NLP - TP2 REPORT

Prepared by EL HARRARI ANAS

Mai, 15 2023

This report consists of a detailed explanation of the written code and an overview of the manipulation objectives

Table of Contents

Introduction

°Natural language processing is an interdisciplinary subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, mainly how to program computers to process and analyze large amounts of natural language data.

In Natural language processing, natural language is considered essential to generate an efficient algorithm. The process of treating the natural language consists of several steps that take into consideration the language itself, the type of speech and the corpus

One of the first fundamental steps in NLP processing is the pre-treatment of the text; several processes include (cleaning, stemming, tagging and so on). This manipulation aims, as a principal objectification to get familiar with this kind of treatment

In this first manipulation, we will go through some of the most important things in the process of the pre-treatment of the corpus (the aimed text), this manipulation contains two essential parts, the first one seeks to explore the pos_tagging functions of the two popular Open-Source libraries 'NLTK & Spacy' and analyse the major differences between the two, in this part, we will also try to exhibit the importance of tagging in the pre_treatment of a corpus. In the second part, we will make the first step in analyzing the corpus and collecting the external information from it such as the word frequency as we will also figure the concept of bigrams out.

this report contains the details considering the manipulations presented in the attached code and a description of the major purposes of this lab.

° https://en.wikipedia.org/

Exercice1 : Exploring NLTK & Spacy

Overview

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text-processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning.

spaCy is an open-source software library for advanced natural language processing, written in the programming languages Python and Cython

It is a free and open-source library with a lot of built-in capabilities. It's becoming increasingly popular for processing and analyzing data in the field of NLP.

The most well-known techniques include rule-based, artificial neural networks, stochastic, and hybrid.

In the first exercise, we will explore the concept of tagging (or Pos_tagging). Part-of-speech tagging, also known as grammatical tagging in corpus linguistics, is the technique of designating a word in a text as relating to a certain part of speech based on both its meaning and its context.

We aim in this manipulation to understand the pos_tagging so as to compare the different approaches for both NLTK and Spacy.

Several POS tagging systems have been developed to automatically tag words in a phrase with part-of-speech tags. The most well-known techniques include rule-based, artificial neural networks, stochastic, and hybrid.

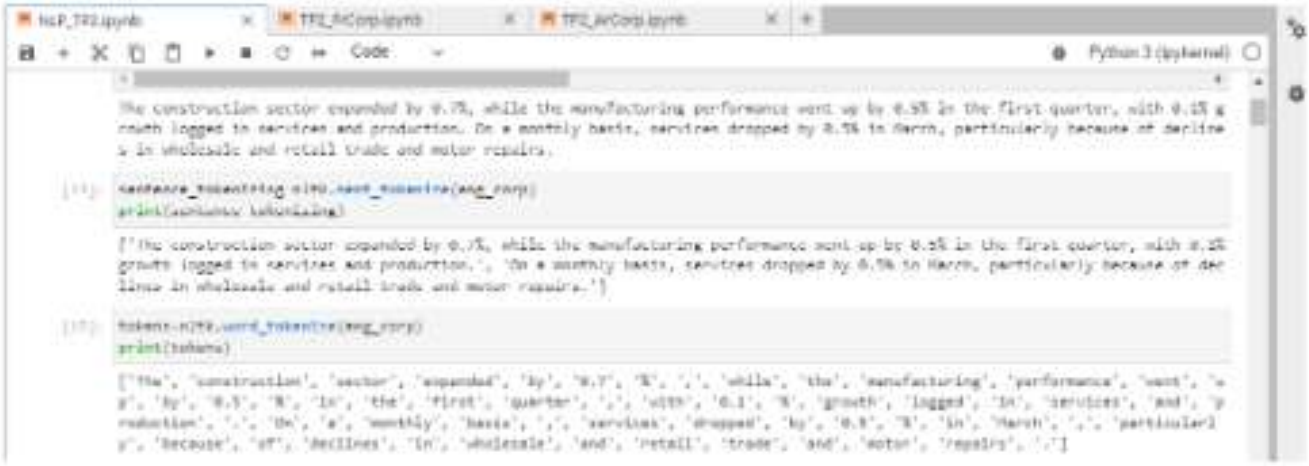
https://www.wikipedia.org/

https://journalofbigdata.springeropen.com/articles/

Exercice1

In order to explore the Pos_Tagging functions of both NLTK and Spacy, we will choose three different corpora from English, French and Arabic. the final objective of this exercise is to compare the obtained results from the different functions.
We start with the following English text which is taken from the CNBC website: "https://www.cnbc.com/2023/05/12/uk-economy-grows-by-0point1percent-in-the-first-quarter-but-inflation-continues-to-weigh-.html"

The pos tagging can easily be done with NLTK by simply calling the function "pos_tag". However, it would be necessary to tokenize our text before processing the tagging. First, we call the function 'sent_tokenize' and 'word_tokenize', which is shown in the capture below :



Then , by simply calling the function 'pos_tag' , we get a list of all the tokenized words and their tags

```
p', 'by', '0.5', '%', 'in', 'the', 'first', 'quarter', ',', 'with', '0.1', '%', 'growth', 'logged', 'in', 'services', 'and', 'p
roduction', '.', 'On', 'a', 'monthly', 'basis', ',', 'services', 'dropped', 'by', '0.5', '%', 'in', 'March', ',', 'particularl
y', 'because', 'of', 'declines', 'in', 'wholesale', 'and', 'retail', 'trade', 'and', 'motor', 'repairs', '.']
```

```
[16]: tagged = nltk.pos_tag(tokens)
      tagged
```

```
[16]: [('the', 'DT'),
       ('construction', 'NN'),
       ('sector', 'NN'),
       ('expanded', 'VBN'),
       ('by', 'IN'),
       ('0.7', 'CD'),
       ('%', 'NN'),
       (',', ','),
       ('while', 'IN'),
       ('the', 'DT'),
       ('manufacturing', 'NN'),
       ('performance', 'NN'),
       ('went', 'VBD'),
       ('up', 'RB'),
       ('by', 'IN'),
       ('0.5', 'CD'),
       ('%', 'NN'),
       ('in', 'IN'),
       ('the', 'DT'),
```

The refrence of all the tag list is availible on the official NLTK documentation

One of the intersting benifits of Pos tagging is generating entities which they can be represented in a very comprehensive form. Entities (or named entity recognition) may be the first step towards information extraction which aims to determine and classify named entities in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, percentages, etc

```
Requirement already satisfied: click<8.0.0,>=7.1.1 in c:\users\hp\anaconda3\lib\site-packages (from typer<0.3.0,>=0.2.0->spacy<
3.0.0,>=2.3.0.dev0->en-core-web-sm==2.3.0) (7.0.4)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\hp\anaconda3\lib\site-packages (from jinja2->spacy<3.0.0,>=2.3.0.de
v0->en-core-web-sm==2.3.0) (2.0.1)
[+] Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
```

```
[5]: import nvgling
     entities = nltk.chunk.ne_chunk(tagged)
     entities
```

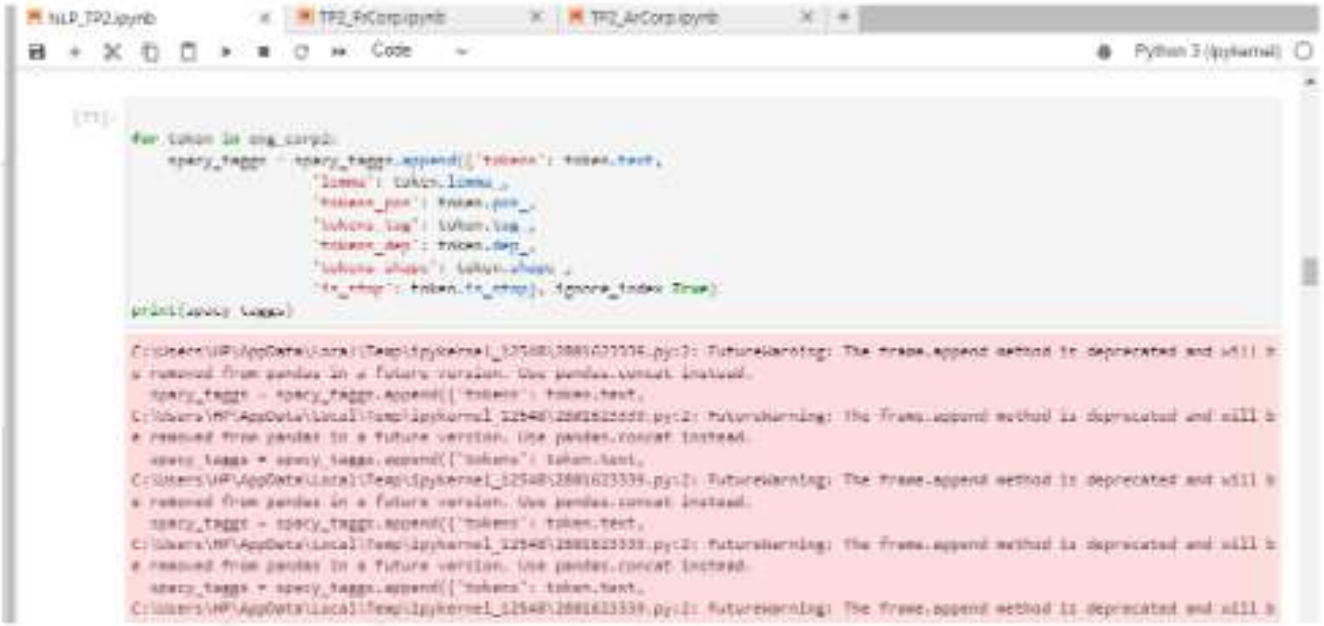
```
[5]:
```

As comparing the functions of both NLTK and Spacy and analyzing the different based approaches is one of the essential objectives of this exercise, we are going, in this part, to explore the pos tagging using the Spacy libraries. We start by creating a spacy object with the same text as before :

Pos Tagging with Spacy

Creating the text

Spacy offers various information extraction tools such as object attributes. We may remind that Spacy is an OOP-based approach. Thus, our corpus is considered an object which inherits all the nlp class attributes (such as tokens, lemma, tags, dependencies and so on)
In this part, we will create a data frame where we will stoke all these attributes for a better representation
After creating the data frame which contains all the text tokens as lines, we add all the object attributes as columns as it is shown in the following captures. These lines of code are the popular way to create and fill a data frame.



And this is the result :

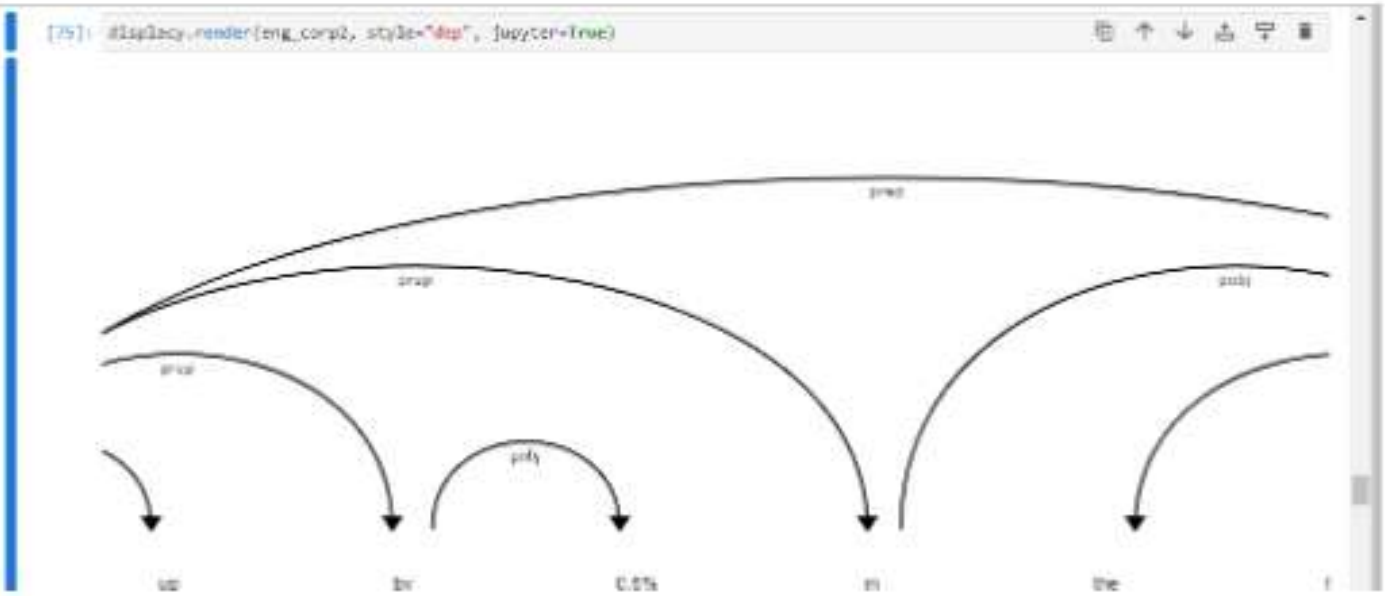
```
spacy_taggs = spacy_taggs.append({'tokens': token.text,
```

	tokens	is_stop	lemma	tokens_dep	tokens_pos	tokens_shape
0	The	True	the	det	DET	Xxx
1	construction	False	construction	compound	NOUN	xxxx
2	sector	False	sector	ROOT	NOUN	xxxx
3	expanded	False	expand	acl	VERB	xxxx
4	by	True	by	prep	ADP	xx
5	0.7	False	0.7	nummod	NUM	d.d
6	%	False	%	pobj	NOUN	%
7	,	False	,	punct	PUNCT	,
8	while	True	while	mark	SCONJ	xxxx
9	manufacturing	False	manufacturing	compound	NOUN	xxxx

Visualization of tags and other attributes may be very helpful to analyse the extracted information. Unfortunately Spacy comes with an interesting visualization under the visualiser 'Displacy'. You can also highlight named entities and their labels with style = "ent".
However, the style = "dep" option does not use any colour to display the POS tags, and the style = "ent" option fails to display the POS tags.
Thus, it may be necessary to develop a specific visualisation function based on the Spacy visualiser

The other attributs may be the object of other manipulations as they consist necessary steps for information extraction

https://towardsdatascience.com/visualizing-part-of-speech-tags-with-nltk-and-spacy-42056fcd777e



The manipulation of the French and so Arabic corpus is followed by the same logic and processing. After specifying the language in the model and text creation, the same functions are applied. The code is presented in detail in the attached labs. Now, we will be moving to one of the main purposes of this exercise: comparison between pos_tagging for both NLTK & Spacy

The French text is taken from the Franceinfo website and the Arabic one is referenced to Wikipedia page.

° https://www.francetvinfo.fr/monde/europe/manifestations-en-ukraine/

After exploring the pos_tagging functionalities for both Spacy and NLTK, we will be moving now to the last step of this part of manipulation which we will try to compare the pos tagging using the both libraries.

Comparison

To sum up :

NLTK	Spacy
pos_tag() function returns a whole list with tokens and their tags	Tags are just an attribute of Spacy Object
NLTK is too slow (0.011 s) in our case	Spacy is ten times faster (0.0011 s) in our case
NLTK gives good results with the three corpus	Only works with English Corpus

However , we may find it essential to mention that both NLTK and Spacy (Despite that fact that NLTK supports the arabic language) , can not give a very great results which may leads to a need of an explicit code programmation in order to ameliorate the occuracy.

Exercice2

Words Frequency and bigrams

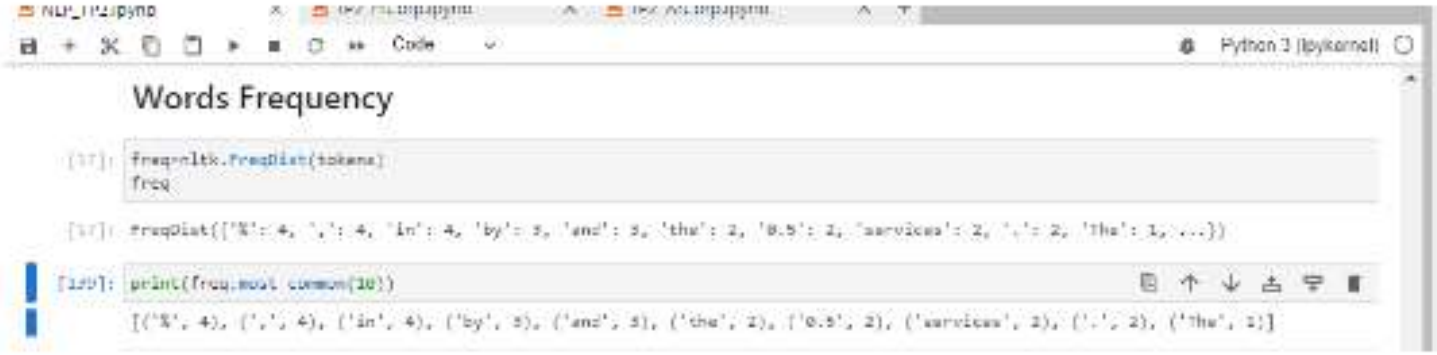
Words Frequency

Words Frequency may be referred to as how an entity often a word appears in a given text. The capacity to count the frequency of words used in a text document or table is a critical step in NLP or Natural Language Process. To do this, we must tokenize the words so that they may be counted as independent objects. There are several libraries available for tokenizing words. However, the most widely used Python library is NLTK.

Bigrams

We identify bigrams in the Bigram language model, which signifies two words coming together in the corpus (the full collection of words/sentences). As an example:
The bigrams in the statement "This is a cute cat" are:
"This is", "is a", "a cute", "cute cat".

We can simply get the frequencies of the text words by using the "FreqDist()" function. This function is provided by NLTK and gives the frequency distribution in the corpus. in other words, this function maps from each sample to the number of times that sample occurred as an outcome. Also, we may be interested to know the most frequent words in our text (Let's the ten most frequent words). This information can be easily get by calling the "most_commun()" function which is shown in the picture below



The result may be sometimes meanless and does not really express the true nature of the corpus. This is due to the presence of some noises (stopwords, prepositions...etc)
In this example, we will be taking only the nouns, verbs and any decimal value or percentage to illustrate this impact

In the most cases , the frequency of the taggs in the corpus (nouns , verbs , prepositions.....) remains an interesting information that may gives a good understanding about the text. In this step , we will , using the same logic as before , extract the most frequent taggs in our text.

```
[('%', 4), ('0.5', 2), ('services', 2), ('construction', 1), ('sector', 1), ('expanded', 1), ('0.7', 1), ('manufacturing', 1),
('performance', 1), ('went', 1)]
```

```
from collections import Counter
tag_freq = Counter(tag for _, tag in tagged)
print(tag_freq)
```

```
Counter({'NN': 14, 'IN': 12, 'DT': 4, 'CD': 4, ',': 4, 'JJ': 4, 'NNS': 4, 'CC': 3, 'VBN': 2, 'VBD': 2, 'RB': 2, '.': 2, 'NNP':
1})
```

```
print(tag_freq.most_common(3))
```

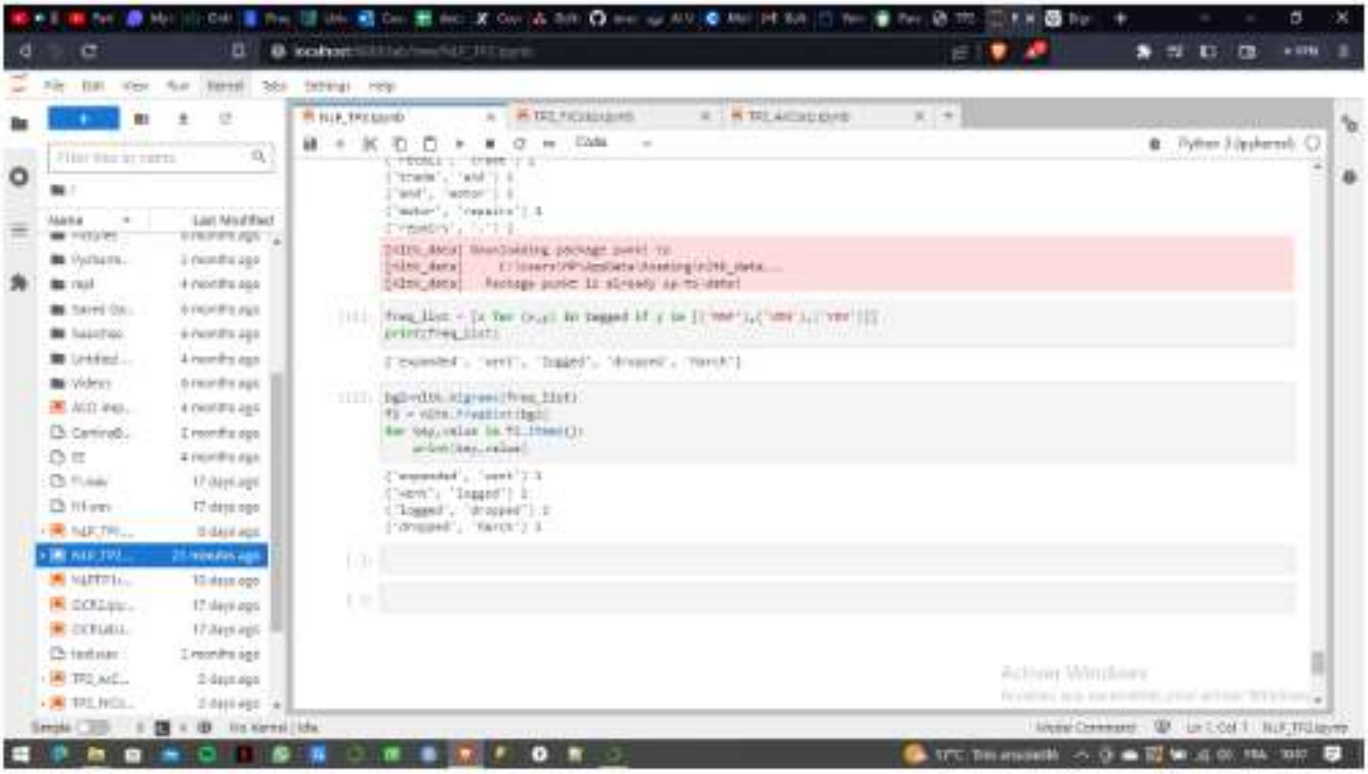
```
[('NN', 14), ('IN', 12), ('DT', 4)]
```

As we said before, a bigram or diagram is a series of two contiguous tokens, which are often letters, syllables, or words. For n=2, a bigram is an n-gram.

In many applications, the frequency distribution of each bigram in a string is often utilized for simple statistical analysis of the text.

The bigrams can also be calculated by simply calling the bigrams() function presented by NLTK (we may notice again that tokenizing the text is a fundamental part of calculating the bigrams). Then, we may use the function FreqDist on these bigrams to calculate the frequency of each pair:

For the last step in this manipulation, we will do the same work on tags (Proper nouns and verbs) to get the most frequent words under these tags



Conclusion

The pre-treatment of a given text remains an essential part of any other process. This work includes tagging, which is considered a critical part of this process and which helps to extract multiple information about the text and its corpora

One of the fundamental tools to achieve this pre_treatment is the two popular Open-Sources Spacy and NLTK

While both can theoretically accomplish any NLP task, each one excels in certain scenarios

Both NLTK and spaCy offer great options when you need to build an NLP system. As we have seen, however, spaCy is the right tool to use in a production environment. Its underlying philosophy – providing a service rather than being a tool – is behind its extreme user-friendliness and performance. spaCy just gets the job done!