COMP 6791 Information Retrieval and Web Search

Project 1 Demo

Student ID: 40256600

By, Ganesh Chandrasekar

Code Walkthrough

Implementation:

- The code is written to run in a standalone manner for each function.
- The required libraries for this program to run properly.

```
import os
import re
import nltk
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from nltk.tokenize import RegexpTokenizer
```

- **clean_text Function:**
  - I use the clean_text function to remove special characters.
  - Using a custom regex to clean the text.
  - This unique character was seen after reading and printing a raw corpus text.
  - The cleaned_text is passed on to the following module.

```
# This clean_text function is used because I encountered (special character)
def clean_text(text):
    cleaned_text = re.sub(r'[\x00-\x1F\x7F-\x9F]', ' ', text)
    cleaned_text = ' '.join(cleaned_text.split())
    return cleaned_text
```

- **tokenizzzz Function:**
  - This function uses custom regex along with NLTK tokenize to detect custom values like: “3,000,00”, “U.S. Embassy”, and “1.30”.
  - Returns tokenized value

```
def tokenizzz(text):
    mytokenizer=RegexpTokenizer(r'\S[A-Z|\.]+\w*|\'s|
[A-Z|a-z]+\'[a-r|t-z]+|\$|[0-9]+\,[0-9]+\,*[0-9|\,]*|
[0-9]+\.[0-9]+|\w+|\d+|\S+')
```

```
    tokens = mytokenizer.tokenize(text)
    return tokens
```

- **tokenizzzToLowercase Function**
  - The function takes each token and converts it to lowercase.
  - The list of values is joined and returned as lowercase text rather than a list of values.

```
# Lowercase
def tokenizzzToLowerCase(text):
    lowercased_tokens = []

    # Iterate through the tokens in the list and make each token lowercase
    for token in text:
        lowercased_token = token.lower()
        lowercased_tokens.append(lowercased_token)

    # Join the lowercase tokens into a single string
    lowercased_text = ' '.join(lowercased_tokens)
    # lowercased_text = text.lower()

    return lowercased_text
```

- **porterStemming Function**
  - This method performs stemming of words using the NLTK's PorterStemmer().
  - The Porter stemming algorithm aims to reduce words to their root or base form by stripping off prefixes and suffixes, retaining the common or stem part of the term to save memory.
  - Each word is stemmed and joined to the default pattern to be more readable.
  - Returns Stemmed output (with minor errors like Overstemming or Understemming).

```
    def porterStemming(text):
      stemmer = PorterStemmer()
      words = text.split()
      stemmed_words = [stemmer.stem(word) for word in words]
      stemmed_text = ' '.join(stemmed_words)
      return stemmed_text
```

- **Remove_custom_stopwords Function:**
  - This function removes stopwords from the read corpus text.
  - We use a custom_stopwords list to provide flexibility, this is extended with the stopwords that are available in the NLKT package.
  - The text is split if it's in the list and the words are joined to retain the default pattern.
  - Returns texts with no stopwords.

```
    def remove_custom_stopwords(text, custom_stopwords=None):
          if custom_stopwords is None:
                custom_stopwords = []

          words = text.split()
    filtered_words = [word for word in words if word.lower() not in
custom_stopwords]
          filtered_text = ' '.join(filtered_words)
   return filtered_text
```

- **Process_article Function:**
  - This function is constructed to call the above functions and write the output into separate files like:
    - ‘Tokenizer-output’, ‘Lowercased-output’, ‘Stemmed-output’, ‘No-stopword-output’.
    - Also, a file is created to show the stopwords used ‘Stopwords-used-for-output’.

```
    def process_article(article_text, article_output_folder):
        # Custom Stopwords list - as per the project requirement
        custom_stopwords = ['a', 'an', 'and', 'are', 'as', 'at', 'for', 'from',
'has', 'he', 'in', 'is','it', 'its', 'of', 'on', 'that', 'the', 'to', 'was',
'were', 'with']
        # Extending to the NLTK's stopwords
        custom_stopwords.extend(stopwords.words('english'))

        # Tokenization
        pivot2_text = tokenizzz(article_text)
        tokens_output_file = os.path.join(article_output_folder,
'Tokenizer-output.txt')
        with open(tokens_output_file, 'w', encoding='utf-8') as output_file:
            output_file.write(" ".join(pivot2_text))
        # Lowercasing
        pivot2_text = tokenizzzToLowerCase(pivot2_text)
        lowercased_output_file = os.path.join(article_output_folder,
'Lowercased-output.txt')
        with open(lowercased_output_file, 'w', encoding='utf-8') as
output_file:
            output_file.write(pivot2_text)
        # Stemming
        pivot2_text = porterStemming(pivot2_text)
        stemmed_output_file = os.path.join(article_output_folder,
'Stemmed-output.txt')
        with open(stemmed_output_file, 'w', encoding='utf-8') as output_file:
            output_file.write(pivot2_text)
        # Stopword Removal
        pivot2_text = remove_custom_stopwords(pivot2_text, custom_stopwords)
        no_stopword_output_file = os.path.join(article_output_folder,
'No-stopword-output.txt')
        with open(no_stopword_output_file, 'w', encoding='utf-8') as
output_file:
            output_file.write(pivot2_text)

        # Write stopwords used
        stopwords_used_file = os.path.join(article_output_folder,
'Stopwords-used-for-output.txt')
        with open(stopwords_used_file, 'w', encoding='utf-8') as output_file:
            output_file.write("\n".join(custom_stopwords))
```

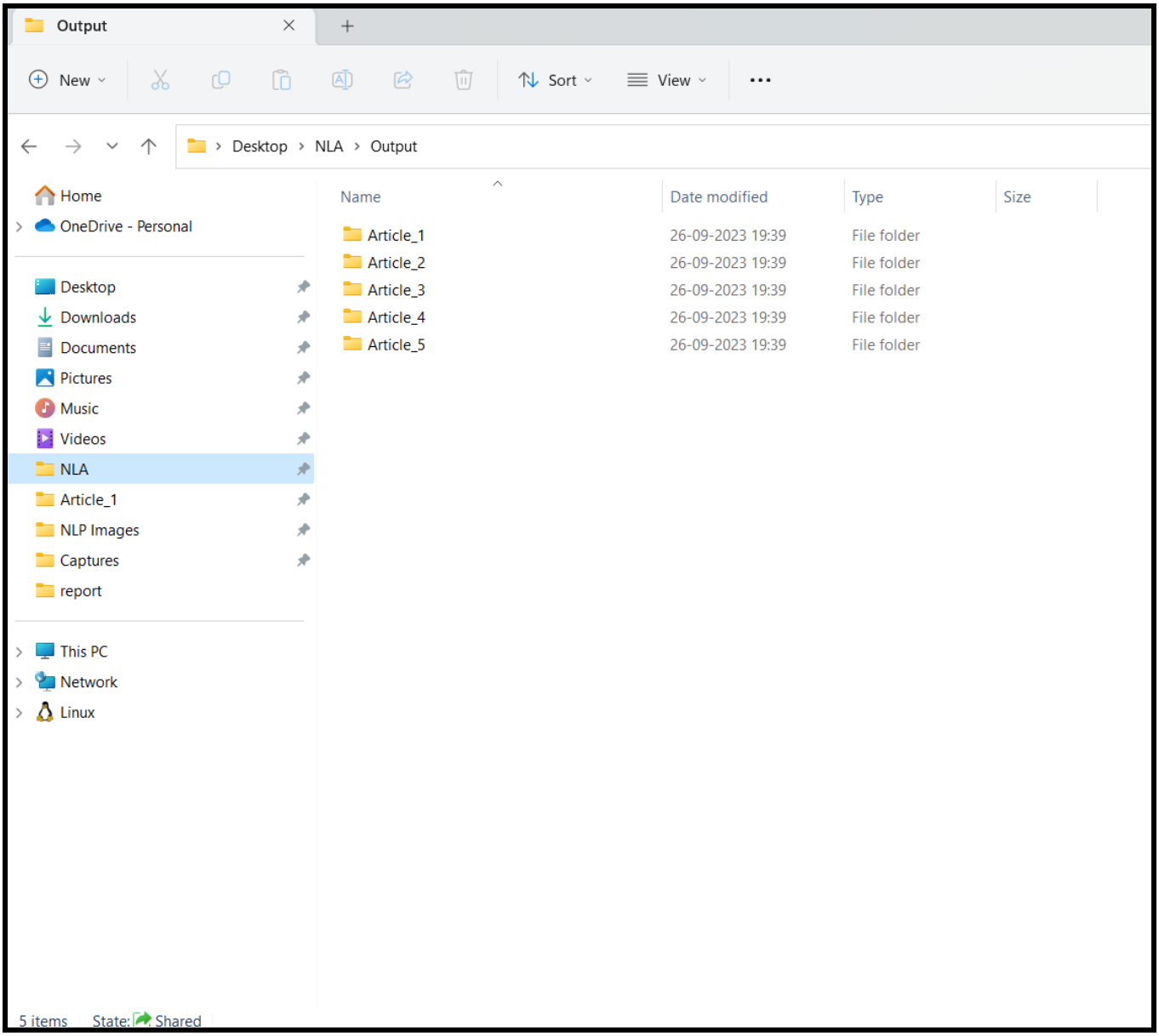
- **Process_reuters_corpus Function:**
    - This function is used to read the reuters21578 file and fetch the .sgm files
    - Based on the project description this function has a num_articles value to fetch exactly the first five news articles.
    - Each file will have a folder with five files from the **process_article** function.
    - Returns the expected structure of folders and files for this project

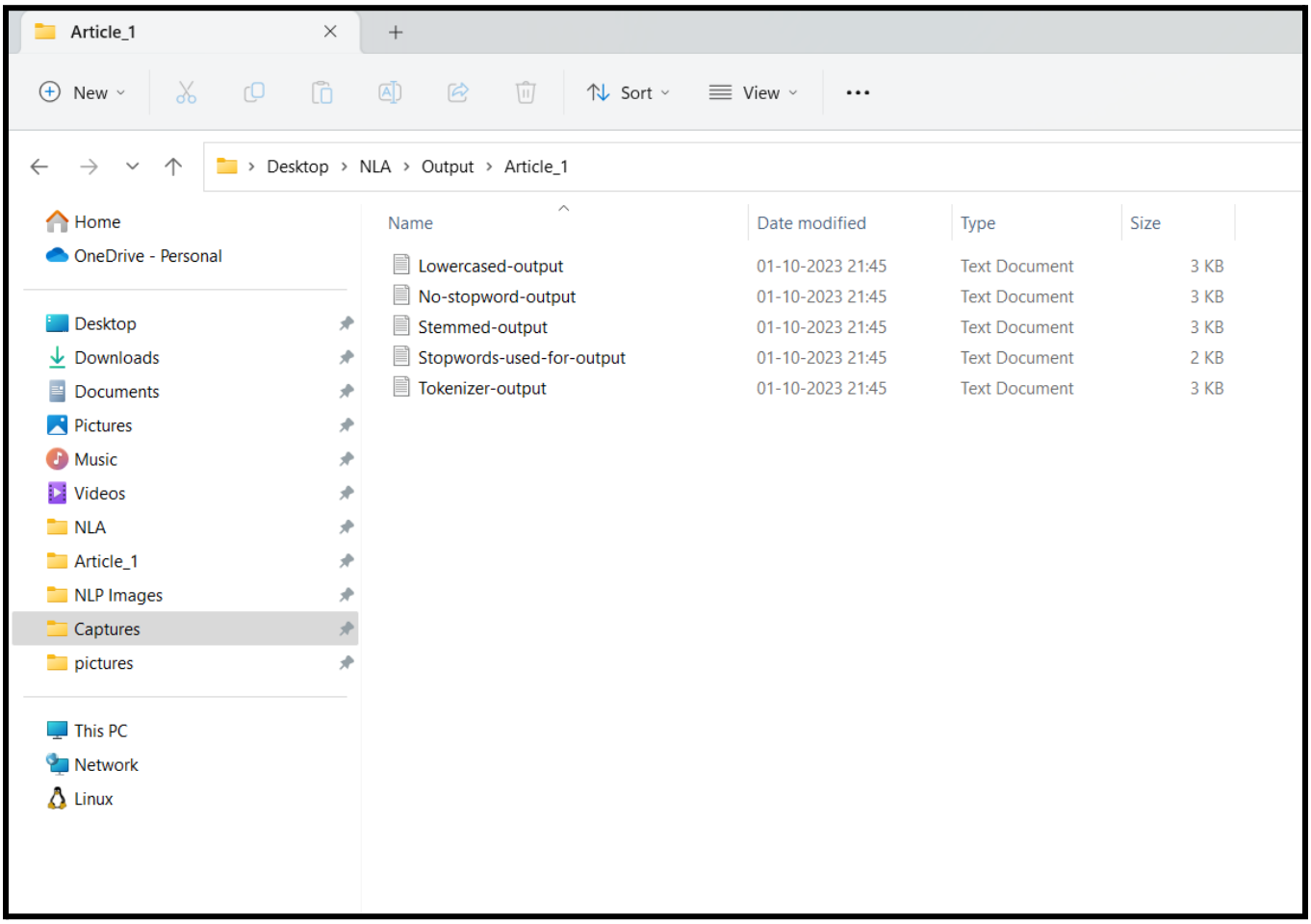
```
# This function is used to read the corpus file from the local storage and traverse
# to the sgm files and select the first five Reuter's news items in the corpus
def process_reuters_corpus(corpus_root, output_root, num_articles=5):
    article_count = 0  # Counter for processed articles
    for root, _, files in os.walk(corpus_root):
        for filename in files:
            if article_count >= num_articles:
                break
            # traversing to the .sgm file path
            if filename.endswith(".sgm"):
                with open(os.path.join(root, filename), 'r', encoding='ISO-8859-1')
as file:
                    # using beautifulSoup to parse the retures file
                    soup = BeautifulSoup(file, 'html.parser')
                    for i, newsitem in enumerate(soup.find_all('reuters')):
                        # This counter is used to select the first five files
                        if article_count >= num_articles:
                            break
                        # Accessing the text part of the file
                        body = newsitem.find('text')
                        if body:
                            artText = body.get_text()
                            cleaned_article_text = clean_text(artText)
                            article_output_folder = os.path.join(output_root,
f'Article_{i + 1}')
 # making the folders for the five news articles and calling the functions to
                            os.makedirs(article_output_folder, exist_ok=True)
                            process_article(cleaned_article_text,
article_output_folder)
                            article_count += 1
```

```
# Usage to process only the first five articles

# To run the code change the below directories alone
corpus_root = 'C:/Users/cbsag/Desktop/NLA/IRT'
output_root = 'C:/Users/cbsag/Desktop/NLA/Output'
num_articles_to_process = 5
process_reuters_corpus(corpus_root, output_root, num_articles_to_process)
```

- **Output (Screenshot)**





- **Limitation**
  - While successful, the pipeline confronts issues such as uncertain word boundaries in tokenization, possible word meaning changes during lowercasing, mistakes in stemming irregular words, and the limits of a predetermined stopword list. Addressing these constraints may improve the pipeline's performance in future revisions.

- **Conclusion:**
  - This project describes an NLTK-based text processing pipeline for the Reuters-21578 corpus, which includes modules for tokenization, lowercasing, stemming, and stopword removal. Despite inherent difficulties, such as uncertain word boundaries and probable semantic meaning loss, the pipeline provides a solid basis for rapid text analysis and information retrieval.