sentiment_analysis_blackcoffer

May 21, 2024

[3]:
```
# !pip install requests beautifulsoup4 openpyxl pandas nltk textblob
```

[27]:
```
import os
import re
import requests
import nltk
import pandas as pd
# import textstat
# from textstat.textstat import legacy_round
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from nltk.sentiment import SentimentIntensityAnalyzer
from nltk.tokenize import sent_tokenize, word_tokenize
from textstat import flesch_reading_ease, syllable_count, textstat
```

[13]:
```
# Download NLTK data files
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\HP\AppData\Roaming\nltk_data…
```

[13]:
```
True
```

[2]:
```
# Read input URLs from Excel
df = pd.read_excel('input.xlsx')
```

[3]:
```
df.head(10)
```

[3]:

	URL_ID	URL
0	blackassign0001	https://insights.blackcoffer.com/rising-it-cit…

```
1  blackassign0002  https://insights.blackcoffer.com/rising-it-cit…
2  blackassign0003  https://insights.blackcoffer.com/internet-dema…
3  blackassign0004  https://insights.blackcoffer.com/rise-of-cyber…
4  blackassign0005  https://insights.blackcoffer.com/ott-platform-…
5  blackassign0006  https://insights.blackcoffer.com/the-rise-of-t…
6  blackassign0007  https://insights.blackcoffer.com/rise-of-cyber…
7  blackassign0008  https://insights.blackcoffer.com/rise-of-inter…
8  blackassign0009  https://insights.blackcoffer.com/rise-of-cyber…
9  blackassign0010  https://insights.blackcoffer.com/rise-of-cyber…
```

[4]:
```python
# List of stopwords files to combine
files = [
    'StopWords_Auditor.txt',
    'StopWords_Currencies.txt',
    'StopWords_DatesandNumbers.txt',
    'StopWords_Generic.txt',
    'StopWords_GenericLong.txt',
    'StopWords_Geographic.txt',
    'StopWords_Names.txt',
]

# Path for the output file
output_file = 'combined_stopwords.txt'

# Create a set to hold the contents of all the files (to avoid duplicates)
all_stopwords = set()

# Iterate over the list of files
for filename in files:
    with open(filename, 'rb') as file:
        # Read the file content and add each line to the set
        for line in file:
            try:
                decoded_line = line.decode('utf-8')
                all_stopwords.add(decoded_line.strip())
            except UnicodeDecodeError:
                print(f"Error decoding line in file: {filename}")

# Write the combined stopwords to a new file
with open(output_file, 'w', encoding='utf-8') as file:
    for stopword in sorted(all_stopwords):  # Sort to keep the order consistent␣
 ↪(optional)
        file.write(stopword + '\n')

print(f'Combined stopwords written to {output_file}')
```

```
Error decoding line in file: StopWords_Currencies.txt
Error decoding line in file: StopWords_Currencies.txt
```

```
Combined stopwords written to combined_stopwords.txt
```

[5]:
```
# with open('combined_stopwords.txt', 'rb') as file:
#     for count, line in enumerate(file):
#         if count >= 10:
#             break
#         print(line.decode('utf-8'))
```

```
AARON

ABBEY

ABBIE

ABBOTT

ABBY

ABDUL

ABDULLAH

ABE

ABEL

ABELL
```

[9]:
```
# Function to read stopwords from a file
def load_stopwords(file_path):
    with open(file_path, 'r') as file:
        stop_words = set(file.read().splitlines())
    return stop_words

# Function to clean text (you can expand this with more cleaning steps as␣
 ↪needed)
def clean_text(text):
    text = text.lower()
    text = text.replace('\n', ' ')
    # Add more cleaning steps if needed
    return text

# Function to tokenize text and remove stopwords
def tokenize_and_remove_stopwords(text, stop_words):
    words = word_tokenize(text)
    filtered_text = [word for word in words if word.isalnum() and word.lower()␣
 ↪not in stop_words]
```

```
    return filtered_text

# Define the function to extract title and text from a URL
def extract_title_and_text(url):
    header = {
        'User-Agent': "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/
 ↪537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36"
    }
    try:
        response = requests.get(url, headers=header)
        response.raise_for_status()  # Raise exception for 404 or other errors
        soup = BeautifulSoup(response.content, 'html.parser')
        title = soup.find('h1').get_text()
        article = ' '.join(p.get_text() for p in soup.find_all('p'))
        return title, article
    except requests.HTTPError as e:
        print(f"Error {e.response.status_code} occurred while extracting data␣
 ↪from {url}: {e.response.reason}")
        return None, None
    except Exception as e:
        print(f"Unexpected error occurred while extracting data from {url}:␣
 ↪{e}")
        return None, None


# Load custom stopwords
stop_words = load_stopwords('combined_stopwords.txt')

# Add new columns for title and text
df['Title'] = None
df['Text'] = None
df['CleanedText'] = None
df['FilteredText'] = None

# Iterate over the DataFrame and extract title and text
for index, row in df.iterrows():
    title, text = extract_title_and_text(row['URL'])
    if title and text:
        cleaned_text = clean_text(text)
        filtered_text = ' '.join(tokenize_and_remove_stopwords(cleaned_text,␣
 ↪stop_words))
        df.at[index, 'Title'] = title
        df.at[index, 'Text'] = text
        df.at[index, 'CleanedText'] = cleaned_text
        df.at[index, 'FilteredText'] = filtered_text
```

```
Error 404 occurred while extracting data from
```

```
https://insights.blackcoffer.com/how-neural-networks-can-be-applied-in-various-
areas-in-the-future/: Not Found
Error 404 occurred while extracting data from
https://insights.blackcoffer.com/covid-19-environmental-impact-for-the-future/:
Not Found
```

[10]:
```
# Display DataFrame with extracted data
df.head(5)
```

[10]:

	URL_ID	URL	\
0	blackassign0001	https://insights.blackcoffer.com/rising-it-cit…	
1	blackassign0002	https://insights.blackcoffer.com/rising-it-cit…	
2	blackassign0003	https://insights.blackcoffer.com/internet-dema…	
3	blackassign0004	https://insights.blackcoffer.com/rise-of-cyber…	
4	blackassign0005	https://insights.blackcoffer.com/ott-platform-…	

	Title	\
0	Rising IT cities and its impact on the economy…	
1	Rising IT Cities and Their Impact on the Econo…	
2	Internet Demand’s Evolution, Communication Imp…	
3	Rise of Cybercrime and its Effect in upcoming …	
4	OTT platform and its impact on the entertainme…	

	Text	\
0	Efficient Supply Chain Assessment: Overcoming …	
1	Efficient Supply Chain Assessment: Overcoming …	
2	Efficient Supply Chain Assessment: Overcoming …	
3	Efficient Supply Chain Assessment: Overcoming …	
4	Efficient Supply Chain Assessment: Overcoming …	

	CleanedText	\
0	efficient supply chain assessment: overcoming …	
1	efficient supply chain assessment: overcoming …	
2	efficient supply chain assessment: overcoming …	
3	efficient supply chain assessment: overcoming …	
4	efficient supply chain assessment: overcoming …	

	FilteredText
0	efficient supply chain assessment overcoming t…
1	efficient supply chain assessment overcoming t…
2	efficient supply chain assessment overcoming t…
3	efficient supply chain assessment overcoming t…
4	efficient supply chain assessment overcoming t…

[15]:
```
df['CleanedText']
```

[15]:
```
0     efficient supply chain assessment: overcoming …
1     efficient supply chain assessment: overcoming …
2     efficient supply chain assessment: overcoming …
3     efficient supply chain assessment: overcoming …
4     efficient supply chain assessment: overcoming …
                          …
95    efficient supply chain assessment: overcoming …
96    efficient supply chain assessment: overcoming …
97    efficient supply chain assessment: overcoming …
98    efficient supply chain assessment: overcoming …
99    efficient supply chain assessment: overcoming …
Name: CleanedText, Length: 100, dtype: object
```

[16]:
```
df['FilteredText']
```

[16]:
```
0     efficient supply chain assessment overcoming t…
1     efficient supply chain assessment overcoming t…
2     efficient supply chain assessment overcoming t…
3     efficient supply chain assessment overcoming t…
4     efficient supply chain assessment overcoming t…
                          …
95    efficient supply chain assessment overcoming t…
96    efficient supply chain assessment overcoming t…
97    efficient supply chain assessment overcoming t…
98    efficient supply chain assessment overcoming t…
99    efficient supply chain assessment overcoming t…
Name: FilteredText, Length: 100, dtype: object
```

[30]:
```
from nltk.sentiment import SentimentIntensityAnalyzer

# Function to read positive words from file
def read_positive_words(file_path):
    with open(file_path, 'r') as file:
        positive_words = set(file.read().splitlines())
    return positive_words

# Function to read negative words from file
def read_negative_words(file_path):
    with open(file_path, 'r') as file:
        negative_words = set(file.read().splitlines())
    return negative_words

# Function to calculate sentiment scores using custom positive and negative␣
 ↪words
def calculate_sentiment_scores(text, positive_words, negative_words):
    sia = SentimentIntensityAnalyzer()
    sentiment_scores = sia.polarity_scores(text)
```

```
    # Count positive and negative words in the text
    words = text.split()
    num_positive_words = sum(word in positive_words for word in words)
    num_negative_words = sum(word in negative_words for word in words)

    # Calculate positive and negative scores based on counts
    total_words = len(words)
    positive_score = num_positive_words / total_words if total_words > 0 else 0
    negative_score = num_negative_words / total_words if total_words > 0 else 0

    # Compound score remains the same as before
    polarity_score = sentiment_scores['compound']

    # For subjectivity score, you can use the difference between positive and␣
 ↪negative scores
    subjectivity_score = abs(positive_score - negative_score)

    return positive_score, negative_score, polarity_score, subjectivity_score

# Function to count syllables in a word
def count_syllables(word):
    vowels = "aeiouy"
    count = 0
    word = word.lower().strip(".:;?!")
    if len(word) == 0:
        return 0
    if word[0] in vowels:
        count += 1
    for index in range(1, len(word)):
        if word[index] in vowels and word[index - 1] not in vowels:
            count += 1
    if word.endswith("e"):
        count -= 1
    if count == 0:
        count += 1
    return count

# Function to calculate readability metrics
def calculate_readability_metrics(text):
    words = text.split()
    sentences = text.count('.') + text.count('!') + text.count('?')
    word_count = len(words)

    # Calculate the number of complex words
    complex_words_count = sum(1 for word in words if count_syllables(word) >= 3)
```

```
    percentage_complex_words = (complex_words_count / word_count) * 100 if␣
↪word_count > 0 else 0

    # Calculate other metrics
    avg_sentence_length = word_count / sentences if sentences > 0 else 0
    fog_index = 0.4 * (avg_sentence_length + percentage_complex_words)
    avg_words_per_sentence = word_count / sentences if sentences > 0 else 0
    avg_syllables_per_word = sum(count_syllables(word) for word in words) /␣
↪word_count if word_count > 0 else 0
    personal_pronouns = sum(word.lower() in ['i', 'we', 'my', 'ours', 'us'] for␣
↪word in words)
    avg_word_length = sum(len(word) for word in words) / word_count if␣
↪word_count > 0 else 0

    return (
        percentage_complex_words, avg_sentence_length, fog_index,
        avg_words_per_sentence, complex_words_count, word_count,
        avg_syllables_per_word, personal_pronouns, avg_word_length
    )

def main(df, positive_words, negative_words):
    for index, row in df.iterrows():
        text = row['Text']  # Adjust according to your DataFrame column name
        if text:  # Check if text is not None or empty
            positive_score, negative_score, polarity_score, subjectivity_score␣
↪= calculate_sentiment_scores(text, positive_words, negative_words)
            metrics = calculate_readability_metrics(text)

            # Unpack metrics
            (percentage_complex_words, avg_sentence_length, fog_index,␣
↪avg_words_per_sentence,
             complex_word_count, word_count, avg_syllables_per_word,␣
↪personal_pronouns, avg_word_length) = metrics

            # Update DataFrame with sentiment and readability scores
            df.at[index, 'PositiveScore'] = positive_score
            df.at[index, 'NegativeScore'] = negative_score
            df.at[index, 'PolarityScore'] = polarity_score
            df.at[index, 'SubjectivityScore'] = subjectivity_score
            df.at[index, 'PercentageComplexWords'] = percentage_complex_words
            df.at[index, 'AvgSentenceLength'] = avg_sentence_length
            df.at[index, 'FogIndex'] = fog_index
            df.at[index, 'AvgWordsPerSentence'] = avg_words_per_sentence
            df.at[index, 'ComplexWordCount'] = complex_word_count
            df.at[index, 'WordCount'] = word_count
            df.at[index, 'AvgSyllablesPerWord'] = avg_syllables_per_word
```

```
            df.at[index, 'PersonalPronouns'] = personal_pronouns
            df.at[index, 'AvgWordLength'] = avg_word_length

# Call main
positive_words = read_positive_words('positive-words.txt')
negative_words = read_negative_words('negative-words.txt')
main(df, positive_words, negative_words)
```

[32]:
```
df.head(2)
```

[32]:
```
           URL_ID                                                URL  \
0  blackassign0001  https://insights.blackcoffer.com/rising-it-cit…
1  blackassign0002  https://insights.blackcoffer.com/rising-it-cit…

                                              Title  \
0  Rising IT cities and its impact on the economy…
1  Rising IT Cities and Their Impact on the Econo…

                                               Text  \
0  Efficient Supply Chain Assessment: Overcoming …
1  Efficient Supply Chain Assessment: Overcoming …

                                         CleanedText  \
0  efficient supply chain assessment: overcoming …
1  efficient supply chain assessment: overcoming …

                                       FilteredText PositiveScore  \
0  efficient supply chain assessment overcoming t…      0.018797
1  efficient supply chain assessment overcoming t…      0.033312

  NegativeScore PolarityScore SubjectivityScore  … avg_word_length  \
0       0.00188        0.9957          0.016917  …             None
1      0.014456        0.9995          0.018856  …             None

  PercentageComplexWords AvgSentenceLength   FogIndex AvgWordsPerSentence  \
0              21.804511         17.733333  15.815138           17.733333
1              25.204274         18.940476  17.657900           18.940476

  ComplexWordCount WordCount AvgSyllablesPerWord PersonalPronouns  \
0            116.0     532.0            1.808271              4.0
1            401.0    1591.0            1.923947              5.0

   AvgWordLength
0       5.389098
1       5.759899

[2 rows x 28 columns]
```

[34]:
```
output= df[['URL_ID', 'URL','PositiveScore', 'NegativeScore', 'PolarityScore',␣
↪'SubjectivityScore',
      'avg_sentence_length', 'percentage_complex_words', 'fog_index',
      'avg_words_per_sentence', 'complex_word_count', 'word_count',
      'avg_syllables_per_word', 'personal_pronouns',␣
↪'avg_word_length','AvgSyllablesPerWord', 'PersonalPronouns',␣
↪'AvgWordLength']]
```

[35]:
```
output.head(2)
```

[35]:
```
           URL_ID                                                URL  \
0  blackassign0001  https://insights.blackcoffer.com/rising-it-cit…
1  blackassign0002  https://insights.blackcoffer.com/rising-it-cit…

  PositiveScore NegativeScore PolarityScore SubjectivityScore  \
0      0.018797       0.00188        0.9957          0.016917
1      0.033312      0.014456        0.9995          0.018856

  avg_sentence_length percentage_complex_words fog_index  \
0                None                     None      None
1                None                     None      None

  avg_words_per_sentence complex_word_count word_count avg_syllables_per_word  \
0                   None               None       None                    None
1                   None               None       None                    None

  personal_pronouns avg_word_length  AvgSyllablesPerWord  PersonalPronouns  \
0              None            None             1.808271               4.0
1              None            None             1.923947               5.0

   AvgWordLength
0       5.389098
1       5.759899
```

[36]:
```
output.columns
```

[36]:
```
Index(['URL_ID', 'URL', 'PositiveScore', 'NegativeScore', 'PolarityScore',
       'SubjectivityScore', 'avg_sentence_length', 'percentage_complex_words',
       'fog_index', 'avg_words_per_sentence', 'complex_word_count',
       'word_count', 'avg_syllables_per_word', 'personal_pronouns',
       'avg_word_length', 'AvgSyllablesPerWord', 'PersonalPronouns',
       'AvgWordLength'],
      dtype='object')
```

[37]:
```
output.to_csv('output.csv', index=False)
```

[ ]:
```

```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[4]:
```
def extract_title_and_text(url):
    header = {
        'User-Agent': "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/
 ↪537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36"
    }
    try:
        response = requests.get(url, headers=header)
        response.raise_for_status()  # Raise exception for 404 or other errors
        soup = BeautifulSoup(response.content, 'html.parser')
        title = soup.find('h1').get_text()
```

```
        article = ' '.join(p.get_text() for p in soup.find_all('p'))
        return title, article
    except Exception as e:
        print(f"Error occurred while extracting data from {url}: {e}")
        return None, None

def clean_and_tokenize(text):
    stop_words = set(stopwords.words('english'))
    words = word_tokenize(text)
    filtered_text = [word for word in words if word.lower() not in stop_words]
    return filtered_text

def calculate_sentiment_scores(docs, pos_words, neg_words):
    positive_scores = []
    negative_scores = []
    for doc in docs:
        positive_words = [word for word in doc if word.lower() in pos_words]
        negative_words = [word for word in doc if word.lower() in neg_words]
        positive_scores.append(len(positive_words))
        negative_scores.append(len(negative_words))
    return positive_scores, negative_scores

def measure_text_features(text_dir):
    avg_sentence_lengths = []
    percentages_of_complex_words = []
    fog_indices = []
    complex_word_counts = []
    avg_syllable_word_counts = []
    stopwords_set = set(stopwords.words('english'))

    def measure(file_path):
        with open(file_path, 'r') as f:
            text = f.read()
            text = re.sub(r'[^\w\s.]', '', text)
            sentences = text.split('.')
            num_sentences = len(sentences)
            words = [word for word in text.split() if word.lower() not in␣
 ↪stopwords_set]
            num_words = len(words)
            complex_words = [word for word in words if len(word) > 2]
            num_complex_words = len(complex_words)
            avg_sentence_length = num_words / num_sentences
            avg_syllable_word_count = sum(len(re.findall(r'[aeiouy]+', word.
 ↪lower())) / max(1, len(word)) for word in words) / num_words
            percentage_of_complex_words = num_complex_words / num_words
            fog_index = 0.4 * (avg_sentence_length +␣
 ↪percentage_of_complex_words)
```

```
            return avg_sentence_length, percentage_of_complex_words, fog_index,␣
 ↪num_complex_words, avg_syllable_word_count

    for file_name in os.listdir(text_dir):
        file_path = os.path.join(text_dir, file_name)
        if os.path.isfile(file_path):
            x, y, z, a, b = measure(file_path)
            avg_sentence_lengths.append(x)
            percentages_of_complex_words.append(y)
            fog_indices.append(z)
            complex_word_counts.append(a)
            avg_syllable_word_counts.append(b)

    return avg_sentence_lengths, percentages_of_complex_words, fog_indices,␣
 ↪complex_word_counts, avg_syllable_word_counts

def count_personal_pronouns(text_dir):
    personal_pronouns = set(["i", "we", "my", "ours", "us"])
    pp_counts = []

    def count_pronouns(text):
        return sum(1 for word in text.split() if word.lower() in␣
 ↪personal_pronouns)

    for file_name in os.listdir(text_dir):
        file_path = os.path.join(text_dir, file_name)
        if os.path.isfile(file_path):
            with open(file_path, 'r') as f:
                text = f.read()
                pp_count = count_pronouns(text)
                pp_counts.append(pp_count)

    return pp_counts

def main():
    # Directories
    text_dir = "/gdrive/MyDrive/project/Data_Extraction_and_NLP/TestAssignment/
 ↪TitleText"
    stopwords_dir = "/gdrive/MyDrive/project/Data_Extraction_and_NLP/
 ↪TestAssignment/StopWords"
    sentiment_dir = "/gdrive/MyDrive/project/Data_Extraction_and_NLP/
 ↪TestAssignment/MasterDictionary"

    # Extracting data
    df = pd.read_csv("your_input_data.csv")
    urls = df['URL'].tolist()
```

```
    docs = []

    for url in urls:
        title, text = extract_title_and_text(url)
        if text:
            file_name = os.path.join(text_dir, f"{url}.txt")
            with open(file_name, 'w') as f:
                f.write(title + '\n' + text)
            cleaned_text = clean_and_tokenize(text)
            docs.append(cleaned_text)

    # Sentiment analysis
    pos_words = set()
    neg_words = set()

    for file_name in os.listdir(sentiment_dir):
        with open(os.path.join(sentiment_dir, file_name), 'r',␣
↪encoding='ISO-8859-1') as f:
            if file_name == 'positive-words.txt':
                pos_words.update(f.read().splitlines())
            else:
                neg_words.update(f.read().splitlines())

    positive_scores, negative_scores = calculate_sentiment_scores(docs,␣
↪pos_words, neg_words)

    # Measuring text features
    avg_sentence_lengths, percentages_of_complex_words, fog_indices,␣
↪complex_word_counts, avg_syllable_word_counts =␣
↪measure_text_features(text_dir)

    # Counting personal pronouns
    pp_counts = count_personal_pronouns(text_dir)

    # Creating output DataFrame
    output_df = pd.read_excel('Output Data Structure.xlsx')
    output_df.drop([44 - 37, 57 - 37, 144 - 37], axis=0, inplace=True)

    output_df['Positive_Score'] = positive_scores
    output_df['Negative_Score'] = negative_scores
    output_df['Polarity_Score'] = (output_df['Positive_Score'] -␣
↪output_df['Negative_Score']) / (output_df['Positive_Score'] +␣
↪output_df['Negative_Score'] + 0.000001)
    output_df['Subjectivity_Score'] = (output_df['Positive_Score'] +␣
↪output_df['Negative_Score']) / (output_df['Total_Words'] + 0.000001)
    output_df['Avg_Sentence_Length'] = avg_sentence_lengths
    output_df['Percentage_of_Complex_Words'] = percentages_of_complex_words
```

```
    output_df['Fog_Index'] = fog_indices
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[4], line 34
     32   #write title and text to the file
     33   file_name = '/gdrive/MyDrive/project/Data_Extraction_and_NLP/
 ↪TestAssignment/TitleText/' + str(url_id) + '.txt'
---> 34   with open(file_name, 'w') as file:
     35     file.write(title + '\n' + article)
     37 # Directories

File ~\Anaconda\Lib\site-packages\IPython\core\interactiveshell.py:310, in␣
 ↪_modified_open(file, *args, **kwargs)
    303 if file in {0, 1, 2}:
    304     raise ValueError(
    305         f"IPython won't let you open fd={file} by default "
    306         "as it is likely to crash IPython. If you know what you are␣
 ↪doing, "
    307         "you can use builtins' open."
    308     )
--> 310 return io_open(file, *args, **kwargs)

FileNotFoundError: [Errno 2] No such file or directory: '/gdrive/MyDrive/projec /
 ↪Data_Extraction_and_NLP/TestAssignment/TitleText/blackassign0001.txt'
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: