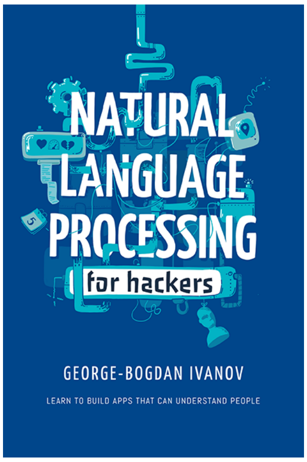
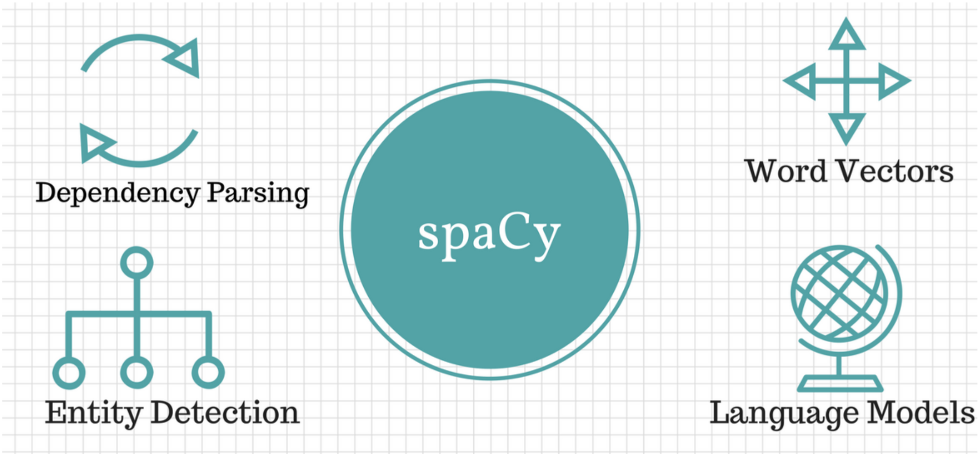
Natural Language Processing FOR HACKERS





Complete Guide to spaCy

Updates

- **29-Apr-2018** – Fixed import in extension code (Thanks Ruben)

spaCy is a relatively new framework in the Python Natural Language Processing environment but it quickly gains ground and will most likely become the de facto library. There are some really good reasons for its popularity:

It's really FAST

Written in Cython, it was specifically designed to be as fast as possible

It's really ACCURATE

spaCy implementation of its dependency parser is one of the best-performing in the world:
It Depends: Dependency Parser

Complete Guide to Topic Modeling

Complete Guide to spaCy

Complete guide to build your own Named Entity Recognizer with Python

Splitting text into sentences

Recipe: Text clustering using NLTK and scikit-learn

TOPICS

Deep Learning (4)

Deep Natural Language Processing (4)

Machine Learning (5)

NLTK (11)

Resources (2)

Scikit-Learn (8)

spaCy (2)

Text Mining (4)

Uncategorized (2)

Wordnet (3)

Batteries included

- **Index preserving tokenization** (details about this later)
- Models for *Part Of Speech tagging*, *Named Entity Recognition* and *Dependency Parsing*
- Supports **8 languages** out of the box
- Easy and **beautiful visualizations**
- Pretrained **word vectors**

Extensible

It plays nicely with all the other already existing tools that you know and love: **Scikit-Learn**, **TensorFlow**, **gensim**

DeepLearning Ready

It also has its own deep learning framework that's especially designed for NLP tasks: Thinc

Quickstart

spaCy is easy to install:

```
pip install -U spaCy
python -m spacy download en

```

Notice that the installation doesn't automatically download the English model. We need to do that ourselves.

```
for token in doc:
    print('"' + token.text + '"')

# "Hello"
# "     "
# "World"
# "!"

```

Notice the index preserving tokenization in action. Rather than only keeping the words, spaCy keeps the spaces too. This is helpful for situations when you need to replace words in the original text or add some annotations. With NLTK tokenization, there's no way to know exactly where a tokenized word is in the original raw text. spaCy preserves this *"link"* between the word and its place in the raw text. Here's how to get the exact index of a word:

```
import spacy
nlp = spacy.load('en')
doc = nlp('Hello     World!')
for token in doc:
    print('"' + token.text + '"', token.idx)

# "Hello" 0
# "     " 6
# "World" 10
# "!" 15

```

The `Token` class exposes a lot of word-level attributes. Here are a few examples:

```
doc = nlp("Next week I'll   be in Madrid.")
for token in doc:
    print("{0}\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}\t{
        token.text,
        token.idx,
        token.lemma_,
        token.is_punct,
        token.is_space,
        token.shape_,
        token.pos_,
        token.tag_
    ))
```

```
# I 10  -PRON-  False   False   X   PRON    PRP
# 'll   11  will    False   False   'xx VERB
#       15      False   True        SPACE   _SP
# be    17  be  False   False   xx  VERB    VB
# in    20  in  False   False   xx  ADP IN
# Madrid    23  madrid  False   False   Xxxxx
# . 29  .   True    False   .   PUNCT   .

```

The spaCy toolbox

Let's now explore what are the models bundled up inside spaCy.

Sentence detection

Here's how to achieve one of the most common NLP tasks with spaCy:

```
doc = nlp("These are apples. These are oranges.

for sent in doc.sents:
    print(sent)

# These are apples.
# These are oranges.

```

Part Of Speech Tagging

We've already seen how this works but let's have another look:

```
doc = nlp("Next week I'll be in Madrid.")
print([(token.text, token.tag_) for token in do

# [('Next', 'JJ'), ('week', 'NN'), ('I', 'PRP')

```

Named Entity Recognition

Doing NER with spaCy is super easy and the pretrained model performs pretty well:

```
doc = nlp("Next week I'll be in Madrid.")
for ent in doc.ents:
    print(ent.text, ent.label_)

# Next week DATE
```

You can also view the IOB style tagging of the sentence like this:

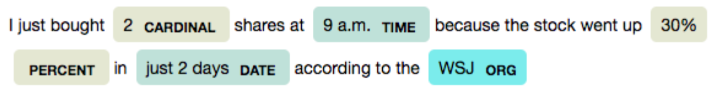
```
from nltk.chunk import conlltags2tree


doc = nlp("Next week I'll be in Madrid.")
iob_tagged = [
    (
        token.text,
        token.tag_,
        "{0}-{1}".format(token.ent_iob_, token.
    ) for token in doc
]

print(iob_tagged)

# In case you like the nltk.Tree format
print(conlltags2tree(iob_tagged))

# [('Next', 'JJ', 'B-DATE'), ('week', 'NN', 'I-

# (S
#   (DATE Next/JJ week/NN)
#   I/PRP
#   'll/MD
#   be/VB
#   in/IN
#   (GPE Madrid/NNP)
#   ./.)

```

The spaCy NER also has a healthy variety of entities. You can view the full list here: Entity Types

```
doc = nlp("I just bought 2 shares at 9 a.m. bec
for ent in doc.ents:
    print(ent.text, ent.label_)

# 2 CARDINAL
# 9 a.m. TIME
# 30% PERCENT
# just 2 days DATE
# WSJ ORG

```

Let's use `displaCy` to view a beautiful visualization of the Named Entity annotated sentence:

```
from spacy import displacy
```



Chunking

spaCy automatically detects noun-phrases as well:

```
doc = nlp("Wall Street Journal just published a
for chunk in doc.noun_chunks:
    print(chunk.text, chunk.label_, chunk.root.

# Wall Street Journal NP Journal
# an interesting piece NP piece
# crypto currencies NP currencies

```

Notice how the chunker also computes the root of the phrase, the main word of the phrase.

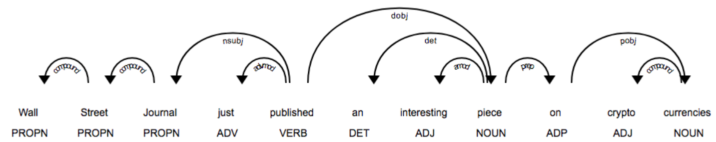
Dependency Parsing

This is what makes spaCy really stand out. Let's see the dependency parser in action:

```
doc = nlp('Wall Street Journal just published a

for token in doc:
    print("{0}/{1} <--{2}-- {3}/{4}".format(
        token.text, token.tag_, token.dep_, tok

# Wall/NNP <--compound-- Street/NNP
# Street/NNP <--compound-- Journal/NNP
# Journal/NNP <--nsubj-- published/VBD
# just/RB <--advmod-- published/VBD
# published/VBD <--ROOT-- published/VBD
# an/DT <--det-- piece/NN
# interesting/JJ <--amod-- piece/NN
# piece/NN <--dobj-- published/VBD
# on/IN <--prep-- piece/NN
# crypto/JJ <--compound-- currencies/NNS
# currencies/NNS <--pobj-- on/IN

```

dependency tree, displacy comes in handy:

```
from spacy import displacy

doc = nlp('Wall Street Journal just published a
displacy.render(doc, style='dep', jupyter=True,

```



Word Vectors

spaCy comes shipped with a Word Vector model as well. We'll need to download a larger model for that:

```
python -m spacy download en_core_web_lg

```

The vectors are attached to spaCy objects: `Token`, `Lexeme` (a sort of unnatached token, part of the vocabulary), `Span` and `Doc`. The multi-token objects average its constituent vectors.

Explaining word vectors(aka word embeddings) are not the purpose of this tutorial. Here are a few properties word vectors have:

- If two words are similar, they appear in similar contexts
- Word vectors are computed taking into account the context (surrounding words)

similar words should have similar word vectors

- Using vectors we can derive relationships between words

Let's see how we can access the embedding of a word in spaCy:

```
nlp = spacy.load('en_core_web_lg')
print(nlp.vocab['banana'].vector)

```

There's a really famous example of word embedding math: `"man" - "woman" + "queen" = "king"`. It sounds pretty crazy to be true, so let's test that out:

```
from scipy import spatial

cosine_similarity = lambda x, y: 1 - spatial.di

man = nlp.vocab['man'].vector
woman = nlp.vocab['woman'].vector
queen = nlp.vocab['queen'].vector
king = nlp.vocab['king'].vector

# We now need to find the closest vector in the
maybe_king = man - woman + queen
computed_similarities = []

for word in nlp.vocab:
    # Ignore words without vectors
    if not word.has_vector:
        continue

    similarity = cosine_similarity(maybe_king,
    computed_similarities.append((word, similar

computed_similarities = sorted(computed_similar
print([w[0].text for w in computed_similarities

# ['Queen', 'QUEEN', 'queen', 'King', 'KING', '

```

Surprisingly, the closest word vector in the vocabulary for "man" – "woman" + "queen"

Maybe behind every King is a Queen.

Computing Similarity

Based on the word embeddings, spaCy offers a similarity interface for all of it's building blocks: `Token`, `Span`, `Doc` and `Lexeme`. Here's how to use that similarity interface:

```
banana = nlp.vocab['banana']
dog = nlp.vocab['dog']
fruit = nlp.vocab['fruit']
animal = nlp.vocab['animal']

print(dog.similarity(animal), dog.similarity(fr
print(banana.similarity(fruit), banana.similari

```

Let's now use this technique on entire texts:

```
target = nlp("Cats are beautiful animals.")

doc1 = nlp("Dogs are awesome.")
doc2 = nlp("Some gorgeous creatures are felines
doc3 = nlp("Dolphins are swimming mammals.")

print(target.similarity(doc1))  # 0.89017652184
print(target.similarity(doc2))  # 0.91158284491
print(target.similarity(doc3))  # 0.78229567528

```

Extending spaCy

The entire spaCy architecture is built upon three building blocks: `Document` (the big encompassing container), `Token` (most of the time, a word) and `Span` (set of consecutive `Tokens`). The extensions you create can add extra functionality to anyone of the these components. There are some examples out there for what you can do. Let's create an extension ourselves.

Extension

```
import spacy
from spacy.tokens import Doc
from nltk.sentiment.vader import SentimentInten

sentiment_analyzer = SentimentIntensityAnalyzer
def polarity_scores(doc):
    return sentiment_analyzer.polarity_scores(d

Doc.set_extension('polarity_scores', getter=pol

nlp = spacy.load('en')
doc = nlp("Really Whaaat event apple nice! it!"
print(doc._.polarity_scores)
# {'neg': 0.0, 'neu': 0.596, 'pos': 0.404, 'com

```

One can easily create extensions for every component type. Such extensions only have access to the context of that component. What happens if you need the tokenized text along with the Part-Of-Speech tags. Let's now build a custom `pipeline`. Pipelines are another important abstraction of spaCy. The `nlp` object goes through a list of pipelines and runs them on the document. For example the `tagger` is ran first, then the `parser` and `ner` pipelines are applied on the already POS annotated document. Here's how the `nlp` default pipeline structure looks like:

```
nlp = spacy.load('en')
print(nlp.pipeline)

# [('tagger', <spacy.pipeline.Tagger object at
#  ('parser', <spacy.pipeline.DependencyParser
#  ('ner', <spacy.pipeline.EntityRecognizer obj

```

Creating a custom pipeline

Let's build a custom pipeline that needs to be applied after the `tagger` pipeline is ran.

```
from nltk.corpus import wordnet as wn
from spacy.tokens import Token


def penn_to_wn(tag):
    if tag.startswith('N'):
        return 'n'

    if tag.startswith('V'):
        return 'v'

    if tag.startswith('J'):
        return 'a'

    if tag.startswith('R'):
        return 'r'

    return None


class WordnetPipeline(object):
    def __init__(self, nlp):
        Token.set_extension('synset', default=N

    def __call__(self, doc):
        for token in doc:
            wn_tag = penn_to_wn(token.tag_)
            if wn_tag is None:
                continue

            ss = wn.synsets(token.text, wn_tag)
            token._.set('synset', ss)

        return doc


nlp = spacy.load('en')
wn_pipeline = WordnetPipeline(nlp)
nlp.add_pipe(wn_pipeline, name='wn_synsets')
doc = nlp("Paris is the awesome capital of Fran

for token in doc:
    print(token.text, "-", token._.synset)

# Paris - Synset('paris.n.01')
# is - Synset('be.v.01')
# the - None
# awesome - Synset('amazing.s.02')
# capital - Synset('capital.n.01')
# of - None
# France - Synset('france.n.01')
# . - None

```

Let’s see how the pipeline structure looks like:

```
print(nlp.pipeline)

```

```
# ('wn_synsets', <__main__.WordnetPipeline obj
```

Conclusions

spaCy is a modern, reliable NLP framework that quickly became the standard for doing NLP with Python. Its main advantages are: speed, accuracy, extensibility. It also comes shipped with useful assets like word embeddings. It can act as the central part of your production NLP pipeline.

Related

Roundup of Python NLP Libraries
July 17, 2018
In "Resources"

What is Natural Language Processing?
November 30, 2016
In "general"

Complete Guide to Word Embeddings
April 6, 2018
In "Deep Learning"

DATE
March 28, 2018

AUTHOR
bogdani

CATEGORY
spaCy

TAG
embeddings
framework
IOB-tagging
part-of-speech
similarity
spaCy
tagging
text analysis
word vectors
wordnet

COMMENTS
18 Comments

18 Comments

Liza
03/29/2018 - 9:30 AM

Hi Bogdan,

I was playing around with spaCy and I have a problem with Matcher, it doesn't work… and I couldn't find any support…do I have to load it separetly? Or could it becaus of wrong setting?

and an other question (sorry I kinda have a lot…) I was reding your blog about tranning NER (an amazing tutorial thought), but do you have similar articals, if I want to build and train my own one (or may be some tips:))?

Thanks a lot!

REPLY

> bogdani
> 03/29/2018 - 12:52 PM
>
> Hi Liza,
>
> Can you maybe paste the Matcher code? I don't have much experience with it either.

missing? What do you need more for training your NER? Does the 2nd episode shed some light? https://nlpforhackers.io/training-ner-large-dataset/

Thanks,
Bogdan.

REPLY

ahmed mohammed
04/07/2018 - 3:00 AM

this is clear precise and informative thank you Bogdani

REPLY

Ruben
04/29/2018 - 9:49 AM

Hi Bogdani, great tutorial!
I've replicated it on a jupyter notebook, I just found a typo in line 22 in "Creating a custome pipeline", it should be

```
token.set_extension('synset', default=None)
```

instead of

```
Token.set_extension('synset', default=None)
```

Otherwise thanks for the article, it was very useful!

REPLY

bogdani
04/29/2018 - 9:49 PM

Hi Ruben,

I was missing this import `from spacy.tokens import Token`. Fixed, thanks for reporting this ☺

REPLY

Banu Selin Tosun
04/21/2019 - 5:41 PM

Hi,

I also needed another adjustment in the above mentioned snipped.

1st I needed to download wordnet as `nltk.download('wordnet')`
2nd at the above mentioned line, the token extensions should be set to force=True via `def __init__(self, nlp): Token.set_extension('synset', default=None, force=True)`

Everything works after that.
I am not sure if this is system

libraries are all up to date.
Maybe a newer issue in the new version.

Thank you,
Selin

REPLY

transhuaman
06/07/2018 - 10:24 AM

simple witing with high information. great!

best npl training

REPLY

dat
06/21/2018 - 9:24 AM

great article, thank for sharing.

REPLY

bogdani
06/21/2018 - 9:35 AM

Thank you, Sir ☺

REPLY

Archana
10/29/2018 - 10:10 AM

REPLY

bogdani
10/29/2018 - 11:32 AM

Unfortunately, no

REPLY

Arpit
12/18/2018 - 8:23 PM

Hello Bogdani,

its one of the best tutorial for SpaCy specially adding the pipeline part.

I was having a doubt relating to the .similarity function in SpaCy. Suppose when comparing two sentences does it consider the POS tagging and parsing pipelines?? I doubt it happens because it uses GloVe vector representations which does not support the POS tagging etc.

Do you have any ideas how can i use the parts of speech and dependency parsing features (like provided by spacy) in word vectors models ??

Thanks

REPLY

bogdani
12/20/2018 - 12:03 PM

cosine similarity between the mean vectors of the 2 sentences. Because embeddings don't take into account the POS, the similarity function won't take that into account either. I have a tutorial on a different similarty function here: https://nlpforhackers.io/wordnet-sentence-similarity/

REPLY

Arpit
12/19/2018 - 8:03 PM

Can you explain how the similarity function of SpaCy works ? Does it use the tagging and dependency parsing information into account when finding the similarity score ?

REPLY

bogdani
12/20/2018 - 12:17 PM

Think it's the cosine similarity between the mean of the word vectors

REPLY

Kalyan
12/20/2018 - 3:13 AM

output of dependency parser using displacy, I got this error

```
1 <span class="ansi-red-intense-fg ansi-bold">Typ
```

REPLY

bogdani
12/20/2018 - 12:17 PM

What Python version are you using?

REPLY

cristo
06/03/2019 - 3:21 PM

spacy is only a shit tool...if You need faster tokenizer go with nltk...spacy
become uselessly slow
and cumbersome..please; ps. package like tm in R, text2vec, and others works 100 time better than this crock...

REPLY

YOUR EMAIL ADDRESS WILL NOT BE PUBLISHED. REQUIRED FIELDS ARE MARKED *

Visual Text

Paragraph

NAME *

EMAIL *

WEBSITE

Post Comment

NOTIFY ME OF FOLLOW-UP COMMENTS BY EMAIL.

NOTIFY ME OF NEW POSTS BY EMAIL.