Python Text Analysis Models.

Natural Language Processing Project Report
Submitted to

DEVI AHILYA VISHWAVIDHYALYA INDORE

In Partial Fulfillment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

in

BIG DATA ANALYTIC

by

Er. Manish Kumar Singh
DS7B-2208

Under the Supervision of

Mr. Puneet Gupta
Assistant Professor

SCHOOL OF DATA SCIENCE AND FORECASTING

2022-23

Python Text Analysis Models.

May 5, 2023

```
[ ]: from flask import Flask, request, render_template
```

This line imports the Flask web framework, which is a popular Python library for building web applications. It also imports the `request` and `render_template` modules, which are used to handle HTTP requests and to render HTML templates, respectively

```
[ ]: from sklearn.feature_extraction.text import TfidfVectorizerfrom
     sklearn.metrics.pairwise import cosine_similarity
```

These lines import two modules from scikit-learn, a machine learning library for Python. `TfidfVectorizer` is used to transform text data into a matrix of TF-IDF features, which can then be used for clustering or classification tasks. `cosine_similarity` is used to calculate the cosine similarity between vectors, which is a common measure of similarity used in text analysis.

```
[ ]: from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
```

This line imports the `SentimentIntensityAnalyzer` class from the `vaderSentiment` library, which is a Python package for sentiment analysis. This class is used to compute sentiment scores for text data, which can be used to determine whether a piece of text is positive, negative, or neutral.

```
[ ]: import nltk
```

This line imports the `nltk` library, which is a popular Python library for natural language processing.

```
[ ]: from string import punctuation
     import re
     from nltk.corpus import stopwords
```

These lines import the `punctuation` variable from the Python `string` module, which contains a set of common punctuation characters. They also import the `re` module, which provides regular expression matching operations. Finally, they import the `stopwords` module from the `nltk.corpus` package, which contains a set of stopwords for various Languages.

```
[ ]: nltk.download('stopwords')
```

This line downloads the set of stopwords for the English language from the `nltk` library. Once downloaded, the stopwords can be used to filter out common words that do not carry much meaning when analyzing text.

```
[ ]: set(stopwords.words('english'))
```

This line creates a set of stopwords for the English language using the stopwords module from the nltk library. This set will be used later to filter out common words that do not carry much meaning when analyzing text.

```
[ ]: app = Flask(__name__)
```

This line creates a new Flask application instance with the name of the current module.

```
[ ]: @app.route('/')
     def my_form():
         return render_template('form.html')
```

This function sets up a route for the root URL of the web application. When the user visits the root URL, the my_form() function is called, which renders a HTML form called form.html that prompts the user to enter some text.

```
[ ]: @app.route('/', methods=['POST'])
     def my_form_post():
```

This function sets up a route for HTTP POST requests to the root URL of the web application. When the user submits the form via the "Submit" button, the my_form_post() function is called to process the text entered by the user.

```
[ ]: stop_words = stopwords.words('english')
```

This line creates a list of stopwords for the English language using the stopwords module from the nltk library.

```
[ ]: text1 = request.form['text1'].lower()
```

This line retrieves the text entered by the user in the form, converts it to lowercase, and stores it in the text1 variable.

```
[ ]: text_final = ''.join(c for c in text1 if not c.isdigit())
```

This line removes all digits from the user's input text by iterating through each character c in text1 and appending it to a new string text_final only if it is not a digit.

```
[ ]: processed_doc1 = ' '.join([word for word in text_final.split() if word not in␣
      ↪stop_words])
```

This line tokenizes the cleaned text in text_final into a list of words, and then filters out any stopwords from that list using a list comprehension. The filtered words are then joined back into a string using spaces as separators, and stored in the processed_doc1 variable.

```
[ ]: sa = SentimentIntensityAnalyzer()
     dd = sa.polarity_scores(text=processed_doc1)
     compound = round((1 + dd['compound'])/2, 2)
```

These lines use the SentimentIntensityAnalyzer class from the vaderSentiment library to compute the sentiment scores for the processed text in processed_doc1. The sentiment scores are returned as a dictionary of four values: neg, neu, pos, and compound. The compound score is a normalized score between -1 and 1 that represents the overall sentiment of the text, where values closer to -1 indicate negative sentiment and values closer to 1 indicate positive sentiment. The `round()` function is used to round the `compound` score to two decimal places.

```
[ ]: return render_template('form.html', final=compound,
                            text1=text_final,text2=dd['pos'],
                            text5=dd['neg'],text4=compound,text3=dd['neu'])
```

This line renders the `form.html` template with the `compound` score, the cleaned text (`text_final`), and the individual sentiment scores (`pos`, `neg`, and `neu`) passed as arguments to the template. The template will display the sentiment analysis results to the user.

```
[ ]: if __name__ == "__main__":
         app.run(debug=True, host="127.0.0.1", port=5002, threaded=True)
```

This line of code starts the Flask web application in debug mode on the local machine at IP address 127.0.0.1 (localhost) and port 5002. The debug parameter is set to True to enable debug mode, which displays detailed error messages if an error occurs. The threaded parameter is set to True to allow the application to handle multiple requests at the same time by creating a new thread for each request.

This is an HTML template file used in a Flask web application for sentiment analysis.

- `<html>`: Indicates the start of an HTML document.
- `<head>`: Defines the header section of the HTML document, which contains information about the document, such as its title, styles, and metadata.
- `<style>`: Defines the style rules for HTML elements.
- `table, th, td`: Defines a CSS selector that targets all `table`, `th`, and `td` elements in the HTML document.
- `{% if final %}`: A Jinja template tag that checks if the variable `final` is defined and not None.
- `<h1>`: Defines a heading level 1.
- `<form>`: Defines a form for user input.
- `<textarea>`: Defines a multi-line input field for text input.
- `<input>`: Defines a single-line input field for submitting the form.
- `{% else %}`: A Jinja template tag that handles the case where the `final` variable is not defined or None.
- `<div>`: Defines a container for HTML elements.
- `<h2>`: Defines a heading level 2.
- `<table>`: Defines an HTML table.
- `<tr>`: Defines a table row.
- `<th>`: Defines a table header cell.
- `<td>`: Defines a table data cell.
- `{{ }}`: A Jinja template expression that outputs the value of a variable or expression.
- `</table>`: Indicates the end of an HTML table.
- `</div>`: Indicates the end of a container element.
- `</body>`: Indicates the end of the body section of the HTML document.
- `</html>`: Indicates the end of the HTML document.