## >>> Python for NLP: Vocabulary and Phrase Matching with SpaCy

>>> By •

>>> This is the third article in this series of articles on Python for   
 NLTK Python's how saw we , the In .Processing Language Natural <<<   
 >>> and libraries can be used to perform simple NLP tasks such as , . We   
 entity named ,tagging speech of parts perform to how saw also <<<   
 >>> recognition and noun-parsing. However, all of these operations are   
 .words individual on performed <<<

>>> In this article, we will move a step further and explore vocabulary   
 patterns define will We .library spaCy the using matching phrase and <<<   
 >>> and then will see which phrases that match the pattern we define. This   
 of parts involve that expressions regular defining to similar is <<<   
 >>> speech.

### >>> Rule-Based Matching

>>> The spaCy library comes with Matcher tool that can be used to specify   
 tool Matcher the use to process The .matching phrase for rules custom <<<   
 >>> is pretty straight forward. The first thing you have to do is define   
 the add to have you ,Next .match to want you that patterns the <<<   
 >>> patterns to the Matcher tool and finally, you have to apply   
 rules your match to want you that document the to tool Matcher the <<<   
 >>> with. This is best explained with the help of an example.

>>> For rule-based matching, you need to perform the following steps:

#### >>> Creating Matcher Object

>>> The first step is to create the matcher object:

>>> import spacy

>>> nlp = spacy.load('en\_core\_web\_sm')

>>> from spacy.matcher import Matcher

>>> m\_tool = Matcher(nlp.vocab)

#### >>> Defining Patterns

>>> The next step is to define the patterns that will be used to filter   
 -"quick-brown phrases the find to want we Suppose .phrases similar <<<   
 >>> fox", "quick brown fox", "quickbrownfox" or "quick brownfox". To do   
 :patterns four following the create to need we ,so <<<

>>> p1 = [{'LOWER': 'quickbrownfox'}]

>>> p2 = [{'LOWER': 'quick'}, {'IS\_PUNCT': True}, {'LOWER': 'brown'},   
 ['fox'} :{'LOWER' ,True} :{'IS\_PUNCT' <<<

>>> p3 = [{'LOWER': 'quick'}, {'LOWER': 'brown'}, {'LOWER': 'fox'}]

>>> p4 = [{'LOWER': 'quick'}, {'LOWER': 'brownfox'}]

>>> In the above script,

* >>> p1 looks for the phrase "quickbrownfox"
* >>> p2 looks for the phrase "quick-brown-fox"
* >>> p3 tries to search for "qucik brown fox"
* >>> p4 looks for the phrase "quick brownfox"

>>> The token attribute LOWER defines that the phrase should be converted   
 .matching before case lower into <<<

>>> Once the patterns are defined, we need to add them to   
 .earlier created we that object Matcher the <<<

>>> m\_tool.add('QBF', None, p1, p2, p3, p4)

>>> Here "QBF" is the name of our matcher. You can give it any name.

#### >>> Applying Matcher to the Document

>>> We have our matcher ready. The next step is to apply the matcher on a   
 a create first Let's .match any get can we if see and document text <<<   
 >>> simple document:

>>> sentence = nlp(u'The quick-brown-fox jumps over the lazy dog. The   
 \ .well eats fox brown quick <<<

>>> the quickbrownfox is dead. the dog misses the quick   
 (brownfox' <<<

>>> To apply the matcher to a document. The document is needed to be   
 all be will result The .object matcher the to parameter a as passed <<<   
 >>> the ids of the phrases matched in the document, along with their   
 following the Execute .document the in positions ending and starting <<<   
 >>> script:

>>> phrase\_matches = m\_tool(sentence)

>>> print(phrase\_matches )

>>> The output of the script above looks like this:

>>> [(12825528024649263697, 1, 6), (12825528024649263697, 13, 16),   
 (31) ,29 ,(12825528024649263697 ,22) ,21 ,(12825528024649263697 <<<

>>> From the output, you can see that four phrases have been matched. The   
 the ,matched phrase the of id the is output each in number long first <<<   
 >>> second and third numbers are the starting and ending positions of the   
 .phrase <<<

>>> To actually view the result in a better way, we can iterate through   
 the Execute .value string its display and phrase matched each <<<   
 >>> following script:

>>> for match\_id, start, end in phrase\_matches:

>>> string\_id = nlp.vocab.strings[match\_id]

>>> span = sentence[start:end]

>>> print(match\_id, string\_id, start, end, span.text)

>>> Output:

>>> 12825528024649263697 QBF 1 6 quick-brown-fox

>>> 12825528024649263697 QBF 13 16 quick brown fox

>>> 12825528024649263697 QBF 21 22 quickbrownfox

>>> 12825528024649263697 QBF 29 31 quick brownfox

>>> From the output, you can see all the matched phrases along with their   
 .position ending and start and ids vocabulary <<<

#### >>> More Options for Rule-Based Matching

>>> Official documentation from the sPacy library contains details of all   
 .matching phrase for used be can that the <<<

>>> For instance, the "\*" attribute is defined to search for one or more   
 .token the of instances <<<

>>> Let's write a simple pattern that can identify the phrase "quick--   
 .quick-brown---fox or "brown--fox <<<

>>> Let's first remove the previous matcher QBF.

>>> m\_tool.remove('QBF')

>>> Next, we need to define our new pattern:

>>> p1 = [{'LOWER': 'quick'}, {'IS\_PUNCT': True, 'OP':'\*'}, {'LOWER':   
 ['fox'} :{'LOWER' ,'OP':'\*'} ,True :{'IS\_PUNCT' ,'brown'} <<<

>>> m\_tool.add('QBF', None, p1)

>>> The pattern p1 will match all the phrases where there are one or more   
 our define now Let's .fox brown quick phrase the in punctuations <<<   
 >>> document for filtering:

>>> sentence = nlp(u'The quick--brown--fox jumps over the quick-brown---   
 (fox' <<<

>>> You can see our document has two phrases quick--brown--fox and quick-   
 mather our apply Let's .pattern our match should you that ,brown---fox <<<   
 >>> to the document and see the results:

>>> phrase\_matches = m\_tool(sentence)

>>> for match\_id, start, end in phrase\_matches:

>>> string\_id = nlp.vocab.strings[match\_id]

>>> span = sentence[start:end]

>>> print(match\_id, string\_id, start, end, span.text)

>>> The output of the script above looks like this:

>>> 12825528024649263697 QBF 1 6 quick--brown--fox

>>> 12825528024649263697 QBF 10 15 quick-brown---fox

>>> From the output, you can see that our matcher has successfully matched   
 .phrases two the <<<

#### >>> Subscribe to our Newsletter

>>> Top of Form

>>> Get occassional tutorials, guides, and jobs in your inbox. No spam   
 .time any at Unsubscribe .ever <<<

>>> Newsletter Signup

>>> Subscribe

>>> Bottom of Form

### >>> Phrase-Based Matching

>>> In the last section, we saw how we can define rules that can be used   
 ,rules defining to addition In .document the from phrases identify to <<<   
 >>> we can directly specify the phrases that we are looking for. This is a   
 .matching phrase of way efficient more <<<

>>> In this section, we will be doing phrase matching inside a Wikipedia   
 .intelligence Artificial on article <<<

>>> Before we see the steps to perform phrase-matching, let's first parse   
 phrase perform to using be will we that article Wikipedia the <<<   
 >>> matching. Execute the following script:

>>> import bs4 as bs

>>> import urllib.request

>>> import re

>>> import nltk

>>> scrapped\_data = urllib.request.urlopen('https://en.wikipedia.org/wiki/   
 (Artificial\_intelligence' <<<

>>> article = scrapped\_data .read()

>>> parsed\_article = bs.BeautifulSoup(article,'lxml')

>>> paragraphs = parsed\_article.find\_all('p')

>>> article\_text = ""

>>> for p in paragraphs:

>>> article\_text += p.text

>>> processed\_article = article\_text.lower()

>>> processed\_article = re.sub('[^a-zA-Z]', ' ', processed\_article )

>>> processed\_article = re.sub(r'\s+', ' ', processed\_article)

>>> The script has been explained in detail in my article on You can go   
 in works parsing how understand to want you if article the read and <<<   
 >>> Python.

>>> The processed\_article contains the document that we will use for   
 .phrase-matching <<<

>>> The steps to perform phrase matching are quite similar to rule based   
 .matching <<<

#### >>> Create Phrase Matcher Object

>>> As a first step, you need to create PhraseMatcher object. The   
 :that does script following <<<

>>> import spacy

>>> nlp = spacy.load('en\_core\_web\_sm')

>>> from spacy.matcher import PhraseMatcher

>>> phrase\_matcher = PhraseMatcher(nlp.vocab)

>>> Notice in the previous section we created Matcher object. Here, in   
 .object PhraseMathcer creating are we ,case this <<<

#### >>> Create Phrase List

>>> In the second step, you need to create a list of phrases to match and   
 following the in shown as documents NLP spaCy to list the convert then <<<   
 >>> script:

>>> phrases = ['machine learning', 'robots', 'intelligent agents']

>>> patterns = [nlp(text) for text in phrases]

>>> Finally, you need to add your phrase list to the phrase matcher.

>>> phrase\_matcher.add('AI', None, \*patterns)

>>> Here the name of our matcher is AI.

#### >>> Applying Matcher to the Document

>>> Like rule-based matching, we again need to apply our phrase matcher to   
 document spaCy in not is article parsed our ,However .document the <<<   
 >>> format. Therefore, we will convert our article into sPacy document   
 .article the to matcher phrase our apply then will and format <<<

>>> sentence = nlp (processed\_article)

>>> matched\_phrases = phrase\_matcher(sentence)

>>> In the output, we will have all the ids of all the matched phrases   
 :below shown as document the in indexes end and start their with along <<<

>>> [(5530044837203964789, 37, 39),

>>> (5530044837203964789, 402, 404),

>>> (5530044837203964789, 693, 694),

>>> (5530044837203964789, 1284, 1286),

>>> (5530044837203964789, 3059, 3061),

>>> (5530044837203964789, 3218, 3220),

>>> (5530044837203964789, 3753, 3754),

>>> (5530044837203964789, 5212, 5213),

>>> (5530044837203964789, 5287, 5288),

>>> (5530044837203964789, 6769, 6771),

>>> (5530044837203964789, 6781, 6783),

>>> (5530044837203964789, 7496, 7498),

>>> (5530044837203964789, 7635, 7637),

>>> (5530044837203964789, 8002, 8004),

>>> (5530044837203964789, 9461, 9462),

>>> (5530044837203964789, 9955, 9957),

>>> (5530044837203964789, 10784, 10785),

>>> (5530044837203964789, 11250, 11251),

>>> (5530044837203964789, 12290, 12291),

>>> (5530044837203964789, 12411, 12412),

>>> (5530044837203964789, 12455, 12456)]

>>> To see the string value of the matched phrases, execute the following   
 :script <<<

>>> for match\_id, start, end in matched\_phrases:

>>> string\_id = nlp.vocab.strings[match\_id]

>>> span = sentence[start:end]

>>> print(match\_id, string\_id, start, end, span.text)

>>> In the output, you will see the strig value of the matched phrases as   
 :below shown <<<

>>> 5530044837203964789 AI 37 39 intelligent agents

>>> 5530044837203964789 AI 402 404 machine learning

>>> 5530044837203964789 AI 693 694 robots

>>> 5530044837203964789 AI 1284 1286 machine learning

>>> 5530044837203964789 AI 3059 3061 intelligent agents

>>> 5530044837203964789 AI 3218 3220 machine learning

>>> 5530044837203964789 AI 3753 3754 robots

>>> 5530044837203964789 AI 5212 5213 robots

>>> 5530044837203964789 AI 5287 5288 robots

>>> 5530044837203964789 AI 6769 6771 machine learning

>>> 5530044837203964789 AI 6781 6783 machine learning

>>> 5530044837203964789 AI 7496 7498 machine learning

>>> 5530044837203964789 AI 7635 7637 machine learning

>>> 5530044837203964789 AI 8002 8004 machine learning

>>> 5530044837203964789 AI 9461 9462 robots

>>> 5530044837203964789 AI 9955 9957 machine learning

>>> 5530044837203964789 AI 10784 10785 robots

>>> 5530044837203964789 AI 11250 11251 robots

>>> 5530044837203964789 AI 12290 12291 robots

>>> 5530044837203964789 AI 12411 12412 robots

>>> 5530044837203964789 AI 12455 12456 robots

>>> From the output, you can see all the three phrases that we tried to   
 .ids string the and index end and start their with along search <<<

### >>> Stop Words

>>> Before we conclude this article, I just wanted to touch the concept of   
 etc ""an ,"a" ,"the" as such words English are words Stop .words stop <<<   
 >>> that do not have any meaning of their own. Stop words are often not   
 language or classification text as such tasks NLP for useful very <<<   
 >>> modeling. So it is often better to remove these stop words before   
 .document the of processing further <<<

>>> The spaCy library contains 305 stop words. In addition, depending upon   
 spaCy the from words stop remove or add also can we ,requirements our <<<   
 >>> library.

>>> To see the default spaCy stop words, we can use stop\_words attribute   
 :below shown as model spaCy the of <<<

>>> import spacy

>>> sp = spacy.load('en\_core\_web\_sm')

>>> print(sp.Defaults.stop\_words)

>>> In the output, you will see all the sPacy stop words:

>>> {'less', 'except', 'top', 'me', 'three', 'fifteen', 'a', 'is',   
 ,'any' ,'has' ,'must' ,'without' ,'everyone' ,'then' ,'all' ,'those' <<<   
 >>> 'anyhow', 'keep', 'through', 'bottom', 'get', 'indeed', 'it', 'still',   
 ,'myself' ,'various' ,'eight' ,'though' ,'doing' ,'whatever' ,'ten' <<<   
 >>> 'across', 'wherever', 'himself', 'always', 'thus', 'am', 'after',   
 ,'regarding' ,'rather' ,'own' ,'down' ,'at' ,'perhaps' ,'should' <<<   
 >>> 'which', 'anywhere', 'whence', 'would', 'been', 'how', 'herself',   
 ,'from' ,'alone' ,'seems' ,'every' ,'behind' ,'please' ,'might' ,'now' <<<   
 >>> 'via', 'its', 'become', 'hers', 'there', 'front', 'whose', 'before',   
 ,'eleven' ,'five' ,'two' ,'whither' ,'up' ,'whereafter' ,'against' <<<   
 >>> 'why', 'below', 'out', 'whereas', 'serious', 'six', 'give', 'also',   
 ,'have' ,'else' ,'onto' ,'again' ,'none' ,'anyway' ,'his' ,'became' <<<   
 >>> 'few', 'thereby', 'whoever', 'yet', 'part', 'just', 'afterwards',   
 ,'therefore' ,'once' ,'can' ,'not' ,'hereby' ,'see' ,'mostly' <<<   
 >>> 'together', 'whom', 'elsewhere', 'beforehand', 'themselves', 'with',   
 ,'becoming' ,'who' ,'are' ,'former' ,'upon' ,'many' ,'seem' <<<   
 >>> 'formerly', 'between', 'cannot', 'him', 'that', 'first', 'more',   
 ,'whereupon' ,'my' ,'whereby' ,'under' ,'whenever' ,'although' <<<   
 >>> 'anyone', 'toward', 'by', 'four', 'since', 'amongst', 'move', 'each',   
 ,'when' ,'name' ,'if' ,'used' ,'besides' ,'as' ,'somehow' ,'forty' <<<   
 >>> 'ever', 'however', 'otherwise', 'hundred', 'moreover', 'your',   
 ,'enough' ,'her' ,'where' ,'another' ,'empty' ,'the' ,'sometimes' <<<   
 >>> 'quite', 'throughout', 'anything', 'she', 'and', 'does', 'above',   
 ,'re' ,'off' ,'nobody' ,'made' ,'back' ,'this' ,'in' ,'show' ,'within' <<<   
 >>> 'meanwhile', 'than', 'neither', 'twenty', 'call', 'you', 'next',   
 ,'latterly' ,'such' ,'seemed' ,'or' ,'go' ,'therein' ,'thereupon' <<<   
 >>> 'already', 'mine', 'yourself', 'an', 'amount', 'hereupon', 'namely',   
 ,'whole' ,'done' ,'be' ,'could' ,'yours' ,'of' ,'their' ,'same' <<<   
 >>> 'seeming', 'someone', 'these', 'towards', 'among', 'becomes', 'per',   
 ,'make' ,'well' ,'ours' ,'latter' ,'both' ,'beside' ,'beyond' ,'thru' <<<   
 >>> 'nowhere', 'about', 'were', 'others', 'due', 'yourselves', 'unless',   
 ,'something' ,'our' ,'everything' ,'most' ,'too' ,'even' ,'thereafter' <<<   
 >>> 'did', 'using', 'full', 'while', 'will', 'only', 'nor', 'often',   
 ,'very' ,'was' ,'along' ,'some' ,'over' ,'least' ,'being' ,'side' <<<   
 >>> 'on', 'into', 'nine', 'noone', 'several', 'i', 'one', 'third',   
 ,'either' ,'because' ,'whether' ,'here' ,'further' ,'but' ,'herein' <<<   
 >>> 'hereafter', 'really', 'so', 'somewhere', 'we', 'nevertheless',   
 ,'everywhere' ,'ca' ,'almost' ,'thence' ,'they' ,'had' ,'last' <<<   
 >>> 'itself', 'no', 'ourselves', 'may', 'wherein', 'take', 'around',   
 ,'twelve' ,'say' ,'what' ,'do' ,'until' ,'to' ,'them' ,'never' <<<   
 >>> 'nothing', 'during', 'sixty', 'sometime', 'us', 'fifty', 'much',   
 ['put' ,'he' ,'hence' ,'other' ,'for' <<<

>>> You can also check if a word is a stop word or not. To do so, you can   
 :below shown as attribute is\_stop the use <<<

>>> sp.vocab['wonder'].is\_stop

>>> Since "wonder" is not a spaCy stop word, you will see False in the   
 .output <<<

>>> To add or remove stopwords in spaCy, you can use sp.Defaults.stop\_word   
 .respectively methods (sp.Defaults.stop\_words.remove( and (s.add( <<<

>>> sp.Defaults.stop\_words.add('wonder')

>>> Next, we need to set the is\_stop tag for wonder to 'True` as shown   
 :below <<<

>>> sp.vocab['wonder'].is\_stop = True

### >>> Conclusion

>>> Phrase and vocabulary matching is one of the most important natural   
 our continued we ,article this In .tasks processing language <<<   
 >>> discussion about how to use Python to perform rule-based and phrase   
 .words stop spaCy saw also we ,addition In .matching based <<<

>>> In the , we will see parts of speech tagging and named entity   
 .detail in recognition <<<