Part of Speech (POS) Tagging and Chunking with NLTK

h2kinfosys.com/blog/part-of-speech-tagging-chunking-with-nltk

August 26, 2020

Artificial Intelligence Tutorials

What is Part of Speech

A part of speech POS Tagging is a classification system in the English Language that reveals the role a word plays in a context. There are eight parts of speech in the English Language: nouns, pronouns, verbs, adverbs, adjectives, prepositions, conjunctions, and interjections.

- Nouns are the name of places, animals, persons or things and examples include Tokyo, Elephant, Mike, vehicle
- Pronouns are words that replace a noun and examples include he, she, they, her, him, etc
- Verbs are action words are example include go, eat, drive, write, sleep, etc
- Adverbs are words that further describes a verb, adjective or another and example include quickly, boldly, often, poorly, yearly, etc
- Adjectives are words that further describes a noun or pronoun and example include quick, angry, new, cheerful, yellow, etc
- Propositions are words that relate nouns or pronouns with other words and examples include in, at, on, for, under, above, etc
- Conjunctions are words used to join words or phrases in a sentence and examples include and, while, but, nevertheless, because, etc
- Interjections are words used to convey emotions or surprises and examples include Wow, Really, Hurrah, Oh, Alas, etc.

Now that you know what each part of speech are, let's discuss Part of Speech Tagging

Part of Speech (POS) Tagging

POS tagging in simple terms means allocating every word in a sentence to a part of speech. NLTK has a method called pos_tag that performs POS tagging on a sentence. The methods apply supervised learning approaches that utilize features such as context, the capitulation of words, punctuations, and so on to determine the part of speech.

POS tagging is a critical procedure to understand the meaning of a sentence and know the relationship between words.

There are 35 POS tags in NLTK's pos_tag methods. The tags are shown in the table below

Tag	Abbreviation	Words
Coordinating Conjunction	CC	But, yet, although
Determiner	DT	A, An, The, This, My, Most
Cardinal Digit	CD	One, Two, Three, Forty
Existential There	EX	There
Foreign Word	FW	En masse, bona fide, et cetera, et al
Subordinating Conjunction or Preposition	IN	Over, Behind, Into
Adjective	JJ	Beautiful, Slow, New
Adjective, Comparative	JJR	Greater, Better, Older
Adjective, Superlative	JJS	Greatest, Best, Oldest
List Marker	LS	i, ii, iii, iv, …
Modal	MD	Have, Can, Shall
Noun, Singular	NN	School, Table, Pen
Noun, Plural	NNS	Schools, Tables, Pens
Proper Noun, Singular	NNP	Monday, Chicago, Mark
Proper Noun, Plural	NNPS	Koreans, Universities, Americans
Predeterminer	PDT	Both, All, The
Possessive Endings	POS	David’s, Dan’s, Francis’
Personal Pronoun	PRP	I, They, She
Possessive Pronoun	PRP$	His, Her, Their
Adverb	RB	Later, Very, Already
Adverb, Comparative	RBR	Better, More, Worse
Adverb, Superlative	RBS	Best, Most, Worst
Particle	RP	At, Across, About
To	TO	To

Verb, Base Form	VB	Jump, Eat, Play
Verb, Past Tense	VBD	Jumped, Ate, Played
Verb, Present Participle	VBG	Jumping, Eating, Playing
Verb, Past Participle	VBN	Taken, Given, Gone
Verb, Present Tense but not Third Person Singular	VBP	End, Go, Endure
Verb, Present Tense, Third Person Singular	VBZ	Jumps, Eats, Plays
Wh – Determiner	WDT	Which, What, Whichever
Wh – Pronouns	WP	Which, Whom, What
Possessive Wh – Pronoun	WP$	Whose
Wh – Adverb	WRB	Where, Why, When

Now that you know what the POS tags are, let's take a code example to demonstrate the steps involved in POS tagging

```
#import the nltk library
import nltk
#define a text
sentence = "The man was excited after he was informed about his promotion at work"
#tokenize the text
tokens = nltk.word_tokenize(sentence)

#Perform POS tagging
nltk.pos_tag(tokens)
```

Output:

```
[('The', 'DT'),
 ('man', 'NN'),
 ('was', 'VBD'),
 ('excited', 'VBN'),
 ('after', 'IN'),
 ('he', 'PRP'),
 ('was', 'VBD'),
 ('informed', 'VBN'),
 ('about', 'IN'),
 ('his', 'PRP$'),
 ('promotion', 'NN'),
 ('at', 'IN'),
 ('work', 'NN')]
```

You can also check for more information about a tag using the help.upenn_tagset() method. Say I have forgotten what JJ means, I can find out by typing this line of code

```
nltk.help.upenn_tagset("JJ")
```

Output:

```
JJ: adjective or numeral, ordinal
    third ill-mannered pre-war regrettable oiled calamitous first separable
    ectoplasmic battery-powered participatory fourth still-to-be-named
    multilingual multi-disciplinary ...
```

The code informs us that JJ means ‘adjective’ and went on to list some examples.

Chunking

Chunking can be defined as the process of extracting phrases or chunks of texts from unstructured texts. There are situations where a single word cannot encapsulate the complete meaning of a text. In such cases, chunks can be used to extract meaningful insights. In other words, chunking allows more flexibility in the extraction process.

Chunking works on top of POS tags such that it takes input from the POS tags and outputs the chunks. A common group of chunk tags is the noun phrase chunk (NP chunk). To create a noun phrase chunk, a chunk grammar is first defined using POS tags. This chunk grammar contains the rule with which the chunks would be created.

The rule is created using regular expressions and the following syntax

```
? means match 0 or 1 repetitions
* means match 0 or more repetitions
+ means match 1 or more
. means any character but not a new line
```

The POS tags and regular expressions are placed inside the < > placeholders. <RB.?> for instance would mean 0 or more of any adverbial tense. Let’s take a coding example to drive home our point.

```
#import the library
import nltk
#define the text
sentence = "I told the children I was going to tell them a story. They were excited"
#tokenize the text
tokens = nltk.word_tokenize(sentence)
#perform POS tagging
tags = nltk.pos_tag(tokens)
#define a chunk grammar named mychunk
chunk_grammar = """ mychunk: {<NNS.?>*<PRP.?>*<VBD?>}"""
#parse the grammar with regular expression parser
parser = nltk.RegexpParser(chunk_grammar)
#assign the chunk
tree = parser.parse(tags)
print(tree)
```

After getting the POS tags, the chunk grammar defined would select plural nouns with not more than 1 repetition, followed by personal pronouns with not more than 1 repetition, followed by the past tense verb with not more than 1 repetition, anywhere in the text. A RegexpParser was used to parse the chunk grammar. The POS tags were parsed with the parse() method to print the chunk. See the out output.

Output:

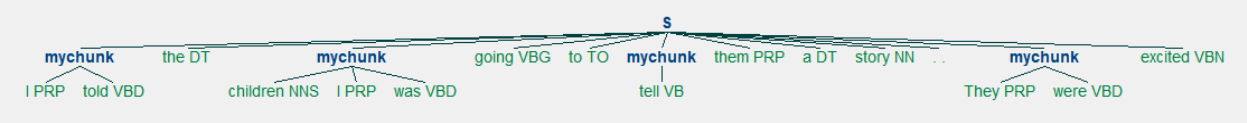
```
(S
  (mychunk I/PRP told/VBD)
  the/DT
  (mychunk children/NNS I/PRP was/VBD)
  going/VBG
  to/TO
  (mychunk tell/VB)
  them/PRP
  a/DT
  story/NN
  ./.
  (mychunk They/PRP were/VBD)
  excited/VBN)
```

As seen, “I told”, “Children I was”, “Tell” and “They were” were the selected chunk. To visualize the results better, you can use draw() method

```
tree.draw()
```

Output:

```
(S
  (mychunk I/PRP told/VBD)
  the/DT
  (mychunk children/NNS I/PRP was/VBD)
  going/VBG
  to/TO
  (mychunk tell/VB)
  them/PRP
  a/DT
  story/NN
  ./.
  (mychunk They/PRP were/VBD)
  excited/VBN)
(mychunk I/PRP told/VBD)
(mychunk children/NNS I/PRP was/VBD)
(mychunk tell/VB)
(mychunk They/PRP were/VBD)
```



Why is Chunking Important

Chunking can be used for entity detection. It can be used to extract word patterns from a large text without necessarily analyzing the entire text. Say you have a very large text and you only need to extract information such as name, item bought, and price of the item and the date. You can define a chunk grammar and use chunking to detect this pattern and extract the information quickly.

In summary, while it is good to use POS tagging to understand the role of a word in a sentence. Chunking helps to identify and understand phrases in a sentence and can perform rapid word filtering based on some defined grammar.

Facebook Comments