Lab10_NLP_viviyan

May 26, 2021

0.0.1 viviyan richards w

205229133

In this lab, you will extract named entities from the given text file using NLTK. You will also recognize entities based on the regular expression patterns.

0.0.2 EXERCISE-1

0.0.3 Extract all named entities from the following text:

[1]:
```
import nltk
from nltk.tree import Tree
from nltk.tokenize import word_tokenize
from nltk.tag import pos_tag
from nltk.chunk import ne_chunk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\RAVIKUMAR\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\RAVIKUMAR\AppData\Roaming\nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     C:\Users\RAVIKUMAR\AppData\Roaming\nltk_data…
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\RAVIKUMAR\AppData\Roaming\nltk_data…
[nltk_data]   Package words is already up-to-date!
```

[1]:
```
True
```

[2]:
```

```

```
Sentence1 = "Rajkumar said on Monday that WASHINGTON -- In the wake of a string␣
 ↪of abuses by New York police officers in the 1990s,Loretta E. Lynch, the top␣
 ↪federal prosecutor in Brooklyn, spoke forcefully about the pain of a broken␣
 ↪trust that African-Americans felt and said the responsibility for repairing␣
 ↪generations of miscommunication and mistrust fell to law enforcement."
```

[3]:
```
tokens = word_tokenize(Sentence1)
tags = pos_tag(tokens)
ne_tree = ne_chunk(tags)
print(ne_tree[:])
```

```
[Tree('PERSON', [('Rajkumar', 'NNP')]), ('said', 'VBD'), ('on', 'IN'),
('Monday', 'NNP'), ('that', 'IN'), Tree('ORGANIZATION', [('WASHINGTON',
'NNP')]), ('--', ':'), ('In', 'IN'), ('the', 'DT'), ('wake', 'NN'), ('of',
'IN'), ('a', 'DT'), ('string', 'NN'), ('of', 'IN'), ('abuses', 'NNS'), ('by',
'IN'), Tree('GPE', [('New', 'NNP'), ('York', 'NNP')]), ('police', 'NN'),
('officers', 'NNS'), ('in', 'IN'), ('the', 'DT'), ('1990s', 'CD'), (',', ','),
Tree('PERSON', [('Loretta', 'NNP'), ('E.', 'NNP'), ('Lynch', 'NNP')]), (',',
','), ('the', 'DT'), ('top', 'JJ'), ('federal', 'JJ'), ('prosecutor', 'NN'),
('in', 'IN'), Tree('GPE', [('Brooklyn', 'NNP')]), (',', ','), ('spoke', 'VBD'),
('forcefully', 'RB'), ('about', 'IN'), ('the', 'DT'), ('pain', 'NN'), ('of',
'IN'), ('a', 'DT'), ('broken', 'JJ'), ('trust', 'NN'), ('that', 'IN'),
('African-Americans', 'NNP'), ('felt', 'VBD'), ('and', 'CC'), ('said', 'VBD'),
('the', 'DT'), ('responsibility', 'NN'), ('for', 'IN'), ('repairing', 'VBG'),
('generations', 'NNS'), ('of', 'IN'), ('miscommunication', 'NN'), ('and', 'CC'),
('mistrust', 'NN'), ('fell', 'VBD'), ('to', 'TO'), ('law', 'NN'),
('enforcement', 'NN'), ('.', '.')]
```

[4]:
```
ne_tree = ne_chunk(pos_tag(word_tokenize(Sentence1)))
```

[5]:
```
for i in ne_tree:
    print(i)
```

```
(PERSON Rajkumar/NNP)
('said', 'VBD')
('on', 'IN')
('Monday', 'NNP')
('that', 'IN')
(ORGANIZATION WASHINGTON/NNP)
('--', ':')
('In', 'IN')
('the', 'DT')
('wake', 'NN')
('of', 'IN')
('a', 'DT')
('string', 'NN')
('of', 'IN')
('abuses', 'NNS')
```

```
('by', 'IN')
(GPE New/NNP York/NNP)
('police', 'NN')
('officers', 'NNS')
('in', 'IN')
('the', 'DT')
('1990s', 'CD')
(',', ',')
(PERSON Loretta/NNP E./NNP Lynch/NNP)
(',', ',')
('the', 'DT')
('top', 'JJ')
('federal', 'JJ')
('prosecutor', 'NN')
('in', 'IN')
(GPE Brooklyn/NNP)
(',', ',')
('spoke', 'VBD')
('forcefully', 'RB')
('about', 'IN')
('the', 'DT')
('pain', 'NN')
('of', 'IN')
('a', 'DT')
('broken', 'JJ')
('trust', 'NN')
('that', 'IN')
('African-Americans', 'NNP')
('felt', 'VBD')
('and', 'CC')
('said', 'VBD')
('the', 'DT')
('responsibility', 'NN')
('for', 'IN')
('repairing', 'VBG')
('generations', 'NNS')
('of', 'IN')
('miscommunication', 'NN')
('and', 'CC')
('mistrust', 'NN')
('fell', 'VBD')
('to', 'TO')
('law', 'NN')
('enforcement', 'NN')
('.', '.')
```

0.0.4 Question-1

0.0.5 Count and print the number of PERSON, LOCATION and ORGANIZATION in the given sentence.

[6]:
```
import nltk
from collections import Counter
for chunk in ne_tree:
    if hasattr(chunk, 'label'):
        print([Counter(label) for label in chunk])
```

```
[Counter({'Rajkumar': 1, 'NNP': 1})]
[Counter({'WASHINGTON': 1, 'NNP': 1})]
[Counter({'New': 1, 'NNP': 1}), Counter({'York': 1, 'NNP': 1})]
[Counter({'Loretta': 1, 'NNP': 1}), Counter({'E.': 1, 'NNP': 1}),
Counter({'Lynch': 1, 'NNP': 1})]
[Counter({'Brooklyn': 1, 'NNP': 1})]
```

0.1 Question 2

0.1.1 Observe the results. Does named entity, "police officers" get recognized?.

[7]:
```
word = nltk.word_tokenize(Sentence1)
pos_tag = nltk.pos_tag(word)
chunk = nltk.ne_chunk(pos_tag)
grammar = "NP: {<NN><NNS>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(chunk)
NE = [ " ".join(w for w, t in ele) for ele in result if isinstance(ele, nltk.
 ↪Tree)]
print (NE)
```

```
['Rajkumar', 'WASHINGTON', 'New York', 'police officers', 'Loretta E. Lynch',
'Brooklyn']
```

0.1.2 Write a regular expression patter to detect this. You will need nltk.RegexpParser class to define pattern and parse terms to detect patterns.

[8]:
```
grammar = "NP: {<NN><NNS>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(ne_tree)
NE = [ " ".join(w for w, t in ele) for ele in result if isinstance(ele, nltk.
 ↪Tree)]
print(NE)
```

```
['Rajkumar', 'WASHINGTON', 'New York', 'police officers', 'Loretta E. Lynch',
'Brooklyn']
```

0.1.3 Question-3

Does the named entity, “the top federal prosecutor” get recognized?.

[9]:
```
out=cp.parse(tags)
print(out[:])
```

```
[('Rajkumar', 'NNP'), ('said', 'VBD'), ('on', 'IN'), ('Monday', 'NNP'), ('that',
'IN'), ('WASHINGTON', 'NNP'), ('--', ':'), ('In', 'IN'), ('the', 'DT'), ('wake',
'NN'), ('of', 'IN'), ('a', 'DT'), ('string', 'NN'), ('of', 'IN'), ('abuses',
'NNS'), ('by', 'IN'), ('New', 'NNP'), ('York', 'NNP'), Tree('NP', [('police',
'NN'), ('officers', 'NNS')]), ('in', 'IN'), ('the', 'DT'), ('1990s', 'CD'),
(',', ','), ('Loretta', 'NNP'), ('E.', 'NNP'), ('Lynch', 'NNP'), (',', ','),
('the', 'DT'), ('top', 'JJ'), ('federal', 'JJ'), ('prosecutor', 'NN'), ('in',
'IN'), ('Brooklyn', 'NNP'), (',', ','), ('spoke', 'VBD'), ('forcefully', 'RB'),
('about', 'IN'), ('the', 'DT'), ('pain', 'NN'), ('of', 'IN'), ('a', 'DT'),
('broken', 'JJ'), ('trust', 'NN'), ('that', 'IN'), ('African-Americans', 'NNP'),
('felt', 'VBD'), ('and', 'CC'), ('said', 'VBD'), ('the', 'DT'),
('responsibility', 'NN'), ('for', 'IN'), ('repairing', 'VBG'), ('generations',
'NNS'), ('of', 'IN'), ('miscommunication', 'NN'), ('and', 'CC'), ('mistrust',
'NN'), ('fell', 'VBD'), ('to', 'TO'), ('law', 'NN'), ('enforcement', 'NN'),
('.', '.')]
```

Write a regular expression pattern to detect this.

[10]:
```
grammar = "NP: {<DT><JJ>*<NN>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(ne_tree)
NE = [ " ".join(w for w, t in ele) for ele in result if isinstance(ele, nltk.
 ↪Tree)]
print (NE)
```

```
['Rajkumar', 'WASHINGTON', 'the wake', 'a string', 'New York', 'Loretta E.
Lynch', 'the top federal prosecutor', 'Brooklyn', 'the pain', 'a broken trust',
'the responsibility']
```

0.2 EXERCISE-2

0.2.1 Question-1

0.2.2 Observe the output. Does your code recognize the NE shown in BOLD?($5.1 billion, the mobile phone, the company)

[16]:
```
Sentence2 = "European authorities fined Google a record $5.1 billion on␣
 ↪Wednesday for abusing its power in the mobile phone market and ordered the␣
 ↪company to alter its practices"
```

[24]:
```
tok = word_tokenize(Sentence2)
tagged = nltk.pos_tag(tok)
ne_tree2 = nltk.ne_chunk(tagged,binary=False)
```

```
print(ne_tree2[:])
```

```
[Tree('GPE', [('European', 'JJ')]), ('authorities', 'NNS'), ('fined', 'VBD'),
Tree('PERSON', [('Google', 'NNP')]), ('a', 'DT'), ('record', 'NN'), ('$', '$'),
('5.1', 'CD'), ('billion', 'CD'), ('on', 'IN'), ('Wednesday', 'NNP'), ('for',
'IN'), ('abusing', 'VBG'), ('its', 'PRP$'), ('power', 'NN'), ('in', 'IN'),
('the', 'DT'), ('mobile', 'JJ'), ('phone', 'NN'), ('market', 'NN'), ('and',
'CC'), ('ordered', 'VBD'), ('the', 'DT'), ('company', 'NN'), ('to', 'TO'),
('alter', 'VB'), ('its', 'PRP$'), ('practices', 'NNS')]
```

0.2.3 Write a regular expression that recognizes the entity

[52]:
```
word = nltk.word_tokenize(Sentence2)
pos_tag = nltk.pos_tag(word)
chunk = nltk.ne_chunk(pos_tag)
grammar = "NP: {<CD>|<DT><JJ>*<NN>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(chunk)
NE = [ " ".join(w for w, t in ele) for ele in result if isinstance(ele, nltk.
 ↪Tree)]
print (NE)
```

```
['European', 'Google', 'a record', '5.1', 'billion', 'the mobile phone', 'the
company']
```

0.2.4 Question-2

0.2.5 Write a regular expression that recognizes the entity, “the mobile phone” and similar to this entity such as “the company

[50]:
```
word = nltk.word_tokenize(Sentence2)
pos_tag = nltk.pos_tag(word)
chunk = nltk.ne_chunk(pos_tag)
grammar = "NP: {<DT><JJ>*<NN>}"
cp = nltk.RegexpParser(grammar)
result = cp.parse(chunk)
NE = [ " ".join(w for w, t in ele) for ele in result if isinstance(ele, nltk.
 ↪Tree)]
print (NE)
```

```
['European', 'Google', 'a record', 'the mobile phone', 'the company']
```