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

Test Assignment

Submitted by Shobhit Kumar Singh

System Setup:-

Environment Setup:-

I utilised Conda to establish a Python 3.11 environment, ensuring compatibility with the project's requirements. For other dependencies, I maintained the requirements.txt file and used pip to install all of them at once at the beginning.

Configuration and Dependencies:

The initial steps involved the creation of two essential files—**config.yaml** and **requirements.txt**. The former serves as the configuration file, capturing parameters for the project, while the latter lists necessary dependencies.

File Organisation:

As the assignment was of a single stage, I placed all project components in the root directory rather than creating different directories.

1. **Data directory:** For organising and managing various data files provided in the assignment, facilitating easy access.
2. **Utility Functions (utility_functions.py):** The **utility_functions** directory houses frequently used functions that streamline common operations across the project, contributing to code modularity and reusability.
3. **Preprocessing (preprocessing.py):** The **preprocessing.py** file is dedicated to Natural Language Processing (NLP)-specific data cleaning and preprocessing tasks. It encapsulates all the necessary steps to ensure the quality and relevance of textual data.
4. **Sentiment Scores (sentiment_scores.py):** In **sentiment_scores.py**, positive and negative sentiment scores are calculated. This file encapsulates the logic behind evaluating sentiment based on predefined dictionaries.
5. **Custom Parameters (custom_parameters.py): custom_parameters.py** focuses on processing the parameters requested for the assignment. It acts as a central hub for handling and transforming the various parameters involved in the project.
6. **Main Execution (main.py):** The **main.py** file acts as the project's central execution point. It orchestrates the entire process, from web scraping to calling the functions defined in **preprocessing.py**, **sentiment_scores.py**, and **custom_parameters.py**.
7. **Dependencies (requirements.txt):** The **requirements.txt** file specifies the external dependencies essential for the project. It simplifies the installation process by listing the required libraries.
8. **Configuration File (config.yaml): config.yaml** captures project-specific configurations, allowing for easy adjustment of parameters without modifying the source code. This file plays a crucial role in customising the behaviour of the project.

Approached:-

1. Preprocessing stage

- **Step 0: Count “US”:** Count occurrences of the substring "US" in the input text. It will be utilised in counting personal pronouns because after lower-casing the sentences the string “US” will be treated as a personal pronoun “us”
- **Step 1: Text Cleaning:** Convert text to lowercase and remove non-alphanumeric characters.
- **Step 2: Tokenization:** Break down the text into individual words using tokenisation. The word_tokenize function is likely imported from the NLTK library.
- **Step 3: Count Personal Pronouns:** Count occurrences of personal pronouns in the tokenised words.
- **Step 4: Stopword Removal:** Remove common stopwords from the tokenized words using the NLTK library's stopwords module. Otherwise Load and include custom stopwords from a specified folder.
- **Step 5: Lemmatization:** Reduce words to their base form using lemmatization. The WordNet lemmatizer from the NLTK library is likely used here.
- **Step 6: Part-of-Speech (POS) Tagging using spaCy:** Use spaCy, a natural language processing library, to perform part-of-speech tagging on the lemmatized words. The en_core_web_sm model is loaded. (I was planning to use POS for personal pronoun counting and sentiment analysis but in the instruction, it was mentioned to calculate the score only)

2. Sentiment Score calculation

I define a function called sentiment_scores that calculates sentiment scores for a list of words. To handle configuration files(Directory of Positive and Negative scores) and custom functions (load_config and load_dictionary) from the utility_functions module, I access config.yaml file.

The function sentiment_score has two parameters: word_list (a list of words) and config_path (an optional path to a configuration file, with a default value of "config.yaml"). The function should return a tuple containing positive and negative sentiment scores.

3. Custom Parameter calculation

- **Import the necessary libraries:** including **logging** for logging messages, **yaml** for handling configuration files, **sent_tokenize** from NLTK for sentence tokenization, **syllables** for estimating syllable count, and various functions (**preprocess**, **load_config**, **load_dictionary**, **sentiment_scores**) from custom modules.
- **Load Configuration:** Load the configuration from the specified YAML file (config.yaml) and log messages accordingly.
- **Load Sentiment Dictionaries:** Extract paths for positive and negative sentiment dictionaries from the configuration.

- **Tokenize Sentences:** Tokenize the input text into sentences using NLTK's sent_tokenize function.
- **Initialize Variables:** Initialize variables to keep track of various metrics, including total words, total complex words, total syllables, positive and negative sentiment scores, personal pronouns count, and character count.
- **Process Sentences:**
    1. Iterate through each sentence, preprocess it using the **preprocess** function, and calculate sentiment scores using the sentiment_scores function.
    2. Update the cumulative counts and scores for further calculations.
- **Calculate Custom Parameters:** Calculate various custom parameters based on the accumulated counts and scores, such as average sentence length, percentage of complex words, FOG index, polarity score, subjectivity score, average word length, and more.
- **Logging:** Log messages at different stages of the calculation to indicate successful completion.
- **Return:** Return a dictionary containing the calculated custom parameters.

4. Main.py

- **Import Libraries:** Import necessary libraries including requests for making HTTP requests, BeautifulSoup for web scraping, load_config from utility_functions for loading configurations, calculate_metrics from custom_parameters for calculating custom parameters, logging for logging messages, and pandas for working with DataFrames.
- **Set up Logging Configuration:** Set up logging configuration with INFO level.
- **Function Definition: process_url:** Define a function process_url that processes a single URL and updates the results DataFrame in real-time. It takes various parameters such as url_id, url, headers, parser, element, class_name, strip, and results_df (optional DataFrame to update with results). It returns the updated DataFrame.
- **Webpage Request and Text Extraction:** Make a request to the webpage using requests.get() and check if the status code is not 404. If successful, extract the text content from a specific HTML element using BeautifulSoup.
- **Calculate Custom Parameters:** Calculate custom parameters using the calculate_metrics function.
- **Update Results DataFrame:** If the results_df is not provided, create an empty DataFrame. Add the results to the DataFrame and save the updated DataFrame to a new Excel file (realtime_sheet.xlsx).
- **Handle Failure:** If the webpage request fails (404 status code), log a warning and handle the failure by marking the entire row with "ERROR 404" in the results DataFrame.
- **Function Definition: main:** Define the main function that loads the main configuration from config.yaml, loads web scraping configuration, reads URLs from an Excel sheet, creates an empty DataFrame for results, and processes each URL using the process_url function.
- **Load Configuration:** Load the main configuration and web scraping configuration from config.yaml.

- **Read URLs from Excel:** Read URLs from an Excel sheet using pandas.
- **Process URLs:** Process each URL using the process_url function and update the results DataFrame in real-time.
- **Save Results:** Save the final results DataFrame to a new Excel file (output.xlsx).
- **Main Execution:** If the script is executed as the main program, call the main() function.