Data Analysis - lab 7

ISEP – May 19th 2020

Instructions: prepare a report including the source code and the results **either** in two distinct files (.py and .pdf), **or** in a single file (.ipynb). Send one report per group of three students maximum (you can also work alone), and submit it on Moodle, indicating clearly the associated student's names. The deadline is **20/05/2020 1pm**. **A 1 point penalty will be assigned per hour late**.

Libraries

If not done yet, install the following libraries: gensim (bonus), nltk, textblob and wikipedia. Import the needed libraries and sub-packages in top of your script:

```
#Imports
from gensim.summarization import keywords, summarize #bonus, remove if unused
import numpy as np
import nltk
from nltk import sent_tokenize, wordnet, word_tokenize
from nltk.corpus import wordnet as wn
from nltk.corpus import conll2000, stopwords
from textblob import TextBlob
import wikipedia

#Downloads
nltk.download('all')
nltk.download('averaged_perceptron_tagger');
nltk.download('conll2000')
nltk.download('punkt');
nltk.download('stopwords');
nltk.download('wordnet');
```

A Text processing with NLTK

The Natural Language Toolkit, or NLTK for short, is a Python library written for working and modeling text. It provides good tools for loading and cleaning text that we can use to get our data ready for working with machine learning and deep learning algorithms.

1. In this question, we will use the NLTK library to tokenize the text from the `metamorphosis_clean.txt` file.

   (a) Load the text into a string variable named **text** using the *open* and *read()* commands. Display the 5 first lines using the *split()* method. You may check your result by opening the file in a text editor.

   (b) Use the *sent_tokenize()* command to divide your text into sentences. How many sentences are detected? Display the first sentence.

(c) Use the *word_tokenize()* command on your first sentence. Apply the *nltk.pos_tag()* command to the resulting list, and explain what is this command doing.

(d) Use the *split()* method with default parameters on the **text** variable. Display the 10 first resulting items. What is the difference between the results you obtained using the *split()* and *word_tokenize()* methods ?

(e) Display the list of NLTK English stopwords. Define the term stopword.

(f) Remove all the English stopwords from the list created on question (d). Use the *join()* method to create the resulting text, and display its first sentence.

2. In this question, we will work on the coll2000 corpus, a dataset from the Wall Street Journal.

   (a) Using a loop and the *sents()* function of *coll2000*, display the 3 first sentences of this dataset.

   (b) Display the 3 first tagged sentences using the *tagged.sents()* command.

3. **Bonus:** Use the code below to summarize the text from question 1.

```
print(summarize(text))
print(keywords(text))
```

B WordNet

WordNet is a lexical database for the English language, which was created by Princeton University, and is part of the NLTK corpus. You can use WordNet alongside the NLTK module to find the meanings of words, synonyms, antonyms, and more. We are going to go through some examples.

1. The function *synsets()* from wordnet makes it possible to find synonyms for a word, and even to print their definition.

   (a) Using *synsets()*, print all the elements of synsets for the word "Love". Comment.

   (b) Use the properties *definition()*, *lemmas()* and *examples()* on your first synonym. Explain the role of these functions.

   (c) Print all synonyms and antonyms for the word "Love".

   The WordNet corpus reader gives access to the Open Multilingual WordNet, using ISO-639 language codes.

2. Print all available languages in wordnet.

3. Print the lemma name of the first synset of the word "love" in Japanese.

   Several similarity measures are defined inside WordNet: path_similarity, lch_similarity, wup_similarity, etc.

4. Retrieve the sets of synonyms for the words "cat", "dog" and "car". Then, compute the similarity between all pairs of sets using the 3 similarity functions proposed above. Comment.

```
cat = wn.synset('cat.n.01')
dog = wn.synset('dog.n.01')
car = wn.synset('car.n.01')
```

C TextBlob

TextBlob aims to provide access to common text-processing operations through a familiar interface. You can treat TextBlob objects as if they were Python strings that learned how to do Natural Language Processing.

1. Create your first TextBlob object using the following code:

```
doc_01 = TextBlob('Python is a beautiful high-level,'
                  ' general-purpose programming language!')
print(doc_01)
```

   (a) Use the *tags* property of your newly created object. Explain the role of the *tags* method.

   (b) Use the *noun_phrases* property to extract the nouns from the sentence and display the results.

   The sentiment property returns a named tuple of the form Sentiment (polarity, subjectivity). The polarity score is a float within the range [-1.0, 1.0]. The subjectivity is a float within the range [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective.

2. Consider the sentence "Paris is so beautiful, I like Paris, people are so nice !".

   (a) Convert this sentence into a TextBlob object.

   (b) Using the *sentiment* property, analyze its polarity and subjectivity. Comment.

3. Consider the text in the *wiki.txt* file.

   (a) Load this text and convert it into a TextBlob object. Use the *words* and *sentences* properties to divide it into words and sentences. How many words and sentences were detected ?

   (b) Print the sentiment analysis for each sentence. Comment.

   (c) Use the *translate()* function to translate this text into french.

D TF-IDF analysis with TextBlob and Wikipedia

Using the Wikipedia package and the TextBlob package, we are going to analyze a text corpus from Wikipedia, so that we can compute different statistics on word frequencies inside these texts.

1. Create a list of TextBlob documents on the following topics: "France", "Elton John", "Python", "Fox", "Isaac Newton", "Zinedine Zidane". You may complete the code below:

```
corpus = []
topics = #...to complete ...
for topic in topics:
    corpus.append(TextBlob(wikipedia.summary(topic)))
```

2. Use the following steps to write a function that computes the Term Frequency Inverse Document Frequency (TF-IDF):

   (a) Using the properties *words* and *words.count()* from the TextBlob library, write a function **tf**, taking as input a word and a TextBlob document, and returning the number of times the word appears in the TextBlob document divided by the number of words in the TextBlob document.

   (b) Write a function **n_containing**, taking as input a word and a list of TextBlob documents, and returning the number of TextBlob documents where the word appears.

   (c) Write a function **idf**, taking as input a word and a list of TextBlob documents, and returning the associated Inverse Document Frequency (IDF). **Make sure to prevent from any division by zero**.

   (d) Write a function **tfidf**, taking as input a word, a TextBlob document and a list of TextBlob documents, and returning the associated Term Frequency Inverse Document Frequency (TF-IDF).

3. For each TextBlob document, convert all its characters into lowercase and remove the English stopwords.

4. For each TextBlob document, compute the TF-IDF for all the words it contains, display the maximal TF-IDF value and the word of maximal TF-IDF value.