Blackcoffer

Consulting Website: https://blackcoffer.com | https://lsalead.com/
Web App Products: https://netclan.com/ | https://insights.blackcoffer.com/ | https://hirekingdom.com/ | https://workcroft.com/
Mobile App Products: Netclan | Bwstory

Data Extraction and NLP

The provided Python script appears to be a text analysis tool that extracts information from articles based on a list of URLs stored in an Excel file. Here's an explanation of how the script works:

Dependencies:

nltk: Natural Language Toolkit, used for natural language processing tasks.

textblob: Used for sentiment analysis and other text processing tasks.

textstat: Used for computing readability and complexity statistics.

goose3: A web scraping library for extracting information from articles.

pandas: A data manipulation library for handling data in tabular form.

Function Definitions:

read_stopwords_from_file(file_path): Reads stopwords from a file and returns a list.

create_word_dictionary(file_path): Reads words from a file and creates a dictionary with lowercase words as keys.

read_urls_from_excel(file_path, sheet_name='Sheet1', column_name='URL'): Reads URLs from an Excel file.

Main Script:

It checks if the specified Excel file containing URLs exists.

If the file exists, it reads the URLs and iterates over each one.

For each URL, it extracts the text using the goose3 library.

It then removes custom stopwords from the text and performs various analyses on the cleaned text.

The analyses include sentiment analysis, word frequency distribution, additional analysis using TextBlob, textstat, and NLTK functions.

The results are stored in a list of dictionaries (results) and then converted into a Pandas Data Frame.

The Data Frame is exported to an Excel file.

How to Run the Script:

Ensure that you have all the required dependencies installed (nltk, textblob, textstat, goose3, pandas).

Download the necessary NLTK resources by running nltk.download('punkt') and nltk.download('stopwords').

Provide the correct paths for the input Excel file (excel_file_path) and the output Excel file (output_excel_file).

Run the script using a Python interpreter.

Required Dependencies:

Install the necessary dependencies using the following:

pip install nltk textblob textstat goose3 pandas

Additionally, ensure that you have the NLTK resources downloaded by running the provided nltk.download commands.

Note:

Before running the script, make sure that you have the required permissions to access the URLs and that the URLs are accessible.

The script relies on external resources (URLs) and might behave unexpectedly if the structure of the article's changes.

To run the script, save it with a .py extension (e.g., analyze_text.py) and execute it using a Python interpreter:

```
python analyze_text.py
```

Make sure to adjust file paths and dependencies based on your system and requirements

The script of this assignment is to extract textual data articles from the given URL and perform text analysis to compute variables that are explained below.

```
import nltk

from nltk.tokenize import word_tokenize, sent_tokenize

from nltk.corpus import stopwords

from nltk.probability import FreqDist

from nltk.sentiment import SentimentIntensityAnalyzer

from nltk import pos_tag

from textblob import TextBlob

from textstat import syllable_count

from goose3 import Goose

import os

import string

import pandas as pd
```

```
# Download necessary NLTK resources
nltk.download('punkt')
nltk.download('stopwords')

def read_stopwords_from_file(file_path):
    with open(file_path, 'r') as file:
        return file.read().splitlines()

def create_word_dictionary(file_path):
    with open(file_path, 'r') as file:
        return {word.lower(): True for word in file.read().splitlines()}

def read_urls_from_excel(file_path, sheet_name='Sheet1', column_name='URL'):
    try:
        df = pd.read_excel(file_path, sheet_name=sheet_name)
        return df[column_name].tolist()
    except Exception as e:
        print(f"Error reading Excel file: {e}")
        return []

# Specify the path to the Excel file containing URLs
excel_file_path = r"C:\Users\lenovo\Downloads\Input.xlsx"
```

```
# Check if the file exists before proceeding
if os.path.exists(excel_file_path):
    # Read URLs from the Excel file
    urls = read_urls_from_excel(excel_file_path)

    # Create an empty list to store results
    results = []

    # Loop through each URL
    for url in urls:
        try:
            # Extracting text from the article
            g = Goose()
            article = g.extract(url=url)

            # Check if the article is not empty
            if not article.cleaned_text:
                print(f"Empty article for URL: {url}")
                continue

            # Read stopwords from a folder
            stopwords_folder_path = r"C:\Users\lenovo\Downloads\StopWords-
20231212T124238Z-001\StopWords"
```

```
        all_custom_stopwords = [word.lower() for file_path in
[os.path.join(stopwords_folder_path, file) for file in os.listdir(stopwords_folder_path) if
file.endswith('.txt')] for word in read_stopwords_from_file(file_path)]


        # Remove custom stopwords from the text
        filtered_words = [word.lower() for word in word_tokenize(article.cleaned_text) if
word.lower() not in all_custom_stopwords and word.isalnum()]
        filtered_text = ' '.join(filtered_words)


        # Display the result
        print(f"\nAnalysis for URL: {url}")
        print("Original Text:")
        print(article.cleaned_text)
        print("\nText without Custom Stopwords:")
        print(filtered_text)


        # Read positive and negative words from files and create dictionaries
        positive_words_dict =
create_word_dictionary(r"C:\Users\lenovo\Downloads\MasterDictionary-20231212T124239Z-
001\MasterDictionary\positive-words.txt")
        negative_words_dict =
create_word_dictionary(r"C:\Users\lenovo\Downloads\MasterDictionary-20231212T124239Z-
001\MasterDictionary\negative-words.txt")


        # Sentiment analysis
        sia = SentimentIntensityAnalyzer()
```

```
sentiment_score = sia.polarity_scores(article.cleaned_text)


# Count positive and negative words
positive_count = sum(1 for word in filtered_words if word in positive_words_dict)
negative_count = sum(1 for word in filtered_words if word in negative_words_dict)


# Display the result
print("Sentiment Analysis Score:")
print(sentiment_score)
print("\nPositive Word Count:", positive_count)
print("Negative Word Count:", negative_count)


# Frequency distribution of words
fdist = FreqDist(filtered_words)
print("Word Frequency Distribution:")
print(fdist.most_common(10))


# Additional analysis using TextBlob, textstat, and NLTK functions
blob = TextBlob(article.cleaned_text)
subjectivity_score = blob.sentiment.subjectivity


sentences = sent_tokenize(article.cleaned_text)
sentence_lengths = [len(word_tokenize(sentence)) for sentence in sentences]
avg_sentence_length = sum(sentence_lengths) / len(sentence_lengths)
```

```
words = word_tokenize(article.cleaned_text)
complex_word_count = sum(1 for word in words if syllable_count(word) > 2)
total_word_count = len(words)
percentage_complex_words = (complex_word_count / total_word_count) * 100

fog_index = 0.4 * (avg_sentence_length + percentage_complex_words)

sentences = sent_tokenize(article.cleaned_text)
words_per_sentence = [len(word_tokenize(sentence)) for sentence in sentences]
avg_words_per_sentence = sum(words_per_sentence) / len(words_per_sentence)

complex_words = [word for word in words if syllable_count(word) > 2]
complex_word_count = len(complex_words)

word_count = len(words)

total_syllables = sum(syllable_count(word) for word in words)
average_syllables_per_word = total_syllables / total_word_count

pos_tags = pos_tag(words)
personal_pronoun_tags = ['PRP', 'PRP$', 'WP', 'WP$']
personal_pronoun_count = sum(1 for word, pos in pos_tags if pos in personal_pronoun_tags)
```

```
        total_characters = sum(len(word) for word in words)
        average_word_length = total_characters / total_word_count

        # Append the results to the list
        results.append({
            "URL": url,
            "Title": article.title,
            "Sentiment Score": sentiment_score["compound"],
            "Positive Word Count": positive_count,
            "Negative Word Count": negative_count,
            "Top Words and Frequencies": fdist.most_common(10),
            "Subjectivity Score": subjectivity_score,
            "Average Sentence Length": avg_sentence_length,
            "Percentage of Complex Words": percentage_complex_words,
            "Fog Index": fog_index,
            "Word count": total_word_count,
            "Complex word count": complex_word_count,
            "Average Number of Words Per Sentence": avg_words_per_sentence
            # Add more fields as needed
        })

    except Exception as e:
        print(f"Error processing URL {url}: {e}")
    # Specify the Excel file path
    output_excel_file = r"C:\Users\lenovo\OneDrive\Desktop\resultbook.xlsx"

    # Write the DataFrame to an Excel file
    results_df = pd.DataFrame(results)
    results_df.to_excel(output_excel_file, index=False)

    print(f"Results have been exported to '{output_excel_file}'.")
else:
    print(f"Excel file '{excel_file_path}' not found.").
```

In conclusion, the provided Python script serves as a text analysis tool for extracting information from articles based on a list of URLs stored in an Excel file. The script employs various natural language processing (NLP) techniques and external libraries to perform analyses such as sentiment analysis, word frequency distribution, readability metrics, and more. Here are some key points about the code:

1. **Functionality:**
    - The script successfully extracts text content from articles using the **goose3** library.
    - It employs custom stopword removal to filter out irrelevant words from the text.
    - Various analyses, including sentiment analysis and frequency distribution, are performed on the cleaned text.

- Additional metrics such as subjectivity score, average sentence length, percentage of complex words, and Fog Index are calculated.

2. **Dependencies:**
    - The script relies on external libraries such as **nltk**, **textblob**, **textstat**, **goose3**, and **pandas** for natural language processing, sentiment analysis, readability metrics, web scraping, and data manipulation.
3. **Output:**
    - The results of the analyses are stored in a Pandas DataFrame and exported to an Excel file for further examination and reporting.
4. **Flexibility:**
    - The script allows for customization through the use of external files for custom stopwords, positive/negative words, and URLs from an Excel file.
    - Additional fields can be easily added to the results, providing flexibility for further analysis.
5. **Error Handling:**
    - The script includes error handling to address potential issues such as empty articles or errors during processing.
6. **Execution:**
    - To run the script successfully, users need to have the required dependencies installed, download NLTK resources, and ensure that the specified file paths are accurate.
7. **Caution:**
    - The script is dependent on the structure of the articles retrieved from the provided URLs. Changes in article structure may impact the script's behavior.

In summary, the code is a versatile text analysis tool that can be applied to multiple articles, providing insights into sentiment, word usage, and readability. Users should customize file paths and ensure data accessibility before running the script. Additionally, attention should be given to potential changes in the structure of the articles being analyzed.

Attached link of this script in .py:
https://drive.google.com/file/d/1hVXc_EFFsNiyrSLsazOUR3ePLcZc3UYv/view?usp=drive_link

Attached link of the output .xlsx:
https://docs.google.com/spreadsheets/d/1FlJpe8lTRUXPp3beLRYtLtmtbl49LTKY/edit?usp=drive_link&ouid=115307624080130468530&rtpof=true&sd=true