PROJECT 1

Text preprocessing with NLTK

Gabriel Martinica

MODULES USED:

OS Module:

As defined in https://docs.python.org/3/library/os.html: “This module provides a portable way of using operating system dependent functionality.”

I used the OS module to create directories, as well as read files from a specific directory. More specifically, I created directories for each of the steps so that the output files were organized accordingly. Moreover, the input files given to the program are read by the help of this module

NLTK module:

Referenced for the project: https://www.nltk.org

This module provides all the language processing functionalities needed for this such as word tokenization, porter stemmer libraries, and stop words list.

EXPLANATION OF WORK:

Each module of the project is separated into functions. In total, there are 5 functions for each of the 5 modules, these functions are in the main() function of the file “p1.py”. I will explain each function in detail:

read_files()

This function takes as input two parameters: A string with the path of the directory that contains the files to be read, and an int to determine how many of those files to read from the directory.

It returns a dictionary with the following values:

{document: text}

Here, the document is the name of the file from the directory, and the text is the content of the file after removing metadata tags and HTML entities.

This function lists all the files from a given directory, sorts them alphabetically to get the first 5 files and extracts the raw text.

For each of the files, the text is processed to get the raw content from each article. In my case, I read the file line by line up until I find a <BODY> tag> After this I proceed to append the text from that tag up until I find the end of the article which is demonstrated by the word "reuter" appearing in a single line. I stop appending the text and start on the lookout for the next body tag. Rinse and repeat for the other articles. This gives me the text of each article from which I can then use for the following steps.

It finally writes the contents of each processed file into a new directory containing the modified files.

One strength is that the text of each article can be find accordingly and there is not a big mess of metadata around the file.

The time complexity of this operation is O(n^2) as it must go through the list of files and through the contents of each file.

A weakness of this is that the info from the metadata not related to the body like the ID and the title are lost, so the text cannot be categorized, and a specific article cannot be found by ID. However, if you know the text of the article you can in theory look for that and it will appear.

tokenize()

This function takes as input the dictionary {document: text} and tokenizes the contents of each file using the function:

```
word_tokenize(docs[filename])
```

This nltk function is very useful as it outputs a list of each word in the corpus and gets rid of all the whitespace obtained from preprocessing.

The function returns a dictionary with the content of each file tokenized, divided into a list of words.

Time complexity of this operation is O(n^2) as it goes through the files and the contents of each file.

This function also removes punctuation marks. The good thing is that the tokenized list does not contain most of the common punctuation marks, but it is not perfect as there are different types of punctuation marks that are not accounted for like "……." If that happened to exist in the corpus. Maybe a better suited regex could remove those elements in a more efficient manner.

It writes the contents of the dictionary into output files to directory:

```
"step2files"
```

Make_lowercase()

Similar as above, takes as input the dictionary {document: tokenized text} and converts all the tokenized words to lowercase.

It returns the same dictionary with all the tokenized text converted to lowercase.

Outputs the contents of the dictionary to directory

```
directory = "step3files"
```

Apply_porter_stemmer()

Takes as input the dictionary {document: tokenized lower-case text} and applies the Porter Stemmer algorithm to all the words in the tokenized text. Returns the dictionary after applying the algorithm to each token.

NLTK provides a function that implements this algorithm, and it is the one used for this function.

It writes the contents of the dictionary to a directory

```
directory = "step4files"
```

Remove_stop_words()

Takes as input the dictionary with tokenized words and a list of stop words and proceeds to remove those words from the tokenized words list. It returns the dictionary without the stop words and writes the contents of the dictionary to directory

```
directory = "step5files"
```