NAMED ENTITY RECOGNITION NER USING SPACY | NLP | PART 4

April 27, 2020 Ashutosh Tripathi 2 comments

Named Entity Recognition is the most important or I would say the starting step in Information Retrieval. Information Retrieval is the technique to extract important and useful information from unstructured raw text documents. Named Entity Recognition NER works by locating and identifying the named entities present in unstructured text into the standard categories such as person names, locations, organizations, time expressions, quantities, monetary values, percentage, codes etc. Spacy comes with an extremely fast statistical entity recognition system that assigns labels to contiguous spans of tokens.

Spacy Installation and Basic Operations | NLP Text Processing Library | Part 1

Spacy provides option to add arbitrary classes to entity recognition system and update the model to even include the new examples apart from already defined entities within model.

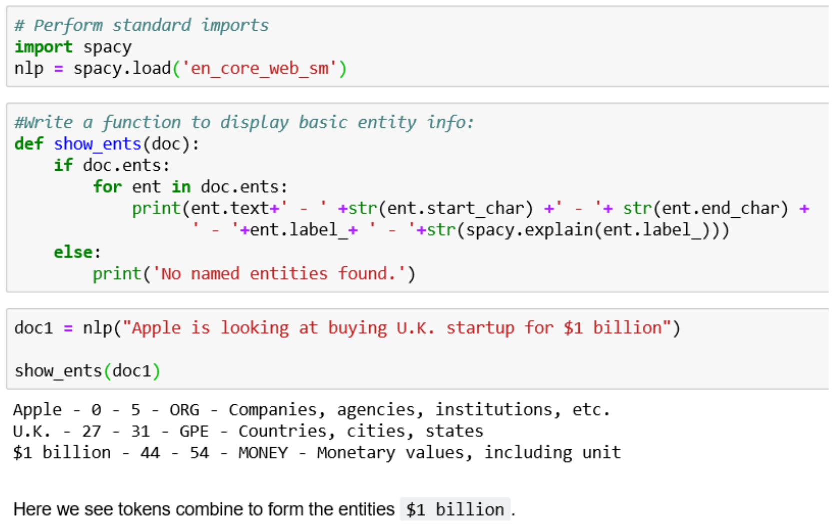
Spacy has the **'ner'** pipeline component that identifies token spans fitting a predetermined set of named entities. These are available as the **'ents'** property of a Doc object.

Complete Jupyter Notebook NER-Named-Entity-Recognition Download

```
# Perform standard imports
import spacy
nlp = spacy.load('en_core_web_sm')
```

```
# Write a function to display basic entity info:
def show_ents(doc):
if doc.ents:
for ent in doc.ents:
    print(ent.text+' - ' +str(ent.start_char) +' - '+
str(ent.end_char) +' - '+ent.label_+ ' -
'+str(spacy.explain(ent.label_)))
else:
    print('No named entities found.')
```

```
doc1 = nlp("Apple is looking at buying U.K. startup for $1
billion")
show_ents(doc1)
```



```
# Perform standard imports
import spacy
nlp = spacy.load('en_core_web_sm')
```

```
#Write a function to display basic entity info:
def show_ents(doc):
    if doc.ents:
        for ent in doc.ents:
            print(ent.text+' - ' +str(ent.start_char) +' - '+ str(ent.end_char) +
                  ' - '+ent.label_+ ' - '+str(spacy.explain(ent.label_)))
    else:
        print('No named entities found.')
```

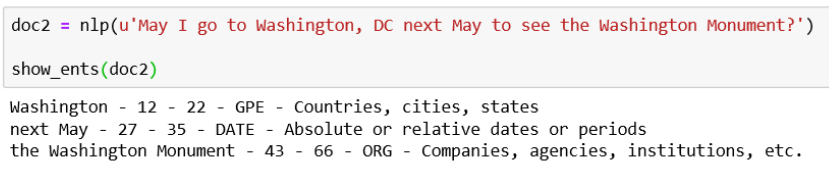
```
doc1 = nlp("Apple is looking at buying U.K. startup for $1 billion")

show_ents(doc1)
```

```
Apple - 0 - 5 - ORG - Companies, agencies, institutions, etc.
U.K. - 27 - 31 - GPE - Countries, cities, states
$1 billion - 44 - 54 - MONEY - Monetary values, including unit
```

Here we see tokens combine to form the entities `$1 billion`.

Text	Start	End	Label	Description
Apple	0	5	`ORG`	Companies, agencies, institutions.
U.K.	27	31	`GPE`	Geopolitical entity, i.e. countries, cities, states.
$1 billion	44	54	`MONEY`	Monetary values, including unit.

```
doc2 = nlp(u'May I go to Washington, DC next May to see the
Washington Monument?')
show_ents(doc2)
```

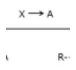


```
doc2 = nlp(u'May I go to Washington, DC next May to see the Washington Monument?')

show_ents(doc2)
```

```
Washington - 12 - 22 - GPE - Countries, cities, states
next May - 27 - 35 - DATE - Absolute or relative dates or periods
the Washington Monument - 43 - 66 - ORG - Companies, agencies, institutions, etc.
```

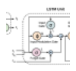
Here we see tokens combine to form the entities `next May` and `the Washington Monument`

```
doc3 = nlp(u'Can I please borrow 500 dollars from you to buy
some Microsoft stock?')
for ent in doc3.ents:
    print(ent.text, ent.start, ent.end, ent.start_char,
ent.end_char, ent.label_)
```


BCNF Decomposition | A step by step approach

Types of Functional Dependencies in Normalization

How to Perform Sentence Segmentation or Sentence Tokenization using spaCy | NLP Series | Part 5


What is the main difference between RNN and LSTM | NLP | RNN vs LSTM

Follow Data Science Duniya

```
doc3 = nlp(u'Can I please borrow 500 dollars from you to buy some Microsoft stock?')

for ent in doc3.ents:
    print(ent.text, ent.start, ent.end, ent.start_char, ent.end_char, ent.label_)
```

```
500 dollars 4 6 20 31 MONEY
Microsoft 11 12 53 62 ORG
```

Entity Annotations

`Doc.ents` are token spans with their own set of annotations.

`ent.text`	The original entity text
`ent.label`	The entity type's hash value
`ent.label_`	The entity type's string description
`ent.start`	The token span's *start* index position in the Doc
`ent.end`	The token span's *stop* index position in the Doc
`ent.start_char`	The entity text's *start* index position in the Doc
`ent.end_char`	The entity text's *stop* index position in the Doc

Accessing Entity Annotations

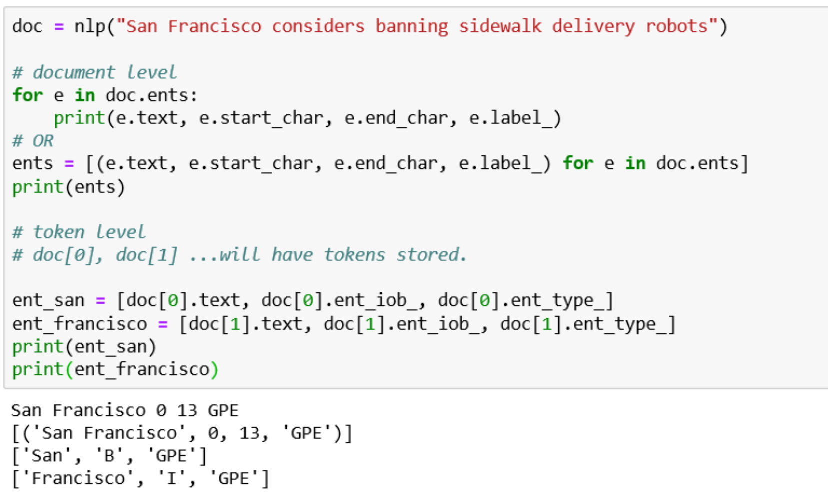
The standard way to access entity annotations is the doc.ents property, which produces a sequence of Span objects. The entity type is accessible either as a hash value using **ent.label** or as a string using **ent.label_**.

The Span object acts as a sequence of tokens, so you can iterate over the entity or index into it. You can also get the text form of the whole entity, as though it were a single token.

You can also access token entity annotations using the token.ent_iob and token.ent_type attributes. token.ent_iob indicates whether an entity starts, continues or ends on the tag. If no entity type is set on a token, it will return an empty string.

```
doc = nlp("San Francisco considers banning sidewalk delivery
robots")
# document level
for e in doc.ents:
    print(e.text, e.start_char, e.end_char, e.label_)
# OR
ents = [(e.text, e.start_char, e.end_char, e.label_) for e in
doc.ents]
print(ents)
```

```
#token level
# doc[0], doc[1] …will have tokens stored.
ent_san = [doc[0].text, doc[0].ent_iob_, doc[0].ent_type_]
ent_francisco = [doc[1].text, doc[1].ent_iob_, doc[1].ent_type_]
print(ent_san)
print(ent_francisco)
```



```
doc = nlp("San Francisco considers banning sidewalk delivery robots")

# document level
for e in doc.ents:
    print(e.text, e.start_char, e.end_char, e.label_)
# OR
ents = [(e.text, e.start_char, e.end_char, e.label_) for e in doc.ents]
print(ents)

# token level
# doc[0], doc[1] ...will have tokens stored.

ent_san = [doc[0].text, doc[0].ent_iob_, doc[0].ent_type_]
ent_francisco = [doc[1].text, doc[1].ent_iob_, doc[1].ent_type_]
print(ent_san)
print(ent_francisco)
```

```
San Francisco 0 13 GPE
[('San Francisco', 0, 13, 'GPE')]
['San', 'B', 'GPE']
['Francisco', 'I', 'GPE']
```

```
IOB SCHEME
I – Token is inside an entity.
O – Token is outside an entity.
B – Token is the beginning of an entity.
```

Text	ent_iob	ent_iob_	ent_type_	Description
San	3	B	"GPE"	beginning of an entity
Francisco	1	I	"GPE"	inside an entity
considers	2	O	""	outside an entity
banning	2	O	""	outside an entity
sidewalk	2	O	""	outside an entity
delivery	2	O	""	outside an entity

Text	ent_iob	ent_iob_	ent_type_	Description
robots	2	0	" "	outside an entity

Note: In the above example only `San Francisco` is recognized as named entity. hence rest of the tokens are described as outside the entity. And in `San Francisco` `San` is the starting of the entity and `Francisco` is inside the entity.

NER Tags

Tags are accessible through the `.label_` property of an entity.

TYPE	DESCRIPTION	EXAMPLE
`PERSON`	People, including fictional.	*Fred Flintstone*
`NORP`	Nationalities or religious or political groups.	*The Republican Party*
`FAC`	Buildings, airports, highways, bridges, etc.	*Logan International Airport, The Golden Gate*
`ORG`	Companies, agencies, institutions, etc.	*Microsoft, FBI, MIT*
`GPE`	Countries, cities, states.	*France, UAR, Chicago, Idaho*
`LOC`	Non-GPE locations, mountain ranges, bodies of water.	*Europe, Nile River, Midwest*
`PRODUCT`	Objects, vehicles, foods, etc. (Not services.)	*Formula 1*
`EVENT`	Named hurricanes, battles, wars, sports events, etc.	*Olympic Games*
`WORK_OF_ART`	Titles of books, songs, etc.	*The Mona Lisa*
`LAW`	Named documents made into laws.	*Roe v. Wade*
`LANGUAGE`	Any named language.	*English*
`DATE`	Absolute or relative dates or periods.	*20 July 1969*
`TIME`	Times smaller than a day.	*Four hours*
`PERCENT`	Percentage, including "%".	*Eighty percent*

MONEY	Monetary values, including unit.	*Twenty Cents*
QUANTITY	Measurements, as of weight or distance.	*Several kilometers, 55kg*
ORDINAL	“first”, “second”, etc.	*9th, Ninth*
CARDINAL	Numerals that do not fall under another type.	*2, Two, Fifty-two*

User Defined Named Entity and Adding it to a Span

Normally we would have spaCy build a library of named entities by training it on several samples of text.
Sometimes, we want to assign specific token a named entity whic is not recognized by the trained spacy model. We can do this as shown in below code.

Example 1

```
doc = nlp(u'Tesla to build a U.K. factory for $6 million')

show_ents(doc)
```

```
U.K. - 17 - 21 - GPE - Countries, cities, states
$6 million - 34 - 44 - MONEY - Monetary values, including unit
```

Right now, spaCy does not recognize "Tesla" as a company.

```
from spacy.tokens import Span
```

```


# Get the hash value of the ORG entity label
ORG = doc.vocab.strings[u'ORG']

# Create a Span for the new entity
new_ent = Span(doc, 0, 1, label=ORG)

# Add the entity to the existing Doc object
doc.ents = list(doc.ents) + [new_ent]
```

In the code above, the arguments passed to `Span()` are:

- `doc` - the name of the Doc object
- `0` - the *start* index position of the token in the doc
- `1` - the *stop* index position (exclusive) in the doc
- `label=ORG` - the label assigned to our entity

```
show_ents(doc)
```

```
Tesla - 0 - 5 - ORG - Companies, agencies, institutions, etc.
U.K. - 17 - 21 - GPE - Countries, cities, states
$6 million - 34 - 44 - MONEY - Monetary values, including unit
```

Example 2

```
doc = nlp("fb is hiring a new vice president of global policy")
ents = [(e.text, e.start_char, e.end_char, e.label_) for e in doc.ents]
print('Before', ents)
# the model didn't recognise "fb" as an entity :(

fb_ent = Span(doc, 0, 1, label="ORG") # create a Span for the new entity
doc.ents = list(doc.ents) + [fb_ent]

ents = [(e.text, e.start_char, e.end_char, e.label_) for e in doc.ents]
print('After', ents)
# [('fb', 0, 2, 'ORG')]
```

```
Before []
After [('fb', 0, 2, 'ORG')]
```

Adding Named Entities to All Matching Spans

What if we want to tag *all* occurrences of a token? In this section we show how to use the PhraseMatcher to identify a series of spans in the Doc:

```
doc = nlp(u'Our company plans to introduce a new vacuum cleaner.
If successful, the vacuum cleaner will be our first product.')
show_ents(doc)
#output: first - 99 - 104 - ORDINAL - "first", "second", etc.
```

```
#Import PhraseMatcher and create a matcher object:
from spacy.matcher import PhraseMatcher
matcher = PhraseMatcher(nlp.vocab)
```

```
#Create the desired phrase patterns:
phrase_list = ['vacuum cleaner', 'vacuum-cleaner']
phrase_patterns = [nlp(text) for text in phrase_list]
```

```
#Apply the patterns to our matcher object:
matcher.add('newproduct', None, *phrase_patterns)
```

```
#Apply the matcher to our Doc object:
matches = matcher(doc)
```

```
#See what matches occur:
matches
#output: [(2689272359382549672, 7, 9), (2689272359382549672, 14,
16)]
```

```
doc = nlp(u'Our company plans to introduce a new vacuum cleaner. '
          u'If successful, the vacuum cleaner will be our first product.')

show_ents(doc)
```

```
first - 99 - 104 - ORDINAL - "first", "second", etc.
```

```
# Import PhraseMatcher and create a matcher object:
from spacy.matcher import PhraseMatcher
matcher = PhraseMatcher(nlp.vocab)
```

```
# Create the desired phrase patterns:
phrase_list = ['vacuum cleaner', 'vacuum-cleaner']
phrase_patterns = [nlp(text) for text in phrase_list]
```

```
# Apply the patterns to our matcher object:
matcher.add('newproduct', None, *phrase_patterns)

# Apply the matcher to our Doc object:
matches = matcher(doc)

# See what matches occur:
matches
```

```
[(2689272359382549672, 7, 9), (2689272359382549672, 14, 16)]
```

```
#Here we create Spans from each match, and create named entities
from them:
from spacy.tokens import Span
PROD = doc.vocab.strings[u'PRODUCT']
new_ents = [Span(doc, match[1],match[2],label=PROD) for match in
matches]
#match[1] contains the start index of the the token and match[2]
the stop index (exclusive) of the token in the doc.
doc.ents = list(doc.ents) + new_ents
show_ents(doc)
```

```
output: vacuum cleaner - 37 - 51 - PRODUCT - Objects, vehicles,
foods, etc. (not services) vacuum cleaner - 72 - 86 - PRODUCT -
Objects, vehicles, foods, etc. (not services) first - 99 - 104 -
ORDINAL - "first", "second", etc.
```

```
# Here we create Spans from each match, and create named entities from them:
from spacy.tokens import Span

PROD = doc.vocab.strings[u'PRODUCT']

new_ents = [Span(doc, match[1],match[2],label=PROD) for match in matches]
# match[1] contains the start index of the the token and match[2] the stop index (exclusive) of the token in the doc.

doc.ents = list(doc.ents) + new_ents
```

```
show_ents(doc)
```

```
vacuum cleaner - 37 - 51 - PRODUCT - Objects, vehicles, foods, etc. (not services)
vacuum cleaner - 72 - 86 - PRODUCT - Objects, vehicles, foods, etc. (not services)
first - 99 - 104 - ORDINAL - "first", "second", etc.
```

Counting Entities

While spaCy may not have a built-in tool for counting entities, we can pass a conditional statement into a list comprehension:

```
In [33]: doc = nlp(u'Originally priced at $29.50, the sweater was marked down to five dollars.')

         show_ents(doc)

         29.50 - 22 - 27 - MONEY - Monetary values, including unit
         five dollars - 60 - 72 - MONEY - Monetary values, including unit
```

```
In [34]: len([ent for ent in doc.ents if ent.label_=='MONEY'])

Out[34]: 2
```

Visualizing NER

Spacy has a library called “displaCy” which helps us to explore the behaviour of the entity recognition model interactively.

If you are training a model, it’s very useful to run the visualization yourself.

You can pass a `Doc` or a list of `Doc` objects to displaCy and run `displacy.serve` to run the web server, or `displacy.render` to generate the raw mark-up.

```
#Import the displaCy library
from spacy import displacy
```

Visualizing Sentences Line by Line

Viewing Specific Entities

You can pass a list of entity types to restrict the visualization:

Styling: customize color and effects

You can also pass background color and gradient options:

This is all about Named Entity Recognition NER and its Visualization using spaCy. Hope you enjoyed the post.

Next Article I will describe about Sentence Segmentation. Stay Tuned!

If you have any feedback to improve the content or any thought please write in the comment section below. Your comments are very valuable.

Previous Articles in spaCy NLP Series:

- SPACY INSTALLATION AND BASIC OPERATIONS | NLP TEXT PROCESSING LIBRARY | PART 1

- A QUICK GUIDE TO TOKENIZATION, LEMMATIZATION, STOP WORDS, AND PHRASE MATCHING USING SPACY | NLP | PART 2
- PARTS OF SPEECH TAGGING AND DEPENDENCY PARSING USING SPACY | NLP | PART 3

Thank You!

Post Credit: Jose Portila Udemy Videos

« Parts of Speech Tagging and Dependency Parsing using spaCy | NLP | Part 3

How to Perform Sentence Segmentation or Sentence Tokenization using spaCy | NLP Series | Part 5 »

2 COMMENTS

Pingback: A Quick Guide to Tokenization, Lemmatization, Stop Words, and Phrase Matching using spaCy | NLP | Part 2 – Data Science, Machine Learning & Artificial Intelligence

Pingback: Numerical Feature Extraction from Text | NLP series | Part 6 – Data Science, Machine Learning & Artificial Intelligence

LEAVE A REPLY