Natural Language Processing

A practical introduction with Python

Meet the instructors

Filippo Chiarello
filippo.chiarello@unipi.it

Simone Barandoni
simone.Barandoni@gmail.com

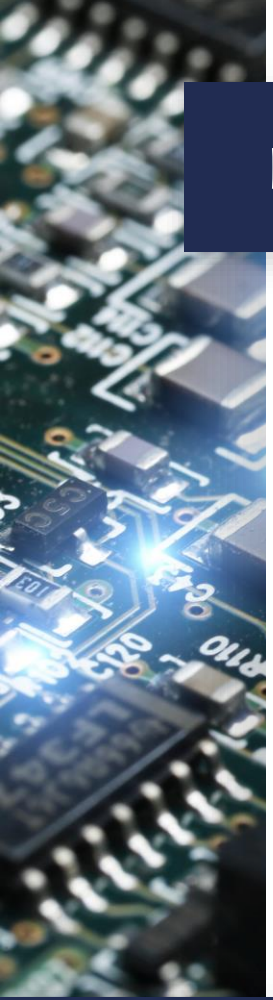
We are part of the B4DS research group:
b4ds.unipi.it

Textual Data

tripadvisor

la Repubblica

Scopus



Natural Language Processing (NLP)

- Branch of artificial intelligence concerned with giving computers the ability to understand text and spoken words.
- Why machines struggle to work with human languages?
  - Ambiguity: "*They were milking cows*"
  - Non-standard languages: "*Great Job @Mark91! U great! SOOO proud of you!* ❤❤❤"
  - Neologisms and idioms: "*retweet*", "*unfriend*"
  - Tricky entity names: "*Let it be was recorded…*"

Natural Language Processing (NLP)

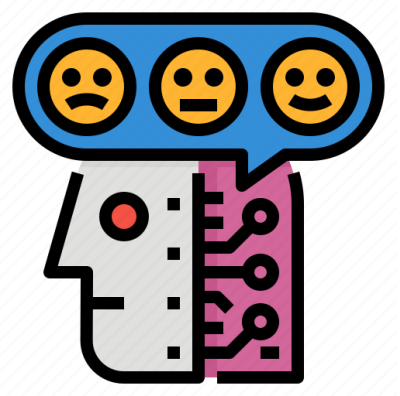
- Some major applications:

Spam Filtering

- The content of an email can be checked and analysed through NLP to determine if it is spam.

Natural Language Processing (NLP)

- Some major applications:



Sentiment Analysis

- A machine can determine if a given text expresses positive, neutral or negative sentiment.

Natural Language Processing (NLP)

- Some major applications:

Speech Recognition

- Spoken language can be recognised and translated into text and vice versa

Natural Language Processing (NLP)

- Some major applications:

Automatic Translators

- Written and spoken texts can be automatically translated into another language

Natural Language Processing (NLP)

- Tools:

  - Simple and clean syntax
  - Easy to learn
  - Huge amount of libraries and packages

- Which version of Python? → 3
- Installation: available at www.python.org

- In this course we will use the Anaconda distribution (www.anaconda.com/products/individual)
- In particular: Jupyter Notebook
- A notebook is an interactive computational environment
- Pieces of code are organised in cells

Natural Language Processing (NLP)

- Tools:
  - Regular Expressions
  - Natural Language ToolKit (NLTK)
  - Set of Python modules which simplify the development of Natural Language Processing methods

Introduction to Python

Jupyter Notebook

Open Jupyter Notebook and *1-Brief_introduction_to_Python* file
You can download it at https://github.com/SimoneBarandoni/nlp-python

Regular Expressions

Regular Expressions

- Searching the occurrence of a word into a text is an easy task
- It consists of searching a string into another one
- Python provides the simple **in** operator:

```
if "hello" in "hello world":
    print("yes")
```

```
yes
```

- But how to deal with more complex searches?

Regular Expressions

- We want to extract from a text:
  - The words starting with ‘in’
  - The words with at least five letters ending with an ‘s’
  - The sentences starting with ‘You’ and ending with a ‘.’
  - All the symbols, including many adjacent ones (e.g. ‘!!!!’)
- To do it we use Regular Expression (RegEx)
  - Standard language to specify search patterns in text, formalized by Stephen Kleene
  - A pattern is a set of rules defining a set of strings which satisfy some criteria
  - A string is a set of characters. For pattern matching, also symbols, spaces, new lines are considered as characters

Regular Expressions in Python

- Module re
- Several useful functions: re. search(…), re.findall(…), re.split(…), re.sub(…), etc.
- Most of the functions take a string and a pattern (a regex) as argument, searching the pattern into the string
- How to write a regex: r'*some text*'
- Regex are case sensitive:

Regex	Matching in string
r'Hello'	«hello world» ✗ «Hello world» ✓
r'goodbye'	«goodbye» ✓ «good bye» ✗

- Some symbols or sequences have special roles inside regex:

Symbol/sequence	Usage example	Match
^	r'^hello'	Strings **starting** with "hello"
$	r'bye$'	Strings **ending** with "bye"
.	r'.'	Any character
\b	r'\bcan\b'	"**can**" preceded and followed by a word border (spaces, symbols): "A can" "American"
\B	r'\Bcan'	"can" not preceded by a word border "American"

- If I need to match the symbols with special roles: add the escape character **** before. **r'\$'** matches a "**$**"; **r'\.'** matches a "**.**"

- To match one or more character that is repeated *n* times we use **quantifiers** after the character which might be repeated

Quantifier	Usage example	Match	Matching example
*	r'hello!*'	"hello" followed by 0 or more "!"	"hello", "hello!", "hello!!", …
+	r'hello!+'	"hello" followed by 1 or more "!"	"hello!", … ,"hello!!!!!!!", …
?	r'hello!?'	"hello" followed by 0 or 1 "!"	"hello", "hello!"
{*x*}	r'hello!{3}'	"hello" followed by exactly 3 copies of "!"	"hello!!!"
{*x*, *y*}	r'hello!{2:4}'	"hello" followed by 2 up to 4 copies of "!"	"hello!!", "hello!!!", "hello!!!!"

- Again, to match precisely the quantifier symbols, use the escape character \

- To assign quantifier to a sequence instead of a single character: **groups**
- A group is writte between parentheses **()** and preceded by the sequence **?:**

Regex	Match	Matching example
r'hello(?:!2!)+'	"hello" followed by one more copies of the sequence !2!	"hello!2!", "hello!2!!2!", "hello!2!!2!!2!", …
r'hello(?:!?1){2}'	"hello" followed by exactly 2 copies of the sequence !?1	"hello!?1"
r'hello(!?1)+'	Without the sequence **?:** It will match the group only	"!?1"

- Character classes **[ ]**: Set of characters in **or** relation

Regex	Match	Matching example
r'[ab]'	Any **a** or **b**	"Dear brother"
r'[1-7]'	Any number from 1 to 7	"I leave at 5:30"
r'[a-zA-Z]'	Any lower or upper case letter	"I leave at 5:30"

- The special symbol ^ stands for ***Not including***

Regex	Match	Matching example
r'[^ab]'	Anything which is not **a** or **b**	"Dear brother"
r'[^hH]ello'	String "ello" preceded by anything but **h** or **H**	"jello" ✓ "1ello" ✓ "Hello" ✗

- Other special characters and corresponding character classes

Regex	Character class	Matching
r'\d'	r'[0-9]'	A number
r'\D'	r'[^0-9]'	Anything but a number
r'\w'	r'[a-zA-Z0-9_]'	An upper or lower case letters, a number, or a '_'
r'\W'	r'[^a-zA-Z0-9_]'	Anything but letters, numbers and '_'
r'\s'	r'[\t\n]'	A space, a tab or a new line
r'\S'	r'[^ \t\n]'	Anything but spaces, tabs and new lines

- Character classes express disjunction between single characters:
  - ‘[abc]’ → **a** or **b** or **c**
- The **OR operator ‘|’** is used to express disjunction between strings:
  - ‘a|bc’ → **a** or **bc**
  - ‘cat|dog’ → **cat** or **dog**
- ‘[Hh]ello’ ≠ ‘H|hello’
  - ‘[Hh]ello’ → **Hello** or **hello**
  - ‘H|hello’ → **H** or **hello**

Further readings: https://docs.python.org/3/library/re.html

- re.findall(***regex***, ***string***):
  - Returns the list of matches of ***regex*** in ***string***. The ***string*** is scanned left-to-right, and matches are returned in the order found. Empty matches are included in the result.
- re.sub(***regex***, ***repl***, ***string***):
  - Return the string obtained by replacing the leftmost non-overlapping occurrences of ***regex*** in ***string*** by the replacement ***repl***. If ***regex*** isn't found, ***string*** is returned unchanged.

REGEX

With Python

Open Jupyter Notebook and *2-Regex_in_Python* file
You can download it at https://github.com/SimoneBarandoni/nlp-python

Exercises 1: Regex

- Write the Regex to match the following strings and use re.findall() to make some test:
  - All the vowels (lowercase and uppercase)
  - Words starting with a consonant
  - Words ending with a punctuation mark
  - Sentences starting with “go” or “ha” (think to a sentence as a set of words and spaces)
- Use re.sub() to substitute *something* in a string in order to have each word written on a new line