Date: 16th April, 2022

Blackcoffer Consulting

Internship Assignment Report

Siddhartha Roy

M.Sc in Data Science

Christ (Deemed-to-be) University, Bangalore.

Table of Contents

Data Extraction and Text Analysis

Objective

The objective of the assignment is to extract textual data from the provided URL and perform text analysis on the obtained textual data.

Data Extraction

Data Extraction is an important step, probably the primary important step, in any data science process. Data extraction is the process of obtaining data from a database platform so that it can be replicated to a destination — such as a data warehouse — designed to support online analytical processing (OLAP). Data extraction is the first step in a data ingestion process called ETL — extract, transform, and load.
In the given assignment, we have used *BeauitfulSoup*, a very flexible web scraping library in Python, to get the text data from the given data of URL objects in the *“Input.xlsx”* file.
We have also used the *requests* library to get access to the HTML of the URLs given.
Finally, we have stored the obtained textual data in a pandas DataFrame and written that DataFrame to a .csv (comma separated value) file, which we have used as our data set for the textual analysis.
The code and algorithm can be found in the attached *“Blackcoffer - Web Scraping Data-BeautifulSoup4.ipynb”* file.

Data Manipulation

All the operations that have been described below are implemented in the *“Blackcoffer - Text-Analysis.ipynb”* file attached herewith.

Data Cleaning using StopWords

We have used the English stop words dictionary from the NLTK library in Python. It is used to filter out frequently occurring, non-informative words such as “a”, “the”, “is”, “are” etc.

Expanding contractions

The *contractions* library in Python has been used to expand contractions in text such as “don’t” and “ain’t”, which become “do not” and “are not”, respectively.

Tokenize

Tokenization is essentially splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms. Each of these smaller units are called tokens. The tokens could be words, numbers or punctuation marks. We have used two modules in Python called “word_tokenize” and “sent_tokenize” from the NLTK library used to tokenize words and sentences, respectively. Alternatively, one way to tokenize words is to use the RegexpTokenize module, which helps you to tokenize words based on a Regular Expression pattern.

Lemmatize

Lemmatization is a sophisticated version of stemming words in textual data. It is used to reduce a word to its proper base form, that is, a word that can be found in the dictionary.
It can be performed in Python using the *WordNetLemmatizer* module found in the NLTK library. In our text analysis, we have performed lemmatizing on each word token in the text body to reduce it to its base form.

The above text data cleaning methodologies have been compiled into a single function which we have applied on the entire data frame of textual data.

Sentiment Analysis

The Opinion Lexicon (or Sentiment Lexicon) Positive and Negative dictionaries have been used to find the Positive and Negative scores of the words in our text data.

Positive and Negative Score

The Positive and Negative scores are found by adding +1 to either the Positive or Negative columns if the respective word is found in the Positive and Negative Opinion lexicon.

Polarity Score

Sentiment polarity for an element **defines the orientation of the expressed sentiment**, i.e., it determines if the text expresses the positive, negative or neutral sentiment of the user about the entity in consideration. It is a float variable between -1 and 1.

Polarity Score = (Positive Score – Negative Score)/ ((Positive Score + Negative Score) + 0.000001)

Subjectivity Score

Subjective sentences generally refer to personal opinion, emotion or judgment whereas objective refers to factual information. Subjectivity is also a float which lies in the range of 0 and 1.

Subjectivity Score = (Positive Score + Negative Score)/ ((Total Words after cleaning) + 0.000001)

Analysis of Readability Index

Analysis of Readability Index is defined using the Gunning Fog Index.

Gunning Fog Index

The Gunning Fog Index estimates the years of formal education a person needs to understand the text on the first reading. For instance, a fog index of 12 requires the reading level of a United States high school senior (around 18 years old). Ideal Gunning Fog Index of any body of text is assumed to be around 7 to 8. Anything above 12 is perceived to be too complicated to read for an average reader.

Fog Index = 0.4 * (Average Sentence Length + Percentage of Complex words)

Complex Words

Complex words are words in a text that contain more than two syllables. For the purpose of calculating the percentage of complex words in the given text, we have taken the help of the *textstatistics* library in Python. We have counted the number of syllables in the words using the *syllable_count()* function.

Average Sentence Length

Average Sentence Length = the number of words / the number of sentences

Word Count

Word count in a body of text can be found by:

- removing stop words such as “a”, “the”, “is”, “are” etc using *nltk.corpus.stopwords*
- removing punctuations from the words using the string.punctuations module.

Then, we have tokenized the words into tokens and found out the length of the obtained list of tokens.

Average Number of Words per Sentence

We have used *nltk.sent_tokenize* to tokenize sentences and *nltk.word_tokenize* to tokenize words. Now, average number of words per sentence can be found out using the formula:

Average Number of Words Per Sentence = the total number of words / the total number of sentences.

Syllable Count per Word

We count the number of Syllables in each word of the text by counting the vowels present in each word. We also handle some exceptions like words ending with "es","ed" by not counting them as a syllable.

Syllables can be counted using the *textstatistics.syllable_count()* function. We have used the *.apply()* and *lambda* function to find out the number of syllables in each other in a body of text and store the information in a separate df column.

Personal Pronouns

To calculate Personal Pronouns mentioned in the text, we use regular expressions to find the counts of the words - "I," "we," "my," "ours," and "us". Special care is taken so that the country name US is not included in the list.

We have defined the following function to find the length of the list of personal pronouns in the given body of text:

```
def personal_pro(text):
 pronounRegex = re.compile(r'\b(I|we|my|ours|(?-i:us))\b',re.I)
 pronouns = pronounRegex.findall(text)
 return len(pronouns)
```

`(?-i:us)` is used as an in-line modifier group where the matching is case-sensitive. As a result, this matches only us and not US (the country).

Average Word Length

Average word length can be found by the following formula:

(Sum of the total number of characters in each word) / (Total number of words)

We have therefore defined a function called *text_len()* to find the number of alphabetical characters in each word and divide them by the number of words in the body of text. This yields the average length of each word in the text data.

```
def text_len(text):
 text = ''.join(text)
 filtered = ''.join(filter(lambda x: x not in string.punctuation, text))
 words = [word for word in filtered.split() if word]
 avg = sum(map(len, words))/len(words)
 return avg
```

Conclusions

In conclusion, we have written the finalized dataframe in the required format by the company into a .csv output file. From the analysis we have drawn, many insights can be drawn about the given text that are not usually derived.