ajt2zw7nv

August 11, 2023

1 Word Boundary

[ ]:
```
import re

def find_word_boundaries(text):
    word_boundaries = re.finditer(r'\b\w+\b', text)
    for match in word_boundaries:
        print(match.group())

# Example text
text = "Hello all, how are you doing today?"

find_word_boundaries(text)
```

```
Hello
all
how
are
you
doing
today
```

2 Tokenization

[ ]:
```
import spacy

nlp = spacy.load("en_core_web_sm")

text = "Hello all, how are you doing today?"

doc = nlp(text)

for token in doc:
    print(token.text)
```

```
Hello
all
```

,
how
are
you
doing
today
?

Different types of tokenization

[ ]:
```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize, sent_tokenize, wordpunct_tokenize
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

[ ]:
```
# Word, Sentence, and Punctuation Tokenization example
text = "Hello, all! How's everyone doing? I hope you're having a great day.␣
 ↪Keep learning!"
print("Word Tokenization: ",word_tokenize(text))
print("Sentence Tokenization: ",sent_tokenize(text))
print("Punctuation Tokenizer: ",wordpunct_tokenize(text)) # It will consider␣
 ↪"'"(apostrophe) in How's as seperate word.
```

```
Word Tokenization:  ['Hello', ',', 'all', '!', 'How', "'s", 'everyone', 'doing',
'?', 'I', 'hope', 'you', "'re", 'having', 'a', 'great', 'day', '.', 'Keep',
'learning', '!']
Sentence Tokenization:  ['Hello, all!', "How's everyone doing?", "I hope you're
having a great day.", 'Keep learning!']
Punctuation Tokenizer:  ['Hello', ',', 'all', '!', 'How', "'", 's', 'everyone',
'doing', '?', 'I', 'hope', 'you', "'", 're', 'having', 'a', 'great', 'day', '.',
'Keep', 'learning', '!']
```

3 Stemming

[ ]:
```
# import the required modules
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

ps = PorterStemmer()

# choose some words to be stemmed
words = ["program", "programs", "programmer", "programmers", "programming"]

for w in words:
    print(w, ":",ps.stem(w))
```

```
program : program
programs : program
programmer : programm
programmers : programm
programming : program
```

- stem() is the function which is Stemming is a technique used to extract the base form of the words by removing affixes from them.
- It is just like cutting down the branches of a tree to its stems.
- For example, the stem of the words eating, eats, eaten is eat.

4 Lemmatization

[ ]:
```
# import these modules
import nltk
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')

lem = WordNetLemmatizer()

# choose some words to be stemmed
words = ["program", "programs", "programmer", "programmers", "programming"]

for w in words:
    print(w, ":",lem.lemmatize(w))
```

```
program : program
programs : program
programmer : programmer
programmers : programmer
programming : programming
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

5 Bigrams

[ ]:
```
#Code to generate bigram
import nltk
# nltk.download('punkt')

data = "The best performance can bring in sky high success."
tokens = nltk.word_tokenize(data)

print(list(nltk.bigrams(tokens)))
```

```
[('The', 'best'), ('best', 'performance'), ('performance', 'can'), ('can',
'bring'), ('bring', 'in'), ('in', 'sky'), ('sky', 'high'), ('high', 'success'),
```

```
('success', '.')]
```

6 Trigrams

[ ]:
```
#Code to generate trigram
import nltk
# nltk.download('punkt')

data = "The best performance can bring in sky high success."
tokens = nltk.word_tokenize(data)

print(list(nltk.trigrams(tokens)))
```

```
[('The', 'best', 'performance'), ('best', 'performance', 'can'), ('performance',
'can', 'bring'), ('can', 'bring', 'in'), ('bring', 'in', 'sky'), ('in', 'sky',
'high'), ('sky', 'high', 'success'), ('high', 'success', '.')]
```

Q] Why bigram and trigram are important?

- Analyzing sequences of words rather than individual words in isolation can reveal patterns, relationships, and syntactic structures within a text.
- For tasks like Named Entity Recognition, bigrams and trigrams can aid in identifying multi-word entities, like “New York City” or “United States of America.”
- N-grams are often used in language modeling, where the goal is to predict the probability of the next word or sequence of words in a given context.

7 Stopwords

[ ]:
```
import nltk
nltk.download('stopwords')  # Download the stopwords dataset

from nltk.corpus import stopwords

# Get the list of English stopwords
stop_words = set(stopwords.words('english'))

print(stop_words)
```

```
{'them', 'few', 'am', "haven't", 'wouldn', 'into', 'here', 'shan', 'out',
'where', 'these', 'on', "hasn't", 'are', 'to', 'once', 've', "aren't",
"that'll", 'you', 'until', "shouldn't", 's', 'other', 'nor', 'again', 'a',
"she's", "didn't", "couldn't", 'wasn', 'haven', 'then', 'very', 'those', 'of',
"you're", 'ours', 'mightn', 't', 'had', 'against', 'about', 'after', "won't",
'not', 'from', 'as', 'why', 'didn', "you'd", 'theirs', 'which', 'ma', 'were',
'ain', 'if', 'aren', 'because', 'been', "don't", 'her', 'the', 'same', 'hadn',
'down', 'with', 'they', 'while', 'himself', 'whom', 'themselves', 'doing',
```

```
'will', "mightn't", 'your', 'and', 'yourself', 'above', 'isn', 'too', 'm',
'this', 'yours', 'we', 'who', 'him', 'over', 'now', "you've", 'should', 'needn',
'below', 'in', 'did', 'it', 'd', 'has', 'than', 'won', 'shouldn', "it's",
'more', 'how', 'mustn', 'be', 'some', "you'll", 'she', 'its', 'for', 'before',
'up', 'doesn', "shan't", 'my', 'each', 'our', 'o', 'hasn', 'so', "wasn't",
'all', 'having', 'there', 'an', "should've", 'only', 'no', 'can', "needn't",
'i', 'is', 'couldn', 'was', 'yourselves', 'herself', 'but', 'myself', 'his',
'what', 'me', 'or', 'under', 'll', 'don', 'most', 'own', 're', 'any', 'weren',
'do', 'during', 'y', 'itself', "wouldn't", 'through', "mustn't", 'ourselves',
"doesn't", "isn't", "hadn't", 'at', 'further', 'their', "weren't", 'by',
'being', 'when', 'that', 'such', 'he', 'between', 'have', 'does', 'off', 'just',
'hers', 'both'}
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

8 POS Tags

[ ]:
```
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

from nltk.tokenize import word_tokenize
from nltk import pos_tag

sentence = "She is reading a book."
tokens = word_tokenize(sentence)
pos_tags = pos_tag(tokens)

print(pos_tags)
```

```
[('She', 'PRP'), ('is', 'VBZ'), ('reading', 'VBG'), ('a', 'DT'), ('book', 'NN'),
('.', '.')]
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

9 Dependency Parsing

[ ]:
```
import spacy

# Load the English language model
nlp = spacy.load("en_core_web_sm")
```

```
# Input sentence
sentence = "The quick brown fox jumps over the lazy dog."

# Process the sentence using spaCy
doc = nlp(sentence)

# Print dependency parse information
for token in doc:
    print(token.text, token.dep_, token.head.text, [child for child in token.
 ↪children])
```

```
The det fox []
quick amod fox []
brown amod fox []
fox nsubj jumps [The, quick, brown]
jumps ROOT jumps [fox, over, .]
over prep jumps [dog]
the det dog []
lazy amod dog []
dog pobj over [the, lazy]
. punct jumps []
```

Dependency Tree

[ ]:
```
import spacy
from spacy import displacy
from IPython.display import HTML

html = displacy.render(doc, style="dep", options={"compact": True}, page=True)
display(HTML(html))
```

```
<IPython.core.display.HTML object>
```

[ ]:
```

```