Assignment 3 : Text Processing

Q1: Regular Expression

Define a function "**tokenize**" as follows:

- takes a string as an input
- converts the string into lower case
- tokenizes the lower-cased string into tokens. A token is defined as follows:
    - a token has at least 2 characters
    - a token must start with an alphabetic letter (i.e. a-z or A-Z),
    - a token can have alphabetic letters, "-" (hyphen), "." (dot), "'" (single quote), or "_" (underscore) in the middle
    - a token must end with an alphabetic letter (i.e. a-z or A-Z)
- removes stop words from the tokens (use English stop words list from NLTK)
- returns the resulting token list as the output

Q2: Sentimeent Analysis

1. First define a function "**sentiment_analysis**" as follows:

    - takes a string, a list of positive words, and a list of negative words as inputs. Assume the lists are read from positive-words.txt and negative-words.txt outside of this function.
    - tokenizes the string using the tokeniz function defined above
    - counts positive words and negative words in the tokens using the positive/negative words lists. With a list of negation words (i.e. not, no, isn't, wasn't, aren't, weren't, don't didn't, cannot, couldn't, won't, neither, nor), the final positive/negative words are defined as follows:
        - Positive words:
            - a positive word not preceded by a negation word
            - a negative word preceded by a negation word
        - Negative words:
            - a negative word not preceded by a negation word
            - a positive word preceded by a negation word
        - determined the sentiment of the string as follows:
            - 2: number of positive words > number of negative words
            - 1: number of positive words <= number of negative words
    - returns the sentiment
2. Define a function called **performance_evaluate** to evaluate the accuracy of the sentiment analysis in (1) as follows:

    - takes an input file ("amazon_review_300.csv"), a list of positive words, and a list of negative words as inputs. The input file has a list of reviews in the format of (label, review).
    - reads the input file to get a list of reviews including review text and label of each review
    - for each review, predicts its sentiment using the function defined in
    - returns the accuracy as the number of correct sentiment predictions/total reviews

Q3: (Bonus) Vector Space Model

1. Define a function **find_similar_doc** as follows:

    - takes two inputs: a list of documents (i.e. docs), and the index of a selected document as an integer (i.e. doc_id).
    - uses the "tokenize" function defined in Q1 to tokenize each document
    - generates normalized tf_idf matrix (each row is normalized) from the tokens (hint: reference to the tf_idf function defined in Section 8.5 in lecture notes)
    - calculates the pairwise cosine distance of documents using the generated tf_idf matrix
    - for the selected doc_id, finds the index of the most similar document (but not itself) by the cosine similarity score
    - returns the index of the most similar document and the similarity score
2. Test your function with "amazon_review_300.csv" and a few reviews from this file.
    - Check the most similar review discovered for each of the selected reviews
    - Can you use the calculated similarity score to determine if two documents are similar?
    - Do you think this function can successfully find similar documents? Why does it work or not work?
    - If it does not work, what can you do to improve the search?
    - Write down your analysis along with some evidence or observations you have in a pdf file and submit this pdf file along with your code.

In [6]:

```
import nltk
from nltk.corpus import stopwords
import csv
from scipy.spatial import distance
import pandas as pd
import numpy as np
from sklearn.preprocessing import normalize
```

In [7]:

```
def tokenize(text):

    tokens = None

    # add your code


    return tokens
```

In [8]:

```
def sentiment_analysis(text, positive_words, negative_words):

    negations=["not", "no", "isn't", "wasn't", "aren't", \
               "weren't", "don't", "didn't", "cannot", \
               "couldn't", "won't", "neither", "nor"]

    sentiment = None

    # add your code

    return sentiment


def performance_evaluate(input_file, positive_words, negative_words):

    accuracy = None

    # add your code

    return accuracy
```

In [108]:

```
# Q3

def find_similar_doc(docs,  doc_id):

    top_sim_index, top_sim_score = None, None

    # add your code

    return top_sim_index, top_sim_score
```

In [111]:

```
if __name__ == "__main__":

    # Test Q1
    text="Composed of 3 CDs and quite a few songs (I haven't an exact count), \
          all of which are heart-rendering and impressively remarkable. \
          It has everything for every listener -- from fast-paced and energetic 
\
          (Dancing the Tokage or Termina Home), to slower and more haunting (Dra
gon God), \
          to purely beautifully composed (Time's Scar), \
          to even some fantastic vocals (Radical Dreamers).\
          This is one of the best videogame soundtracks out there, \
          and surely Mitsuda's best ever. ^_^"

    tokens=tokenize(text)

    print("Q1 tokens:", tokens)

    # Test Q2

    with open("../../dataset/positive-words.txt",'r') as f:
        positive_words=[line.strip() for line in f]

    with open("../../dataset/negative-words.txt",'r') as f:
        negative_words=[line.strip() for line in f]

    acc=performance_evaluate("../../dataset/amazon_review_300.csv", \
                              positive_words, negative_words)
    print("\nQ2 accuracy: {0:.2f}".format(acc))


    # Test Q3
    data = pd.read_csv("../../dataset/amazon_review_300.csv")
    # pick any doc id, e.g. 10, 207
    doc_id =207
    sim_doc_id, sim = find_similar_doc(data["review"], doc_id)
    print("\nSimilarity between {0} and {1} is {2:.2f}: "\
          .format(doc_id, sim_doc_id, sim))
    print("\nselected doc: ", data.loc[doc_id]["review"])
    print("\nsimilar doc: ", data.loc[sim_doc_id]["review"])


    doc_id =10
    sim_doc_id, sim = find_similar_doc(data["review"], doc_id)
    print("\nSimilarity between {0} and {1} is {2:.2f}: "\
          .format(doc_id, sim_doc_id, sim))
    print("\nselected doc: ", data.loc[doc_id]["review"])
    print("\nsimilar doc: ", data.loc[sim_doc_id]["review"])
```

```
Q1 tokens: ['composed', 'cds', 'quite', 'songs', 'exact', 'count', '
heart-rendering', 'impressively', 'remarkable', 'everything', 'every
', 'listener', 'fast-paced', 'energetic', 'dancing', 'tokage', 'term
ina', 'home', 'slower', 'haunting', 'dragon', 'god', 'purely', 'beau
tifully', 'composed', "time's", 'scar', 'even', 'fantastic', 'vocals
', 'radical', 'dreamers', 'one', 'best', 'videogame', 'soundtracks',
'surely', "mitsuda's", 'best', 'ever']

Q2 accuracy: 0.71

Similarity between 207 and 206 is 0.20:

selected doc:  I have been using this product for 5+ years. It was w
onderful. About 1 year ago the company changed packaging and the pro
duct changed slightly. The bottle is taller now and something is mis
sing from the serum. Doesn't work as well as it used to work. I will
not be purchasing this item because of the change.

similar doc:  I have been used this product for many years. But some
how the product I received this time is like fake one. It's very thi
n. I have to used double amount.

Similarity between 10 and 15 is 0.16:

selected doc:  A complete waste of time. Typographical errors, poor
grammar, and a totally pathetic plot add up to absolutely nothing. I
'm embarrassed for this author and very disappointed I actually paid
for this book.

similar doc:  It's glaringly obvious that all of the glowing reviews
have been written by the same person, perhaps the author herself. Th
ey all have the same misspellings and poor sentence structure that i
s featured in the book. Who made Veronica Haddon think she is an aut
hor?
```

```
In [ ]:
```