Linguistica Computazionale

Laboratorio in Python - IIII

Alessio Miaschi

ItaliaNLP Lab, Istituto di Linguistica Computazionale (CNR-ILC), Pisa

alessio.miaschi@ilc.cnr.it

https://alemiaschi.github.io/

http://www.italianlp.it/alessio-miaschi/

Natural Language Processing in Python

Introduzione

- Nei moduli precedenti, abbiamo visto come aprire, leggere e estrarre semplici informazioni da file di testo (e.g. espressioni regolari, divisione del testo sulla base di spazi e ritorno a capo, etc.)

- Come fare per svolgere operazioni di elaborazione del testo più avanzate?

Introduzione

- Nei moduli precedenti, abbiamo visto come aprire, leggere e estrarre semplici informazioni da file di testo (e.g. espressioni regolari, divisione del testo sulla base di spazi e ritorno a capo, etc.)

- Come fare per svolgere operazioni di elaborazione del testo più avanzate?

- Numerose librerie in Python per il NLP:
  - NLTK;
  - spaCy;
  - Gensim;
  - Stanza.

Introduzione

- Nei moduli precedenti, abbiamo visto come aprire, leggere e estrarre semplici informazioni da file di testo (e.g. espressioni regolari, divisione del testo sulla base di spazi e ritorno a capo, etc.)

- Come fare per svolgere operazioni di elaborazione del testo più avanzate?

- Numerose librerie in Python per il NLP:
  - **NLTK**;
  - spaCy;
  - Gensim;
  - Stanza.

Natural Language Toolkit (NLTK)

- **NLTK** (https://www.nltk.org/) è una delle principali librerie in Python per l'elaborazione del linguaggio naturale

- Dispone di una vasta quantità di corpora e risorse lessicali (e.g. WordNet) e metodi/funzioni per:
  - Tokenizzazione;
  - Stemming;
  - POS tagging e Parsing;
  - Classificazione;
  - etc.

Tokenizzare un testo

```
import sys
import nltk

def main(file1):
  tokens_testo = []
  sent_tokenizer = nltk.data.load("tokenizers/punkt/english.pickle")
  with open(file1, "r") as f:
    for line in f:
      frasi = sent_tokenizer.tokenize(line)
      for frase in frasi:
        tokens = nltk.word_tokenize(frase)
        tokens_testo += tokens

  print("Tokens del testo: ")
  print(tokens_testo)

main(sys.argv[1])
```

Tokenizzare un testo

Tokenizzatore

```
import sys
import nltk

def main(file1):
  tokens_testo = []
  sent_tokenizer = nltk.data.load("tokenizers/punkt/english.pickle")
  with open(file1, "r") as f:
    for line in f:
      frasi = sent_tokenizer.tokenize(line)
      for frase in frasi:
        tokens = nltk.word_tokenize(frase)
        tokens_testo += tokens

  print("Tokens del testo: ")
  print(tokens_testo)

main(sys.argv[1])
```

Tokenizzare un testo

Tokenizzatore

```
import sys
import nltk

def main(file1):
  tokens_testo = []
  sent_tokenizer = nltk.data.load("tokenizers/punkt/english.pickle")
  with open(file1, "r") as f:
    for line in f:
      frasi = sent_tokenizer.tokenize(line)
      for frase in frasi:
        tokens = nltk.word_tokenize(frase)
        tokens_testo += tokens

  print("Tokens del testo: ")
  print(tokens_testo)

main(sys.argv[1])
```

Estrazione dei bigrammi

```
import sys
import nltk
from nltk import bigrams

def main(file1):
  tokens_testo = []
  sent_tokenizer = nltk.data.load("tokenizers/punkt/english.pickle")
  with open(file1, "r") as f:
    for line in f:
      frasi = sent_tokenizer.tokenize(line)
      for frase in frasi:
        tokens = nltk.word_tokenize(frase)
        bigrammi = list(bigrams(tokens))
        print(bigrammi)

main(sys.argv[1])
```

Estrazione dei bigrammi

```
import sys
import nltk
from nltk import bigrams

def main(file1):
  tokens_testo = []
  sent_tokenizer = nltk.data.load("tokenizers/punkt/english.pickle")
  with open(file1, "r") as f:
    for line in f:
      frasi = sent_tokenizer.tokenize(line)
      for frase in frasi:
        tokens = nltk.word_tokenize(frase)
        bigrammi = list(bigrams(tokens))
        print(bigrammi)

main(sys.argv[1])
```

POS Tagging

```
import sys
import stanza

def main(file1):
  sent_tokenizer = nltk.data.load('tokenizers/punkt/italian.pickle')
  with open(file1, 'r') as  f:
    for line in f:
      frasi = sent_tokenizer.tokenize(line)
      for frase in frasi:
        tokens = nltk.word_tokenize(frase)
        tokensPOS = nltk.pos_tag(tokens)
        print(tokensPOS)

main(sys.argv[1])

# >> [('Ciao', 'NNP'), (',', ','), ('sono', 'NN'), ('Alessio', 'NNP'), ('.', '.')]
# >> [('Questa', 'NNP'), ('è', 'NNP'), ('la', 'NNP'), ('prima', 'FW'), ('lezione', 'NN'), ('del', 'NN'), ('corso',
'NN'), ('.', '.')]
```

POS Tagging

```
import sys
import stanza

def main(file1):
  sent_tokenizer = nltk.data.load('tokenizers/punkt/italian.pickle')
  with open(file1, 'r') as  f:
    for line in f:
      frasi = sent_tokenizer.tokenize(line)
      for frase in frasi:
        tokens = nltk.word_tokenize(frase)
        tokensPOS = nltk.pos_tag(tokens)
        print(tokensPOS)

main(sys.argv[1])

# >> [('Ciao', 'NNP'), (',', ','), ('sono', 'NN'), ('Alessio', 'NNP'), ('.', '.')]
# >> [('Questa', 'NNP'), ('è', 'NNP'), ('la', 'NNP'), ('prima', 'FW'), ('lezione', 'NN'), ('del', 'NN'), ('corso',
'NN'), ('.', '.')]
```

POS Tagging

```
import sys
import stanza

def main(file1):
  sent_tokenizer = nltk.data.load('tokenizers/punkt/italian.pickle')
  with open(file1, 'r') as  f:
    for line in f:
      frasi = sent_tokenizer.tokenize(line)
      for frase in frasi:
        tokens = nltk.word_tokenize(frase)
        tokensPOS = nltk.pos_tag(tokens)
        print(tokensPOS)

main(sys.argv[1])

# >> [('Ciao', 'NNP'), (',', ','), ('sono', 'NN'), ('Alessio', 'NNP'), ('.', '.')]
# >> [('Questa', 'NNP'), ('è', 'NNP'), ('la', 'NNP'), ('prima', 'FW'), ('lezione', 'NN'), ('del', 'NN'), ('corso',
'NN'), ('.', '.')]
```

Introduzione

- Nei moduli precedenti, abbiamo visto come aprire, leggere e estrarre semplici informazioni da file di testo (e.g. espressioni regolari, divisione del testo sulla base di spazi e ritorno a capo, etc.)

- Come fare per svolgere operazioni di elaborazione del testo più avanzate?

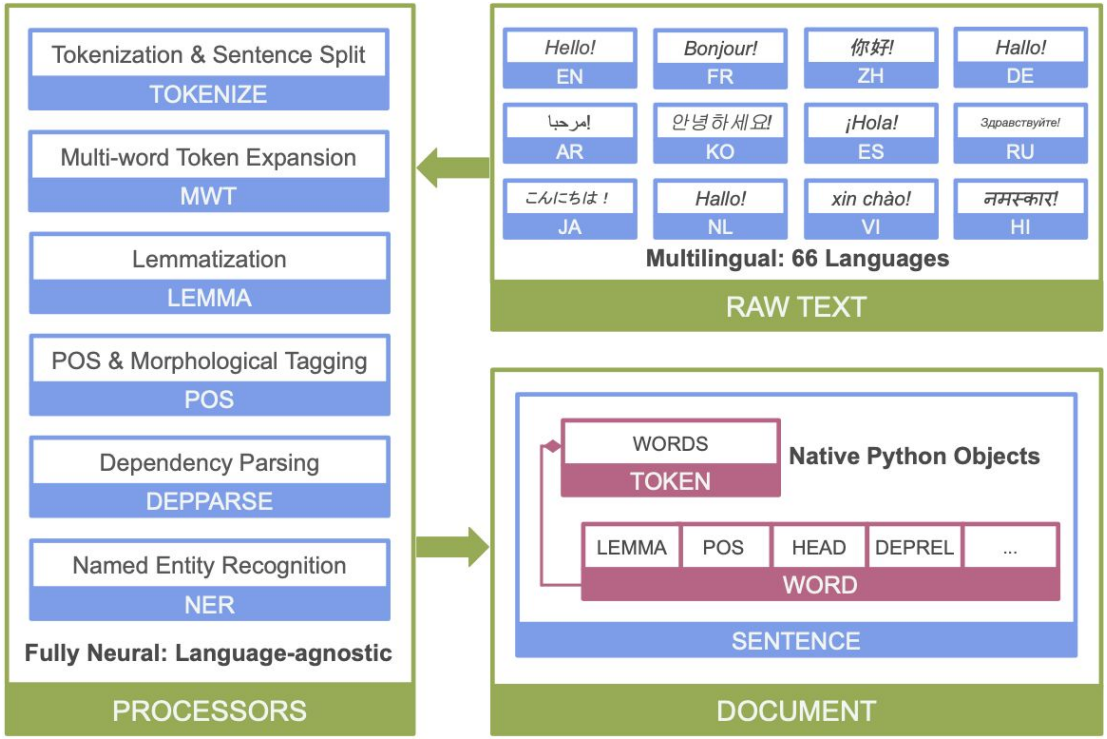
- Numerose librerie in Python per il NLP:
  - NLTK;
  - spaCy;
  - Gensim;
  - **Stanza**.

Stanza

- **Stanza** (https://stanfordnlp.github.io/stanza/) è una libreria in Python sviluppata dallo *Stanford NLP Group* (https://nlp.stanford.edu/)

- Dispone di una serie di tool, da poter utilizzare in una *pipeline*, al fine di poter annotare linguisticamente un testo:
  - Supporto di più di 70 lingue;
  - Particolarmente accurato nell'annotazione linguistica

Stanza


Tokenization & Sentence Split
TOKENIZE
Multi-word Token Expansion
MWT
Lemmatization
LEMMA
POS & Morphological Tagging
POS
Dependency Parsing
DEPPARSE
Named Entity Recognition
NER
Fully Neural: Language-agnostic
PROCESSORS
Hello! EN
Bonjour! FR
你好! ZH
Hallo! DE
مرحبا! AR
안녕하세요! KO
¡Hola! ES
Здравствуйте! RU
こんにちは！ JA
Hallo! NL
xin chào! VI
नमस्कार! HI
Multilingual: 66 Languages
RAW TEXT
WORDS
TOKEN
Native Python Objects
LEMMA
POS
HEAD
DEPREL
...
WORD
SENTENCE
DOCUMENT

Pipeline con Stanza

- Tramite *stanza.Pipeline()* possiamo specificare la lingua del testo e il tipo di processore, e.g.:
  - Tokenizzatore;
  - Multi-Word Token (MWT) Expansion;
  - POS tagging;
  - Dependency parsing;
  - etc.

Pipeline con Stanza

- Tramite *stanza.Pipeline()* possiamo specificare la lingua del testo e il tipo di processore, e.g.:
  - Tokenizzatore;
  - Multi-Word Token (MWT) Expansion;
  - POS tagging;
  - Dependency parsing;
  - etc.

```python
import sys
import stanza

def main(file1):
  fileInput = open(file1, "r")
  raw = fileInput.read()

  nlp = stanza.Pipeline(lang='it', processors='tokenize')
  doc = nlp(raw)
  print(doc)

main(sys.argv[1])

# >> [
# >>  [
# >>    {
# >>      "id": 1,
# >>      "text": "Ciao",
# >>      "start_char": 0,
# >>      "end_char": 4
# >>    },
# >>    {
# >>      "id": 2,
# >>      "text": ",",
# >>      "start_char": 4,
# >>      "end_char": 5
# >> ...
```

Pipeline con Stanza

- Tramite *stanza.Pipeline()* possiamo specificare la lingua del testo e il tipo di processore, e.g.:
  - Tokenizzatore;
  - Multi-Word Token (MWT) Expansion;
  - POS tagging;
  - Dependency parsing;
  - etc.

```python
import sys
import stanza

def main(file1):
  fileInput = open(file1, "r")
  raw = fileInput.read()

  nlp = stanza.Pipeline(lang='it', processors='tokenize')
  doc = nlp(raw)
  print(doc)

main(sys.argv[1])

# >> [
# >>  [
# >>    {
# >>      "id": 1,
# >>      "text": "Ciao",
# >>      "start_char": 0,
# >>      "end_char": 4
# >>    },
# >>    {
# >>      "id": 2,
# >>      "text": ",",
# >>      "start_char": 4,
# >>      "end_char": 5
# >> ...
```

Pipeline con Stanza

- Tramite *stanza.Pipeline()* possiamo specificare la lingua del testo e il tipo di processore, e.g.:
  - Tokenizzatore;
  - Multi-Word Token (MWT) Expansion;
  - POS tagging;
  - Dependency parsing;
  - etc.

```python
import sys
import stanza

def main(file1):
  fileInput = open(file1, "r")
  raw = fileInput.read()

  nlp = stanza.Pipeline(lang='it', processors='tokenize')
  doc = nlp(raw)
  print(doc)

main(sys.argv[1])
```

```
# >> [
# >>  [
# >>    {
# >>      "id": 1,
# >>      "text": "Ciao",
# >>      "start_char": 0,
# >>      "end_char": 4
# >>    },
# >>    {
# >>      "id": 2,
# >>      "text": ",",
# >>      "start_char": 4,
# >>      "end_char": 5
# >> ...
```

Pipeline con Stanza

```python
import stanza

nlp = stanza.Pipeline(lang='it', processors='tokenize,mwt,pos,lemma,depparse')
doc = nlp('Questa è la prima lezione del terzo modulo')

print("id\ttext\thead_id\thead\tdeprel")
for sent in doc.sentences:
  for word in sent.words:
    print(str(word.id) + '\t' + word.text + '\t' + str(word.head) + '\t' +
sent.words[word.head-1].text + '\t' + word.deprel)

# >> id text    head_id head    deprel
# >> 1  Questa  5   lezione nsubj
# >> 2  è   5   lezione cop
# >> 3  la  5   lezione det
# >> 4  prima   5   lezione amod
# >> 5  lezione 0   modulo  root
# >> 6  di  9   modulo  case
# >> 7  il  9   modulo  det
# >> 8  terzo   9   modulo  amod
# >> 9  modulo  5   lezione nmod
```

Esercizi

- Scrivere un programma in python che, a partire da un testo e usando la libreria *nltk/stanza*, estragga il numero totale di frasi presenti al suo interno e calcoli la lunghezza media delle parole (in termini di caratteri)

- Scrivere un programma in python che, a partire da un testo e usando la libreria *nltk/stanza*, restituisca in output la *Type/Token Ratio* (TTR)

- Scrivere un programma in python che, a partire da un testo e usando la libreria *nltk/stanza*, estragga il nome e il verbo (lunghi almeno 4 caratteri) con frequenza massima