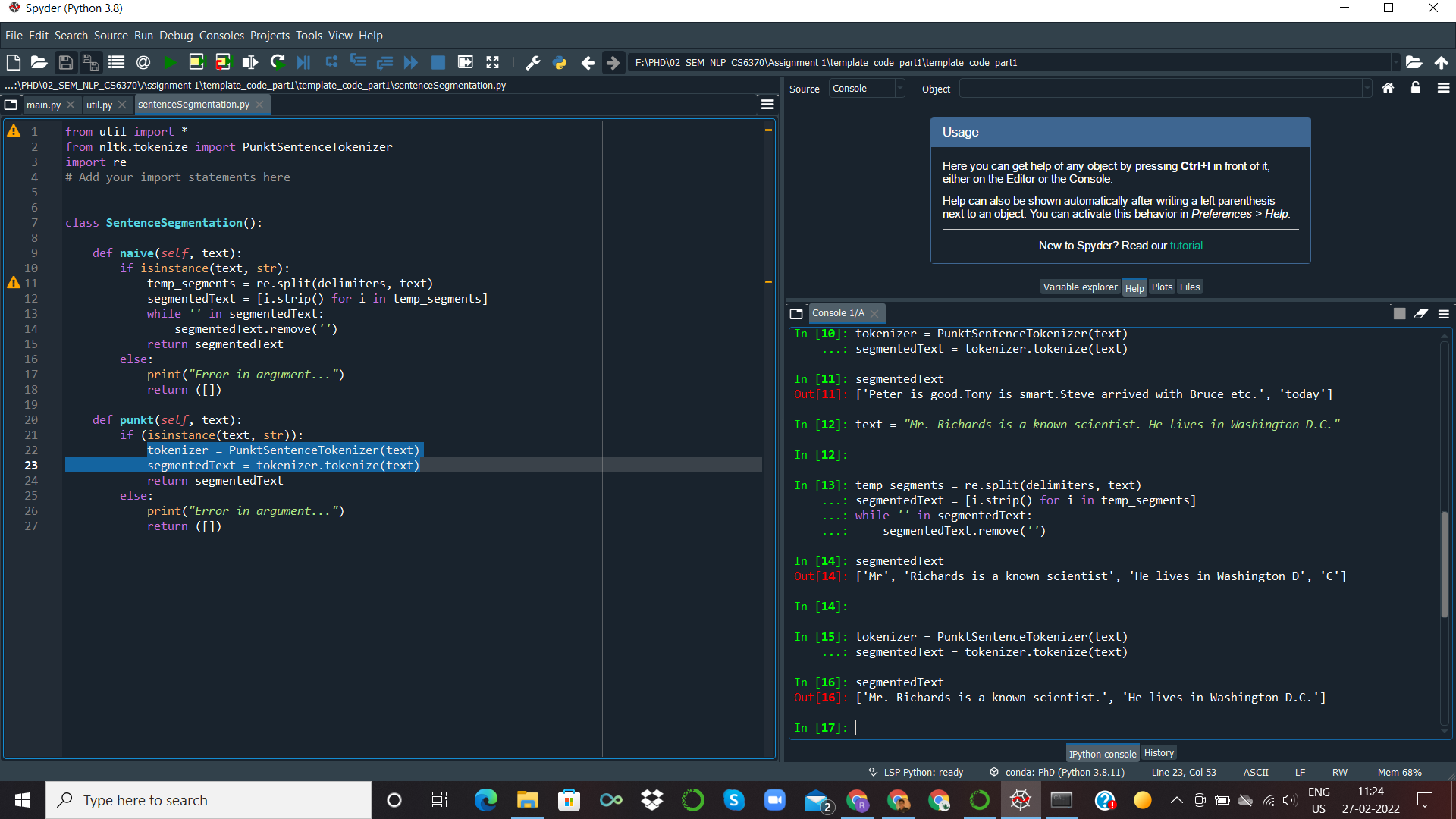
As we see here the Punkt Sentence Tokenizer thinks of “*good.Tony*”, “*smart.Steve*” as abbreviations because of a missing space after the period and hence produces erroneous segmentation. Whereas, the näive top-down approach correctly performs sentence segmentation treating the periods as a delimiter.

**(b) the second method performs better than the first one (if any)**

ANS: As discussed in Q3, the Punkt Sentence Tokenizer is likely to outperform the näive top-down approach in most cases where the text follows proper practices of punctuation in standard English texts and where abbreviations occur with characters like period appearing mid-sentence in addition to occurring at the end of sentences. The Punkt Sentence Tokenizer does so by using an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences. The naive top-down approach fails to recognize the difference between a period being used as an end of sentence marker and as a part of an abbreviation.

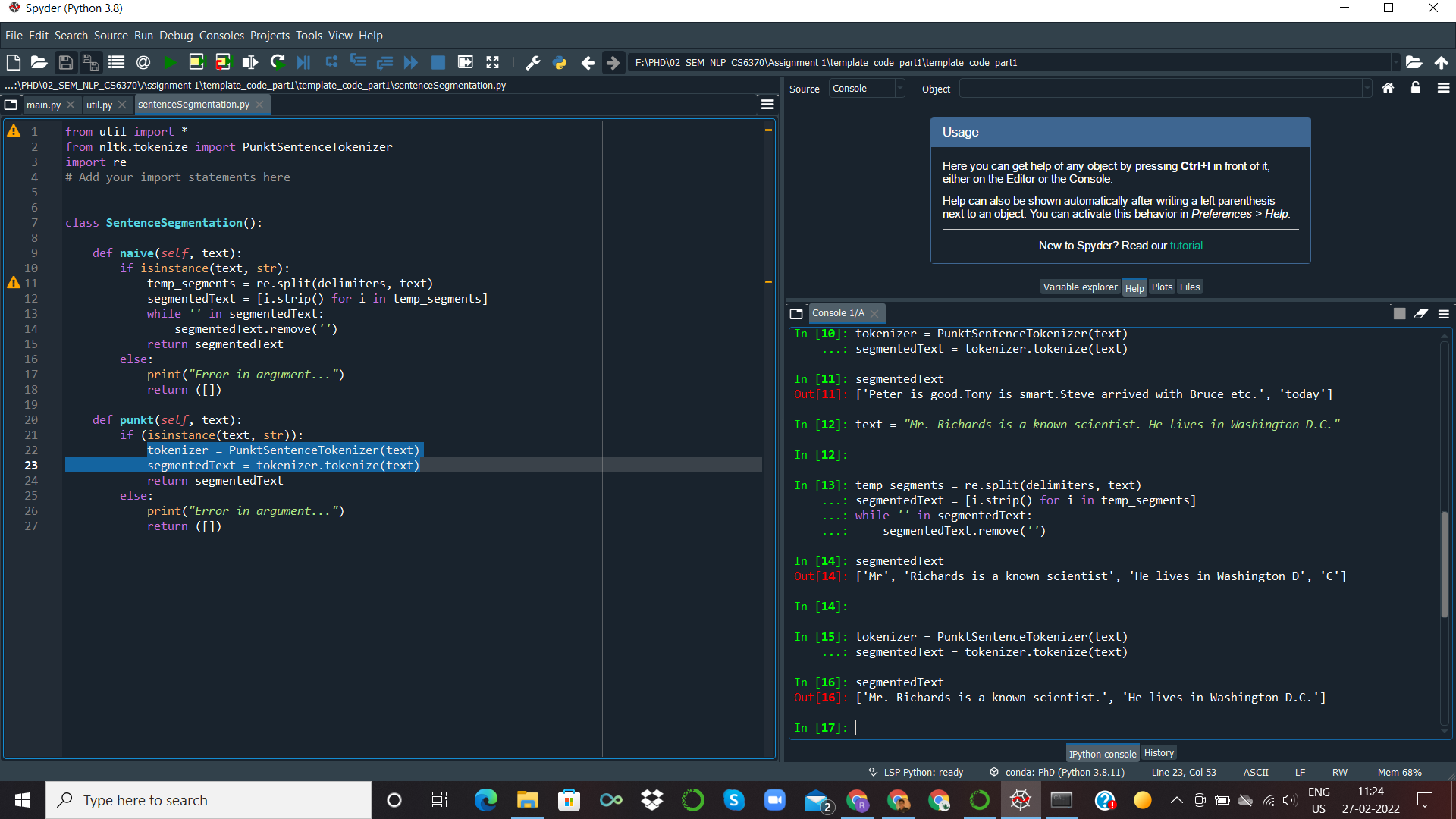
An Example:

Text: “*Mr. Richards is a known scientist. He lives in Washington D.C.*”

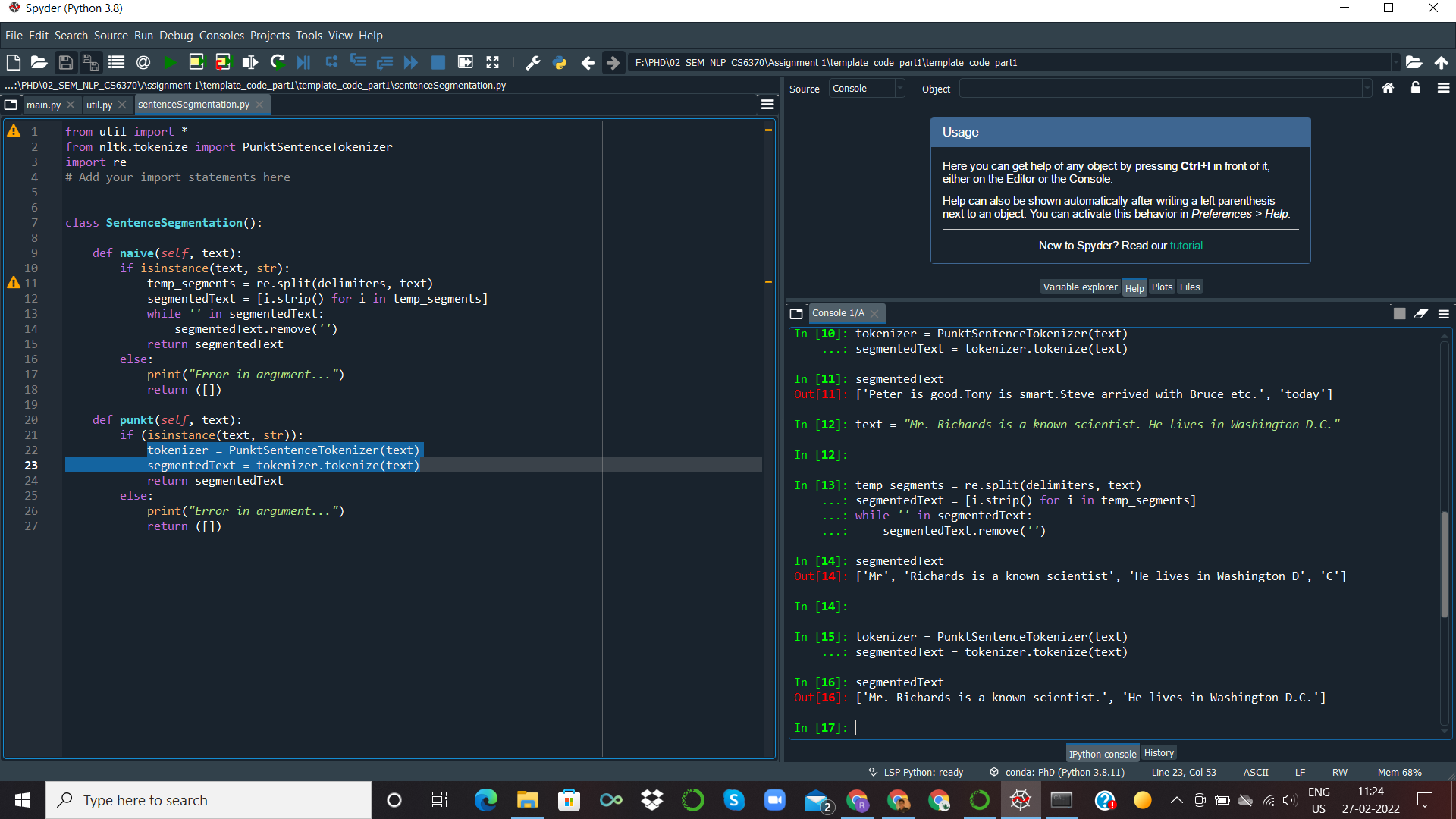


Näive top-down approach:

“*Mr*”, “*Richards is a known scientist*”, “*He lives in Washington D*”, “*C*”



Punkt Sentence Tokenizer: “*Mr. Richards is a known scientist.*”, “*He lives in Washington D.C.*”



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

ANS: Word tokenization refers to the process of breaking down a piece of text into its constituent words. In English texts, it is a standard practice to separate words by one or more space characters in order to maintain clarity and readability of the text. Hence, the most straight-forward and simple top down approach to word tokenization for English texts is the: **White Space Tokenizer**. In this approach, a given sentence or paragraph is tokenized by splitting the input wherever a white space is encountered. We make use of the WhitespaceTokenizer() [5] from nltk.tokenize for performing this naive approach.

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

ANS: NLTK’s Penn Treebank tokenizer uses Bottom-Up approach.

7. **Perform word tokenization of the sentence-segmented documents using**

**(a) The simple method stated above**

ANS**:** Code is submitted separately

**(b) Penn Treebank Tokenizer**

ANS**:** Code is submitted separately

**State a possible scenario along with an example where:**

**(a) the first method performs better than the second one (if any)**

ANS: White space tokenizer works better than Penn Treebank Tokenizer when white space breaks the sentence into meaningful words. As white space tokenizer is fast, so it works better than the Penn Treebank Tokenizer.

**(b) the second method performs better than the first one (if any)**

ANS: Penn Treebank Tokenizer performs better than the White space tokenizer, when we have numerical data. It also helps to reduce the size of vocabulary.

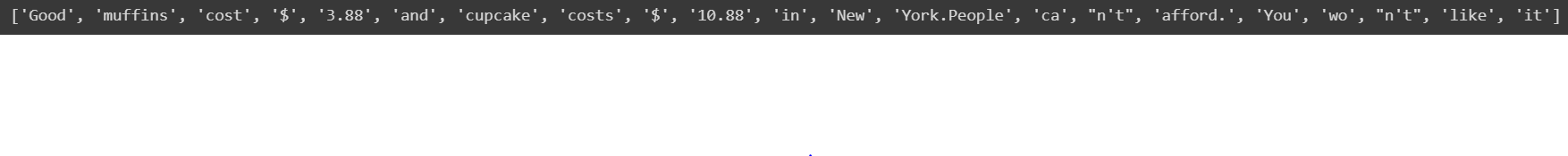
Example:

text = "Good muffins cost $3.88 and cupcake costs $10.88 in New York.People can't afford. You won't like it "

White Space Tokenizer result:



Penn Treebank Tokenizer result:



In this example, all numerical values will not add different words to vocabulary in the Penn Treebank Tokenizer but they will be treated as different words in White space tokenizer.

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

ANS: Stemming and Lemmatization, both techniques are used to reduce the words to normalize form or root words. These techniques reduce unique words in the vocabulary and help to reduce the size of the dictionary.

Difference between Stemming and Lemmatization is:

Stemming[6]:

(i) It generally operates on each word separately without considering the context of words in the sentence.

(ii) Part of Speech tags are not considered generally in stemming.

(iii) It may lead to words without meaning. Example: Moving may get reduced to mov, which does not have any meaning.

(iv) Stemming algorithms remove suffixes and prefixes taking into account a list of common prefixes and suffixes

(v) Stemming operation in the preprocessing step performs really well on tasks like spam classification and basic sentiment analysis.

Lemmatization[6]:

(i) It considers words as well as context of words in the sentence.

(ii) Part of Speech tags are considered in lemmatization.

(iii) It leads to meaningful words. Example: Moving will get reduced to move.

(iv) It takes morphological analysis of the words into account.

(v) Lemmatization in preprocessing step will be useful in language generation tasks like chatbot and question-answering applications as to generate something meaningful, the words should have meaningful representation.

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.**

ANS: Search engine application is an information retrieval process. Given a whole lot of documents in the corpus and a given query, we need to find the most relevant documents. So, in the information retrieval process, we construct a Term Frequency-Inverse Document Frequency(TF-IDF) matrix[7]. This matrix tells us how relevant a word is in a given document of our corpus. We generate this matrix by combination of two measures,Term Frequency and Inverse Document Frequency.

Term Frequency is the count of a word in a particular document. Inverse Document frequency tells us how common a word is in all the documents of our corpus. So, for each word in the search query,we want to find out the TF-IDF score[7]. It would only make sense if the root word reduced from the infected word is actually present in our vocabulary. But in stemming[6], it is not guaranteed that it reduces to meaningful representation of the root word after doing morphological analysis.

**Eg: studies -> studi , change -> chang**

These representations of the word are not present in our vocabulary. But Lemmatization[6] reduces to meaningful representations of normalized form i. e. **lemmas** using part-of-speech tagging and context.

**Eg: studies -> study, change -> change**

Lemmas are present in our vocabulary and now we can generate a TF-IDF score to get the relevant documents. Hence, for a search engine application, lemmatization is preferred over stemming.

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

ANS: The code for this question has been submitted.

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

ANS:The code for this question has been submitted.

12. **In the above question, the list of stopwords denotes top-down knowledge. Can you think of a bottom-up approach for stopword removal?**

ANS: In question 11, we used top-down knowledge i.e. a readily available list of stopwords that were used for stopword removal. The bottom-up approach[8] for stopword removal can be done by two methods.

**a) Statistical approach**

In the **statistical approach[8]**, we find out how frequently a word appears in Documents. We can compute measures like term frequency, document frequency or inverse document frequency and get to know the overall relevance of a word in any document. Generally stopwords are very less relevant in a document and the meaning of the document can be captured even if the stopwords are removed. Hence, stopwords can be identified if a word occurs in more than 50 percent of the documents in the corpus and will have a very low tf-idf score. After identifying them, we can remove all the stopwords using this approach.

**b) Semantic approach**

In the **semantic approach[8]**, we try to calculate the information gained from a stop word. Generally, a stop word carries very little meaning and the information gained from it will also be small. Information gained from a word is inversely proportional to the probability of word occurrence. Hence, the information gained from a word stopword will be very low as the probability of occurrence of a stopword in a document is very high. We can also say that the stopword will have very low discrimination value.

Stopwords can also be identified in another way using **part-of-speech tagging**. Stopwords are more often tagged into the category of articles, prepositions, conjunctions, adverbs and sometimes verbs. So, using pos tagging every word in the corpus and then separating the words that mostly fall into the above categories can be another approach to tackle the problem of stopword removal[8].

========================================================================

REFERENCES:

[1] https://www.math.spbu.ru/user/tseytin/butdu.html

[2] https://www.grammarly.com/blog/end-sentence-punctuation/

[3] https://www.nltk.org/\_modules/nltk/tokenize/punkt.html

[4]https://subscription.packtpub.com/book/application-development/9781782167853/1/ch01lvl1sec12/training-a-sentence-tokenizer

[5] https://www.geeksforgeeks.org/python-nltk-nltk-whitespacetokenizer/

[6] <https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html>

[7]https://monkeylearn.com/blog/what-is-tf-idf/

[8]https://www.researchgate.net/publication/318969652\_AN\_AUTO-GENERATED\_APPROACH\_OF\_STOP\_WORDS\_USING\_AGGREGATED\_ANALYSIS