Documentazione Progetto Python IA

Matteo Conti and Francesco Iannaccone

8 June 2021

1 Estrazione di informazioni testuali da pdf

```
[ ]: from google.colab import drive
     drive.mount('/content/drive')
```

In questa sezione ci occupiamo di effettuare l'operazione di scanning dei dati che sono caricati da un pdf caricato su drive. Per caricare i dati contenuti nei pdf utilizziamo gli appositi metodi forniti dalla libreria pythin PyPDF2:

```
[ ]: !pip install PyPDF2
     import PyPDF2
     from PyPDF2 import PdfFileReader
```

Apertura del file:

```
[ ]: pdfFileObj = open(r'/content/drive/MyDrive/ontologiaPDF.pdf', mode='rb')
     PdfLeggi = PdfFileReader(pdfFileObj, strict=False)
```

Convertiamo le singole pagine del pdf in stringhe python che sono poi unite nella stringa testo:

```
[ ]: pages_number = 5
     pagina = list()
     testo_pagina = list()
     testo = ''
     for i in range(pages_number):
       pagina.append(PdfLeggi.getPage(i))
       testo_pagina.append(pagina[i].extractText())
       testo = testo + testo_pagina[i]
```

2 Estrazione di informazioni da testo

Ci occupiamo ora delle elaborazione del testo precedentemente caricato utilizzando la libreria nltk, una libreria etremamente potente che ci fornisce tutti gli strumenti fondamentali per il NLP

2.1 Implementazione di una Pipeline NLP in Python

Come fase iniziale di pretrattamento del testo andiamo ad effettuare delle operazioni di parsing: in particolarmodo eseguiamo un'operazione di tokenizzazione scomponendo il testo in sigole parole o nelle singole frasi che lo compongono.

[ ]:
```
import nltk
nltk.download('all', quiet=True)
token = nltk.word_tokenize(testo)
frasi = nltk.sent_tokenize(testo)
```

Segue poi la fase di annotazione sintattica: sfruttando il comando pos_tag della libreria nltk andiamo ad effettuare il tagging grammaticale, associando ad ogni token prodotto nella fase precedente la sua categoria grammaticale:

[ ]:
```
POS_tag = nltk.pos_tag(token)
print(POS_tag)
```

Ci occupiamo a questo punto della stemmatizzazione, andando ad associare ad ogni parola del testo il suo stem, utilizzando lo SNowballStemmer fornito da nltk:

[ ]:
```
from nltk.stem import SnowballStemmer
radice = SnowballStemmer('english')
# radice.stem(token[10])
for word in token:
  stem = radice.stem(word)
  print('parola {}, stem: {}'.format(word, stem))
```

Vogliamo procedere ora alla fase di lemmatizzazione: ad ogni parola del testo vogliamo associare la sua forma di base. Per far ciò utilizzaimo il WordNetLemmatizer che sfrutta un database in internet. Prima però dobbiamo definire una funzione che mappi i PosTag estratti in precedenza nel corretto formato previsto dal lemmatizzatore:

[ ]:
```
from nltk.corpus import wordnet
from nltk.stem import *

lemmatizzatore = WordNetLemmatizer()

def map_postag_to_lemtag(treebank_tag):

    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    elif treebank_tag.startswith('S'):
```

```
        return wordnet.ADJ_SAT
    else:
        return wordnet.NOUN
```

Eseguiamo ora un loop che, ad ogni parola, e corrispondente POS_tag, associ il corrispondente lemma:

[ ]:
```
for i,j in POS_tag:
  pos = map_postag_to_lemtag(j)
  lemma = lemmatizzatore.lemmatize(i.lower(),pos.lower())
  print('parola: {}, lemma: {}'.format(i, lemma))
```

Ci occupiamo a questo punto del chunking: cerchiamo all'interno del testo diversi sintagmi nominali definiti da delle regole "grammar":

Cerchiamo sintagmi composti da articolo, nome e aggettivo:

- le regole sono composte con il seguente formato: grammatica="etichetta:{<POS1><POS2><POS3>}"

[ ]:
```
grammar = "SintagmaNominale:{<DT><JJ><NN>}"
cp = nltk.RegexpParser(grammar)
sintagmiNominali = cp.parse(POS_tag)
print(sintagmiNominali)
```

Cerchiamo ora sintagmi più complessi definendo una regola grammaticale condizionale:

- ? se inserito permette di considerare o meno la presenza del tipo di token ricercato
- * può significare 0 volte a tante volte
- + significa almeno una volta oppure tante volte

[ ]:
```
grammar2 = "SintagmaNominale2:{<DT>?<JJ>*<NN>+}"
cp2 = nltk.RegexpParser(grammar2)
sintagmiNominali2 = cp2.parse(POS_tag)
print(sintagmiNominali2)
```

Cerchiamo sintagmi composti da nome + verbo:

[ ]:
```
grammar3 = "SintagmaNominale3:{<NN><VB.*>}"
cp3 = nltk.RegexpParser(grammar3)
sintagmiNominali3 = cp3.parse(POS_tag)
print(sintagmiNominali3)
```

Cerchiamo in fine sintagmi di diversa natura con una regola or:

[ ]:
```
grammar4 = r"""
Sintagma3elementiArtAggNom:{<DT>?<JJ>*<NN>+}
Sintagma2elementiNomeVerbo:{<NN><VB>}
Sintagma1elementoVerbo:{<VB.*>*}
"""
```

```
cp4 = nltk.RegexpParser(grammar4)
sintagmiNominali4 = cp4.parse(POS_tag)
print(sintagmiNominali4)
```

3 Ricerca sinomini, termini correlati

Cerchiamo dal database wordnet i sinonimi di una parola che sostituiti nel testo non ne stravolgono il significato:

[ ]:
```
from nltk.corpus import wordnet as wn
output = wn.synsets('home','a')
print(output)
```

Cerchiamo in fine la descrizione ed una frase di esempio contenete la parola di interesse:

[ ]:
```
parola_che_mi_interessa = wn.synset('home.s.03')
print(parola_che_mi_interessa.definition())
print(parola_che_mi_interessa.examples())
```