

Master of Science (MS) in Data Science

Module: ITC6010A1 – Natural Language Processing

Instructor: Dr. Lazaros Polymenakos

Student’ name: Alkiviadis Kariotis 241735

Term: Spring Term 2023

Type: HW#1 Individual Assignment

Submission Date: Friday, May 19

Words: 1295

Table of Contents

[Summary 3](#_Toc135158424)

[Methodology 3](#_Toc135158425)

[Environment Setup 3](#_Toc135158426)

[Libraries and Dependencies 3](#_Toc135158427)

[Code Explanation 4](#_Toc135158428)

[Assumptions 4](#_Toc135158429)

[Word Cloud Generation 5](#_Toc135158430)

[Output Generation 5](#_Toc135158431)

[Challenges and Improvements 5](#_Toc135158432)

# Summary

In the era of big data, handling and analyzing text data has become increasingly important. One of the preliminary steps in text analytics is text pre-processing, a method to clean and prepare text data for further analysis. Text preprocessing is a critical step in natural language processing and machine learning. It involves cleaning and formatting the text data to the form that could be easily understood and analyzed by algorithms. In this task, we were required to write a Python program to preprocess and extract some properties of a plain text document without using any specific libraries except for sentence tokenization.In the assignment we needed to perform basic text preprocessing tasks on a given and then on my own text document. The tasks included counting the number of paragraphs, sentences, and words, identifying distinct words, counting word frequencies, removing stopwords, and considering the implications of processing text in different languages.

# Methodology

## Environment Setup

The program was written in Python 3.8.11, and it requires the Natural Language Toolkit (NLTK) for sentence tokenization. To run the code, follow these steps:

1. Ensure that Python 3, NLTK as well as all other libraries mentioned in the Libraries and Dependencies section are installed on your machine.
2. Run the script using the command: HW#1\_Code\_Alkiviadis\_Kariotis\_241735.ipynb or by either through Google Collab or Anaconda.

## Libraries and Dependencies

This script relies on several Python libraries. Here's a list of all­ of them:

* Python Standard Library: This script uses various built-in Python modules like string, re, and io. String and re are used for string manipulations and regular expressions operations respectively, while io is used to redirect standard output to a string variable.
* Natural Language Toolkit (NLTK): This is an essential library for Natural Language Processing (NLP) in Python. It is used in this script for sentence tokenization, which is the process of dividing a string of written language into its component sentences.
* Collections: This Python built-in library is used for creating a Counter object, which is a dict subclass for counting hashable objects. It's a collection where elements are stored as dictionary keys and their counts are stored as dictionary values.
* Matplotlib: This is a plotting library for Python. It's used in this script to display the word cloud generated from the WordCloud function.
* Wordcloud: This library is used to generate word clouds. Word clouds are a popular way to depict the most frequently occurring words in a text, where the size of each word in the cloud corresponds to its frequency.
* PIL (Python Imaging Library): It adds image processing capabilities to our Python interpreter.
* FPDF: This library is used for generating the output PDF. It stands for Free PDF, and it allows for PDF generation with Python.

## Code Explanation

The program begins by reading the input text from a file (either the one provided or the one that I create into the code). The text is split into paragraphs based on newline characters, and then each paragraph is tokenized into sentences using the NLTK library. Each sentence is further tokenized into words based on whitespace characters. The words are then processed to handle contractions and separate punctuations, as specified in the assignment instructions. After the tokenization process, the program counts the number of paragraphs, sentences, words, and distinct words. It also generates a frequency count of each word. Stopwords are then removed from the list of words based on the provided list of English stopwords.

Here is a more detailed steps and functions, which are applied in the script to analyze the input text:

1. Sentence Segmentation: The Natural Language Toolkit (NLTK) library's sent\_tokenize function is used to break down the text into individual sentences.
2. Tokenization: The text is tokenized by splitting it based on whitespace characters. The Python split() method and the regular expression (re) module are used for this purpose. Contractions are also processed during tokenization. For example, "I'm" is separated into "I" and "am".
3. Handling Punctuation and Contractions: The script uses helper functions to process contractions (process\_contractions()) and separate punctuations (process\_punctuation()). A specific function (process\_s\_contraction()) handles the "'s" contraction case, which can indicate either a contraction of "is" or a possessive form depending on the context.
4. Counting Word Types and Frequencies: The Python collections library's Counter function is used to count the frequency of each word. A word's type is determined by its unique appearance in the text.
5. Counting Paragraphs: Paragraphs are counted by splitting the text on newline characters.
6. Stopword Removal: A list of stopwords is provided, and these words are removed from the text using list comprehension.

## Assumptions

The program makes a few assumptions:

* Punctuation characters are treated as separate tokens.
* Contractions are split into individual words based on specific rules.
* The input text does not contain any special characters or symbols that could disrupt the tokenization process.

## Word Cloud Generation

In the final part of our code, we provide a visualization of the 50 most common non-stopwords in the form of a word cloud. Word clouds are graphical representations of word frequency that give greater prominence to words that appear more frequently. After removing stopwords from our list of words, we use the Counter class from the collections library to count the frequency of each word. We then select the 50 most common words. Using the WordCloud function from the wordcloud library, we generate a word cloud, with the size of each word corresponding to its frequency in the text. We use the matplotlib library to display this word cloud. This word cloud provides a quick and intuitive understanding of the main themes in our text.

## Output Generation

After the text preprocessing, the script generates multiple outputs:

1. It prints to the console the number of paragraphs, sentences, words (tokens), distinct words, word frequency counts, and the words after removing stopwords.
2. It generates a word cloud of the 50 most common non-stopwords in the text and displays it using matplotlib. This word cloud provides a quick and intuitive understanding of the main themes in our text.
3. It also saves the word cloud as a JPEG file.
4. It creates a PDF document that contains all the outputs mentioned above. This document includes the various counts, the list of words after removing stopwords, and the word cloud. The PDF is generated using the FPDF library.

# Challenges and Improvements

The assignment was challenging, especially with regards to handling contractions and punctuation. The program could be improved by implementing a more sophisticated word tokenization process that can accurately handle various special cases in English text.

Further improvements could include adding error handling and validation checks to ensure that the input text is formatted correctly and contains valid English words. Another future extension of this program could involve adapting it to handle other languages. This would require a different set of tokenization rules and stopwords for each language, which could be specified in separate configuration files. The program could then be modified to select the appropriate configuration file based on the language of the input text. In the context of processing text in different languages, it's important to note that not all languages follow the same rules as English. For example, Chinese doesn't use spaces to separate words, and different tokenization methods would be required. Additionally, different languages have different sets of stopwords, so the list of stopwords would need to be adjusted for different languages. Therefore, while the methodology used in this exercise could be adapted for different languages, it is not universally applicable to all languages.

Overall, this task provided valuable experience in text preprocessing, a fundamental step in natural language processing and machine learning. It highlighted the importance of understanding the specificities of the language we are dealing with and the need for efficient and effective programming techniques.