Programming for Data Science
DS2002 course
07.11.2025

Lab 0 Part 2

Summary: This lab is designed to help you practice the basic programming concepts while working with text data.

In this lab, you do not need to solve all the tasks; however, you should do as much as possible.

Task 7: Tokenize function

Goal: write a "tokenizer" function that takes a text, delimiters list, and a threshold as an input and returns a list of all words that are longer than the threshold

Hint: To split a text using multiple delimeters you can use "split" method from "Regular expression operations" (re) library. You can find more information in the following link: https://docs.python.org/3/library/re.html

Task 8: Text Preprocessing for tweets

Goal: write a "text_processing" function that takes a tweet text and does the following:

- Remove all tags
- Remove hashtag sign but return a list of used hastags
- Replace all punctuations with a space
- Remove extra spaces
- Remove any non-printable characters
- The objective of this task is implement the function logic without using any libraries or string functions

Hints:

- The function output is the processed text and a list of hashtags
- You can get a list with all punctuations by importing "string" library and then using "string.punctuation"
- Sometimes working with lists is easier than strings, you can convert a string to list of characters by using (chars = list(text)) and you can get a string from list by using: (text = "".join(chars))

Example:

Tweet text:

"@user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked"

Function output:

Text: "thanks for lyft credit i cant use cause they dont offer wheelchair vans in pdx disapointed getthanked"
Hashtags: {'disapointed', 'lyft', 'getthanked'}

Task 9: build a text corpus

Goal: write a function that takes a list of texts, processes each text by using previous functions from task 7 and 8, converts all words to lower case and removes all stopwords. The function should return a text corpus that store each unique word in provided list of texts along with its frequency and count of texts where it exists. You need to use the right data structure for your corpus.

For each text in the input list, the function should also return a list of remaining words that we assume it better represents the text.

Hints:

- To split a text using multiple delimeters you can use "split" method from "Regular expression operations" (re) library.
- We we use the stopwords list of NLTK library

```
import nltk
nltk.download('stopwords')
stop_words = nltk.corpus.stopwords.words('english')
```

Task 10: calculate TF.IDF

Goal: Given a list of texts (the text can be a document, an email, a tweet, or any other text) we want to create a numerical representation for each text in our list.

In this task we will use a subset of "**Twitter Sentiment Analysis**" dataset and we will calculate TF*IDF representation for theses tweets.

TF*IDF is short for **term frequency–inverse document frequency**, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus[1].

Term Frequency: It is the ratio of the number of times the word appears in a document compared to the total number of words in that document.

TF(w, d) = count of w in d / number of words in d

Inverse document frequency: It is the measure of how much information the word provides about the topic of the document

IDF(w) = log(N/(df + 1)) where:

N: count of all documents
Df: count of document that include word (w)

[1] https://en.wikipedia.org/wiki/Tf%E2%80%93idf

Dataset download link: https://www.kaggle.com/datasets/jp797498e/twitter-entity-sentiment-analysis

And you can find more details about TF*IDF in this link:
https://en.wikipedia.org/wiki/Tf%E2%80%93idf

Steps:

1. Download the dataset
2. Get a random sample of 1000 tweets, that will be considered as our corpus
3. Use the functions developed in previous tasks for text processing
4. Now for each tweet, we have a list of words that represents this tweet
5. Using the corpus we built in task 9 and the list of words for each tweet, we can calculate a TF*IDF value for each word and then we will have a list of TF_IDF values instead a list of words
6. Start by implementing a function that takes a word and a list of words and returns the term frequency **TF** value
7. Implement another function that takes a word and total documents count and returns the inverted document frequency **IDF** value
8. Now to create **TF*IDF** representation for each tweet, you need to create a vector that has the length of our corpus for each tweet. To do that first you need to create a **word_index** that defines an index for each word in the corpus, we will follow this index when creating TF*IDF vector
9. Finally, you can write TF_IDF function that taks a list of words, and calculate **TF*IDF vector** using TF and DF functions from steps 6 and 7

Task 11: calculate similarity between tweets

Goal: write a function that takes two TF*IDF vectors and returns the cosine similarity between them using the following formula:

$$\text{cosine}(\mathbf{v},\mathbf{w}) = \frac{\mathbf{v}\cdot\mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

In order to test the similarity function, select randomly 50 tweets and calculate the similarity between all of them. Then try to compare the simailarity results to the original tweets. How useful is using cosine similarity function? Can we use the hashtags list defined in task 8 to improve our TF*IDF representation?

Extra Task: It is fun to re-implement task 8 without using any ready-made functions, but instead iterating over text characters and processing them in a primitive way.