Natural Language processing

TP 2 – Tokenization

The objective of text tokenization is to break the text into smaller units which are often more linguistically meaningful.
These smaller linguistic units are usually easier to deal with computationally and semantically.
In this TP we will explore several ways of doing Tokenization, using NLTK, but also doing it manually.

1. Import library

```
from nltk.tokenize import sent_tokenize
```

2. Introuduce this text in english :

para = '''There was nothing so very remarkable in that; nor did Alice think it so very much out of the way to hear the Rabbit say to itself, "Oh dear! Oh dear! I shall be late!" (when she thought it over afterwards,
it occurred to her that she ought to have wondered at this, but at the time it all seemed quite natural);
but when the Rabbit actually took a watch out of its waistcoat-pocket, and looked at it, and then hurried on,
Alice started to her feet, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and burning with curiosity, she ran across the field after it,
and fortunately was just in time to see it pop down a large rabbit-hole under the hedge.'''

3. Print it for visualization

```
for s in sent_tokenize(para):
    print(s+'\n')
```

4. Import first tokenizer method

```
import nltk.data
tokenizer = nltk.data.load('tokenizers/punkt/PY3/english.pickle')
```

5. Tokenize document

```
tokenizer.tokenize(para)
```

6. nltk also provides many pre-trained PunktSentenceTokenizer for other European languages. (Just as information)

```
README          estonian.pickle   italian.pickle    slovene.pickle
czech.pickle    finnish.pickle    norwegian.pickle  spanish.pickle
danish.pickle   french.pickle     polish.pickle     swedish.pickle
dutch.pickle    german.pickle     portuguese.pickle turkish.pickle
english.pickle  greek.pickle      russian.pickle
```

7. Ther methods for tokenization : Similarly, the word_tokenize() function is a wrapper function that calls the tokenize() method on a instance of TreebankWordTokenizer class.

```
from nltk.tokenize import word_tokenize
print(word_tokenize(para)[:20])
```

8. To process large amount of data, please create an instance of TreebankWordTokenizer and call its tokenize() method for more efficient processing. We will get the same results with the following codes as above.

```
from nltk.tokenize import TreebankWordTokenizer
tokenizer = TreebankWordTokenizer()

print(tokenizer.tokenize(para)[:10])
```

9. Comparing different word tokenizer
   a. TreebankWordTokenizer follows the Penn Treebank conventions for word tokenization.
   b. WordPunctTokenizer splits all punctuations into separate tokens.

```
from nltk.tokenize import WordPunctTokenizer
wpt = WordPunctTokenizer()
tbwt = TreebankWordTokenizer()

sent = "Isn't this great? I can't tell!"
```

10. Printing example for both tokenizers

```
wpt.tokenize(sent)
```

```
tbwt.tokenize(sent)
```

Question : which difference do you see ? can you generalize a rule ?

Tokenization using regular expressions

The `nltk` also provides another flexible way for text tokenization based on regular expression.

The `RegexTokenizer` class allows for text tokenization based on the self-defined regular expression patterns.

The regular expression can be created/defined for either the token or the delimiter.

1. Import library

```
from nltk.tokenize import RegexpTokenizer
```

2. Define rule to tokenize

```
retok1 = RegexpTokenizer(pattern= "[a-zA-Z_'-]+")
retok2 = RegexpTokenizer(pattern= "[a-zA-Z_-]+")
retok3 = RegexpTokenizer(pattern= "\s+", gaps=True)
```

3. Printing each tokenization

```
print(retok1.tokenize(sent))
print(retok2.tokenize(sent))
print(retok3.tokenize(sent))
```

Question: See how this tokenizer deals with the apostrophe?