4 | Computational Analyses of Language Use

Extract information from texts

Marco Petolicchio

November 2020

Prerequisites

- `Python` V.3 installed
- `NLTK` library installed and functioning
- Other libraries as `NumPY`, `SciPY`, and so on
- A coffee, tea, or something that you prefer.

The subsequent part will cover some different approaches over feature and information extraction. Note that these instruments are thought for language processing, while they can be used in different fields as computer vision, image recognition, genetic code analysis, and everything you want.

Let’s start :)

Bag of Words

- Bag-of-Word (BOW) model is widely used in feature extraction from texts.
- We can think that each document contains some words from a list.

Let’s imagine that we have two different sentences (inserted here with lowercase for preprocessing purposes):

```
s_1 = ”this sentence contains words.”
s_2 = ”also this one.”
```

The resultant text would be the union of the two (let’s omit the dot):

```
text = ”this sentence contains words also this one”
```

The BOW model permits to obtain quantitative information about the frequency of the items from a list. Now let's assign a binary value which represent the presence (1) or the absence (0) of the items in respect to each sentence:

```
t_1 = [1,1,1,1,0,1,0]
t_2 = [1,0,0,0,1,1,1]
```

For example now that we have a measure of the occurrencies, we could make a tool which tries to predict the genre of the text. BOW is used for spam-filtering purposes in Email, for example, where a recognition of linguistic pattern is made towards mathematical procedures.

The BOW model doesn't care about meaning, context, and so on: the only scope is to consider if a certain word appears in a known document.

Python code

```
import nltk
import matplotlib
import string
from nltk import *
from nltk import word_tokenize
from nltk.corpus import brown
from nltk.corpus import stopwords


def vectorize(tokens):
    vector=[]
    for word in filtered_words:
        vector.append(tokens.count(word))
    return vector
```

```
s_1 = ”this sentence contains words.”

s_2 = ”also this one.”

t_1 = word_tokenize(s_1)

t_2 = word_tokenize(s_2)

words = set(t_1 + t_2)

stop_words = stopwords.words('english')

filtered_words = [word for word in words if word not in stop_words]

v_1 = vectorize(t_1)

v_2 = vectorize(t_2)

print(v_1)

print(v_2)
```

N-Grams

Named Entity recognition