EN.500.113 Gateway Computing: Python

Project B: Sentiment Analysis

Project Due: Friday, October 11 at 23:59 ET

Introduction

Sentiment analysis is the process of identifying positive, negative or neutral opinions from text to understand the social sentiment of a brand, product or service while monitoring conversations. The advances in artificial intelligence and algorithms are making it possible to analyze text, audio, and videos, in a very efficient way. Examples of expressions that can be classified as positive, negative or neutral in text are:

Phrase	**Classification**
I really love the new design!	positive
That was the most horrible customer service ever.	negative
I am not very sure about the solution to my complaint.	neutral

This project aims to create a sentiment analysis tool tailored for analyzing user Twitter reviews of airlines. The current examples come from *Twitter US Airline Sentiment Analyze* - Kaggle[1]. It shows how travelers conveyed their sentiments on Twitter in February 2015. For the sake of this task, we will only use a portion of the dataset; we will focus on a subset of 905 opinions given to United Airlines. Your tool will classify sentiment, visualize the scores and extract key aspects that customers mention as positive or negative.

[1] https://appen.com/pre-labeled-datasets/

Tasks

You have been provided with a template script named `ProjectB.py`, which includes template functions that require completion. This code contains the following:

- Examples of reviews:
  - A `list` of test reviews.
  - A `list` of reviews from Twitter
- Positive keywords: a `list` of positive keywords for this particular task and dataset
- Negative keywords: a `list` of negative keywords for this particular task and dataset
- Unwanted characters: a `string` of characters that should be removed
- Stop words: a `list` of words that do not contribute additional information when the phrases are analyzed
- Placeholders where to call your functions

An extra module named `extras.py` is also provided. It contains two functions. The `pie_plot` function will help you visualize the overall sentiment of your data and `reading_csv_tweet` will read the csv Twitter file. You should **not modify** this file. Both files, `extras.py` and `ProjectB.py` should be placed in the same folder on your computer.

`extras.py` requires Matplotlib package to be installed. Please refer to ProjectA for instructions.

1 Preprocess the input

Your first task is to preprocess the data. To preprocess means to clean the data from unwanted elements that give no information or that can bias your output. To do this, we need six functions: `lower_split`, `unwanted_characters_filter`, `unwanted_numbers`, `unwanted_words`, `concatenating_clean_words` and `preprocess_text`.

1.1 Converting to lowercase: `lower_split` (2.5 points)

This function converts a string to lowercase. The function `lower_split` accepts one parameter (`str`), which is a phrase to be analyzed.

The `return` is a list of all the words in lowercase. For example,

```
print(lower_split('Hello World'))
```

will print `['hello', 'world']`.

1.2 Filtering unwanted characters: `unwanted_characters_filter` (5 points)

Unwanted characters are characters that, although they emphasize the statements, should be discarded as they don't provide any extra meaning to the phrases. The full list of these characters is as follows [2]:

```
!"#$%&\'()*+,-./:;<=>?@[]^_`{|}~
```

This function will then receive two parameters:

- word: a `string` of the words to clean
- unwanted_characters: a `string` with all the unwanted characters

The function will `return` a `str` with the clean word or words.
For example,

```
print(unwanted_characters_filter("Fantastic work!!???",
      unwanted_characters))
```

will print `Fantastic work`.

Hint: remember that there are special characters on the list; you should find a way how to handle them.

1.3 Removing all numbers: `unwanted_numbers` (5 points)

In the same way, your code should remove all the numbers in the phrases as they (most of the time) refer to the time, flight, etc.

The input to this function is a `str` that contains a phrase with or without numbers. The function will `return` a clean version of the input without numbers.

A call of such a function is:

```
print(unwanted_numbers('flight 2423 delayed 45min'))
```

The output will show `flight delayed min`.

1.4 Removing unwanted words: `unwanted_words` (5 points)

Similarly, there is a set of words called *stop words*. They are words that are commonly used in any language. The reason to remove them is to focus on the critical aspects of the users' opinions. Examples of stop words are prepositions (under, below, etc), determiners (the, a, an) and coordinating conjunctions (or, and, but), among others.

For the special case of our task, we will use a modified version of the Minimal stop word (`https://github.com/kavgan/stop-words/blob/master/minimal-stop.txt`).

Hence, this function will remove such words. The parameters of the function are:

[2] The list can be more extensive, but for this project, we will only focus on the basic ones.

- text: a `str` with a phrase
- stop_words: a `list` of stop words tailored for this task (This list is provided.)

The `return` of the function is a `str` with the clean version of the phrase.
For example,

```
print(unwanted_words('I want a new flight',stop_words))
```

will output: `want new flight`.

1.5 Concatenating clean words: `concatenating_clean_words` (5 points)

This is an extra function that will concatenate the clean words from a `list`. This function accepts two parameters:

- tokens: a `list` of words.
- unwanted_characters: a `str` that contains all unwanted characters.

The `return` of the function is a `str` without all the unwanted characters.
For example,

```
print(concatenating_clean_words(['fantastic','work!!!'],
      unwanted_characters))
```

will output: `fantastic work`.

Hint: Use `unwanted_characters_filter` to filter out the unwanted characters per word.

1.6 Preprocess text: `preprocess_text` (2.5 points)

This part of the code will put all previous functions together one by one, as it "pre-processes" (or cleans) all the data received from the customers. The function should:

1. Convert the input text to lowercase
2. Remove the unwanted characters from the output of (1).
3. Remove stop words from the output of (2).
4. Remove any numbers from the output of (3).
5. Return the filtered text.

The function accepts three parameters:

- Text: `str` of text to be cleaned
- unwanted_characters: a `str` of unwanted characters.
- stop words: a `list` of words that are not useful for the analysis

This function returns the cleaned text as a `str`. An example of how to run the function is:

```
print(preprocess_text("I was moved to first class!!!
    Flight 2345 best experience!",
    unwanted_characters, stop_words))
```

The output will print: `moved first class flight best experience`.

2 Lexicon-based sentiment analysis implementation

In assessing customer sentiment, the code places emphasis on customer feedback. It employs two distinct sets of words, one for positive expressions and another for negative expressions, depending on the task and customer inquiries. With this information, the system can grade the degree of satisfaction. Examples of positive words are *good, fantastic, great, and excellent.* Examples of negative expressions are: *terrible, bad, awful, sad.*

We divide this sub-task into "word search" and "analyze sentiment" described in the next sections.

2.1 Word search: `word_search` (5 points)

The first task is to implement a generic word search for the positive and negative words. This function receives:

- keywords: a `list` of words to search
- words: a `list` of words in the text

For example,

```
keyword = ["good", "fantastic"]
words = ["fantastic","fantastic", "good","work"]
print(word_search(keyword, words))
```

will output: `3`.

2.2 Analyze lexicon sentiment: `analyze_sentiment` (5 points)

Once again, we put all the pieces together:

1. Preprocess the data using the `preprocess_text` function from the previous section.
2. Search and count the positive and negative keywords in a text.
3. Compute the score; *i.e.*, the degree of satisfaction or dissatisfaction.

$$\text{partial\_score} = \text{number\_of\_found\_words} * S, \tag{1}$$

where $S = 2$ for the purpose of this project. S is a scalar and can get any value. *partial* can be either positive or negative.[3] *partial* can be either positive or negative.

4. The final score is computed as:

$$\text{sentiment\_score} = \text{positive\_score} - \text{negative\_score}. \tag{2}$$

The parameters for this function are:

- text: text `str` to be analyzed
- positive_keywords: a `list` of given positive keywords to analyze the problem
- negative_keywords: a `list` of given negative keywords to analyze the problem
- unwanted_characters: unwanted characters' `string`
- stop words: a `list` of words that are not useful for the analysis

The code returns an `int` as a score.
For example,

```
text = 'I was moved to the fantastic first class!!!
    After the worst experience!'
positive_keyword = ["good", "fantastic"]
negative_keyword = ["worst"]
words = ["fantastic","fantastic", "good","work"]
print(analyze_sentiment(text, positive_keywords, negative_keywords,
    unwanted_characters, stop_words))
```

This snippet outputs `0`.
Let's take a look at another example,

```
text = 'I feel so good!!! My flight was fantastic!'
positive_keyword = ["good", "fantastic"]
negative_keyword = ["worst"]
words = ["fantastic","fantastic", "good","work"]
print(analyze_sentiment(text, positive_keywords, negative_keywords,
unwanted_characters, stop_words))
```

This snippet outputs `4`.

[3]It is typical to assign different values to S for varying degrees of satisfaction and dissatisfaction of the word sets; *i.e.*, we can have words that are assigned a 2 (for example, good), while others are assigned a 4 (for example, fantastic).

3 Extract aspects (15 points)

To emphasize the critical elements within the feedback, the analysis approach involves computing the word frequency in both positive and negative contexts while subtracting the counts of positive and negative keywords.

To achieve this, you should create a function following these steps:

Algorithm 1 Pseudocode

```
procedure EXTRACT ASPECTS
    positive_aspects, negative_aspects, neutral_aspects = [ ]
    for text in texts do
        sentiment_score = analyze_sentiment(text,...)
        words = preprocess_text(text,...)
        for word in words do
            if sentiment_score > 0 then
                if word not in positive_aspects and word not in positive_keywords then
                    add word to positive_aspects
            else if  sentiment_score < 0 then
                if word not in negative_aspects and not in negative_keywords then
                    add word to negative_aspects
            else if sentiment_score == 0 then
                if word not in neutral_aspects and not in
                    negative_keywords and not in positive_keywords then
                    add word to neutral_aspects
    # rank aspects by frequency
    sorted_positive_aspects = sorted(positive_aspects,...)
    sorted_negative_aspects = sorted(positive_aspects,...)
    sorted_neutral_aspects = sorted(positive_aspects,...)
    return sorted_positive_aspects, sorted_negative sorted_neutral_aspects
```

You must populate a `list` of `lists` for this section. One way to do it is:

```
positive_aspects=[]
word = "wifi"
positive_aspects.append([word,1])
word_2 = "seats"
positive_aspects.append([word_2,1])
print(positive_aspects)
```

The outcome of this snippet is `[['wifi', 1], ['seats', 1]]`

The parameters of this function are:

- text: a `list` of texts to be analyzed
- positive_keywords: a `list` of given positive keywords to analyze the problem
- negative_keywords: a `list` of given negative keywords to analyze the problem

- unwanted_characters: a `str` of unwanted characters' string
- stop words: a `list` of words that are not useful for the analysis

The function should return:

- sorted_positive_aspects: a sorted `list` of `lists` with the number of word occurrences
- sorted_negative_aspects: a sorted `list` of `lists` with the number of word occurrences
- sorted_neutral_aspects: list a sorted `list` of `lists` with the number of word occurrences

For example,

```
reviews = [
    "The overall experience was amazing!",
    "Terrible \$\%\$\% customer service,
        never ever will travel with them again",
    "The attendants were fantastic, but the flight was very bad.",
    "Great value for the price.",
    "Fantastic work!!!",
    "Love it!",
    "Great service!!",
    "Very bad",
    "Super bad",
    "Flight 2345 delayed, bad customer service",
    "AE3546 to Frankfurt on time"
      ]
positive_aspects, negative_aspects, neutral_aspects =
    extract_aspects(reviews,positive_keywords,
    negative_keywords, unwanted_characters,stop_words)
print("Positive aspects:", positive_aspects,
    "\nNegative aspects:",negative_aspects,
    "\nNeutral aspects",neutral_aspects)
```

The output of the snippet is:

```
Positive aspects: [['overall', 1], ['experience', 1], ['value', 1],
['price', 1], ['work', 1], ['service', 1]]
Negative aspects: [['customer', 2], ['service', 2], ['travel', 1],
['very', 1], ['super', 1], ['flight', 1]]
Neutral aspects [['attendants', 1], ['flight', 1], ['very', 1],
['ae', 1], ['frankfurt', 1], ['time', 1]]
```

Hint: Use the provided positive_keywords, negative_keywords, unwanted_characters and stop_words provided in the `main` to test your implementation.

4 Sentiment distribution (10 points)

A sentiment distribution is a statistical distribution of sentiments within a dataset, typically associated with customers' opinions. It represents the frequencies of different sentiment categories or scores; the categories could be positive, negative, or neutral (as we explained before). Hence, the sentiment distribution provides an overview of how sentiments are distributed within the analyzed content.

For our example, the sentiment distribution shows the percentage of reviews categorized as positive, negative, and neutral sentiments. As a result, it provides insights into the overall sentiment tendencies of the reviews.

4.1 Sentiment Distribution and Visualization

The task of this part is to compute and visualize the sentiment distribution of the sentiment scores you have computed. Create a function that calculates the distribution of sentiment scores by counting the number of positive, negative, and neutral scores in the sentiment_scores `list` and stores these counts in a list called `distribution_list`.

This function accepts a unique parameter:

- sentiment_score : a `list` of scores given by analyze_sentiment

Include the following line as the last line of your function:

```
extras.pie_plot(distribution_list)
```

For example, for the next set of scores

```
scores = [-2, 2, 2, 4, 2, 2, 0, 2, 2, 2, 2]
sentiment_distribution(scores)
```

Your code should output the following Pie Chart.

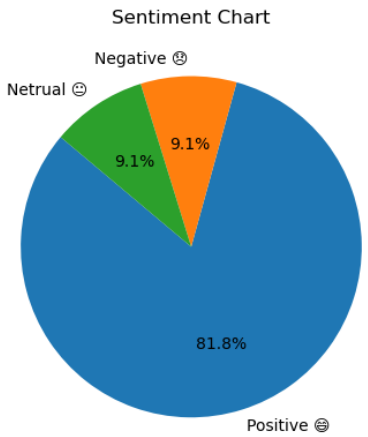

Figure 1: Sentiment Chart Example

Hint: Remember to import the `extra.py` module provided.

5 Compute statistics (5 points)

To compute the statistics, the code must extract and rank the n most positive, negative and neutral aspects mentioned in the reviews. Your task is to create a function with the parameters:

- aspect_category: a `str` that can be positive, negative or neutral.
- aspects: a `list` of `lists` with the number of word occurrences.
- num: an `int` number of aspects to be displayed.

For example,

```
negative_aspects = [['customer', 5], ['service', 4], ['travel', 3],
    ['very', 2], ['flight', 1], ['working', 1]]
extract_rank_aspect("negative", negative_aspects, 5)
```

Will output:

```
Top positive aspects:
flight: 89
service: 48
customer: 33
time: 27
bag: 27
```

6 Code execution (5 points)

For the final execution of the code, we will be using real Twitter phrases.
`reviews = extras.reading_csv_tweet('final_United_Tweets.csv')` reads final_United_Tweets.csv file and stores all the customer reviews in a reviews' `list`.

Follow the next pseudocode as a guide for your final code execution.

Algorithm 2 Pseudocode

```
procedure CODE EXECUTION
    # Define a list of positive keywords
    positive_keywords = ["good","excellent", ...]

    # Define a list of negative keywords
    negative_keywords = ["bad", "terrible", ...]

    # Define unwanted characters
    unwanted_characters = !"#$%&'()*+,-./:;<=>?@[]^_`{|} ~"

    # Define a list of stop words (customized for airlines)
    stop_words = [ "fly", "i", "im", "me", ...]

    # Read CSV file
    reviews = extras.reading_csv_tweet('final_United_Tweets.csv')

    # Analyze sentiments
    sentiment_scores = [analyze_sentiment(review,... )] # populate the sentiment score list

    # Extract aspects
    positive_aspects, negative_aspects, neutral_aspects = extract_aspects(reviews,...)

    # Visualize the sentiment distribution (text-based output)
    sentiment_distribution(sentiment_scores)

    # Display the top positive, negative and neutral aspects
    extract_rank_aspect("positive", positive_aspects, 5)
    extract_rank_aspect("negative", negative_aspects, 5)
    extract_rank_aspect("neutral", neutral_aspects, 5)
```

Use the provided template under the `main` to test all your functions. The output of your code should be:

```
Top positive aspects:
flight: 89
service: 48
customer: 33
time: 27
bag: 27

Top negative aspects:
flight: 50
service: 44
customer: 24
experience: 17
time: 16

Top neutral aspects:
flight: 88
plane: 34
wifi: 23
seat: 22
time: 18
```

Your code should successfully display the following Pie Chart plot.

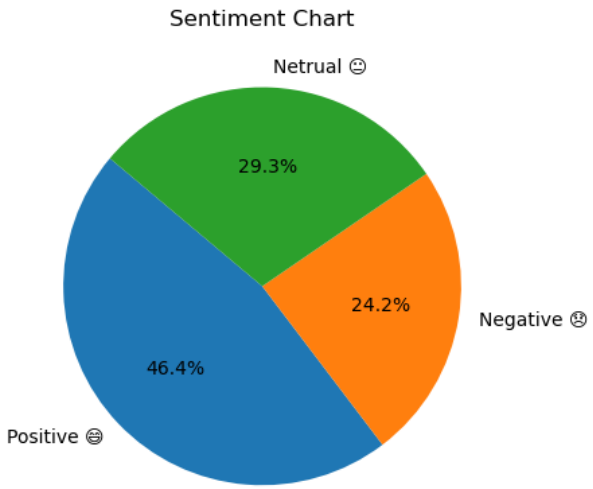

Figure 2: Sentiment Chart, United Airlines Twitter 2015

Grading

- This project should be done individually. Please make sure you are submitting your own work.
- You may not import any modules apart from the ones on the template when completing this project.
- The code should contain comments explaining what the code does.
- Make sure projectB.py is well-commented for full credit. Each method should have an appropriate docstring explaining what it does.
- We will deduct points if file names are misspelled or if Python's global keyword is used.
- Upload projectB.py, to GradeScope by 11:59pm on Friday 10/11/23.