Day63_NLP_1_NLU_Advanced_Tokenization_Text_Preprocessing

August 13, 2025

Advanced Tokenization, N-grams, Stemming, Lemmatization & Stopwords

Today, we continued our journey into Natural Language Processing (NLP) by exploring more tokenization techniques, generating n-grams, and learning about stemming, lemmatization, and stopwords.

1 Whitespace Tokenization

Definition: Splits text based on whitespace (spaces, tabs, newlines) without removing punctuation.

- Useful when punctuation should be preserved as part of the tokens.
- Faster but less precise than other tokenizers.

[1]:
```
# you can also create your own words
AI = '''Artificial Intelligence refers to the intelligence of machines. This is␣
 ↪in contrast to the natural intelligence of
humans and animals. With Artificial Intelligence, machines perform functions␣
 ↪such as learning, planning, reasoning and
problem-solving. Most noteworthy, Artificial Intelligence is the simulation of␣
 ↪human intelligence by machines.
It is probably the fastest-growing development in the World of technology and␣
 ↪innovation. Furthermore, many experts believe
AI could solve major challenges and crisis situations.'''
```

[2]:
```
from nltk.tokenize import WhitespaceTokenizer

wt = WhitespaceTokenizer().tokenize(AI)
print(wt)  # Clean split by spaces
```

```
['Artificial', 'Intelligence', 'refers', 'to', 'the', 'intelligence', 'of',
'machines.', 'This', 'is', 'in', 'contrast', 'to', 'the', 'natural',
'intelligence', 'of', 'humans', 'and', 'animals.', 'With', 'Artificial',
'Intelligence,', 'machines', 'perform', 'functions', 'such', 'as', 'learning,',
'planning,', 'reasoning', 'and', 'problem-solving.', 'Most', 'noteworthy,',
'Artificial', 'Intelligence', 'is', 'the', 'simulation', 'of', 'human',
'intelligence', 'by', 'machines.', 'It', 'is', 'probably', 'the', 'fastest-
```

```
growing', 'development', 'in', 'the', 'World', 'of', 'technology', 'and',
'innovation.', 'Furthermore,', 'many', 'experts', 'believe', 'AI', 'could',
'solve', 'major', 'challenges', 'and', 'crisis', 'situations.']
```

[3]:
```
print(len(wt))  # Count of tokens
```

```
70
```

2 WordPunct Tokenization

Definition: Splits words and punctuation into separate tokens.

- Numbers, punctuation marks, and words are treated individually.

[4]:
```
from nltk.tokenize import wordpunct_tokenize

s = 'Good apple cost $3.88 in Hyderabad. Please buy two of them. Thanks.'
s
```

[4]:
```
'Good apple cost $3.88 in Hyderabad. Please buy two of them. Thanks.'
```

[5]:
```
print(wordpunct_tokenize(s))
```

```
['Good', 'apple', 'cost', '$', '3', '.', '88', 'in', 'Hyderabad', '.', 'Please',
'buy', 'two', 'of', 'them', '.', 'Thanks', '.']
```

[6]:
```
print(len(wordpunct_tokenize(s)))
```

```
18
```

[7]:
```
w_p = wordpunct_tokenize(AI)
print(w_p)
print(len(w_p))
```

```
['Artificial', 'Intelligence', 'refers', 'to', 'the', 'intelligence', 'of',
'machines', '.', 'This', 'is', 'in', 'contrast', 'to', 'the', 'natural',
'intelligence', 'of', 'humans', 'and', 'animals', '.', 'With', 'Artificial',
'Intelligence', ',', 'machines', 'perform', 'functions', 'such', 'as',
'learning', ',', 'planning', ',', 'reasoning', 'and', 'problem', '-', 'solving',
'.', 'Most', 'noteworthy', ',', 'Artificial', 'Intelligence', 'is', 'the',
'simulation', 'of', 'human', 'intelligence', 'by', 'machines', '.', 'It', 'is',
'probably', 'the', 'fastest', '-', 'growing', 'development', 'in', 'the',
'World', 'of', 'technology', 'and', 'innovation', '.', 'Furthermore', ',',
'many', 'experts', 'believe', 'AI', 'could', 'solve', 'major', 'challenges',
'and', 'crisis', 'situations', '.']
85
```

Summary of Tokenizers

Function / Tokenizer	Description
`word_tokenize`	Splits text into words and punctuation (accurate).
`sent_tokenize`	Splits text into sentences.
`blankline_tokenize`	Splits text into paragraphs by blank lines.
`WhitespaceTokenizer`	Splits by spaces/tabs only, keeps punctuation attached.
`wordpunct_tokenize`	Splits words and punctuation into separate tokens.
`pos_tag`	Assigns grammatical roles (nouns, verbs, adjectives) to each token.
`ne_chunk`	Performs Named Entity Recognition (NER) to identify people, places, etc.

3 N-grams

Definition: Sequences of n consecutive tokens.

- Bigram: 2-word sequence.
- Trigram: 3-word sequence.
- N-gram: Any n-word sequence (n > 3).

[8]:
```
import nltk
from nltk.util import bigrams, trigrams, ngrams

string = "we are learner of AI from 4th May 2025 till now"
string
```

[8]:
```
'we are learner of AI from 4th May 2025 till now'
```

[9]:
```
quotes_tokens = nltk.word_tokenize(string)

print(quotes_tokens)
print(len(quotes_tokens))
```

```
['we', 'are', 'learner', 'of', 'AI', 'from', '4th', 'May', '2025', 'till',
'now']
11
```

3.1 Bigrams

[10]:
```
# Bigrams
quotes_tokens_bi = list(nltk.bigrams(quotes_tokens))
print(quotes_tokens_bi)
```

```
[('we', 'are'), ('are', 'learner'), ('learner', 'of'), ('of', 'AI'), ('AI',
'from'), ('from', '4th'), ('4th', 'May'), ('May', '2025'), ('2025', 'till'),
('till', 'now')]
```

3.2 Trigrams

[11]:
```
# Trigrams
quotes_tokens_tri = list(nltk.trigrams(quotes_tokens))
print(quotes_tokens_tri)
```

```
[('we', 'are', 'learner'), ('are', 'learner', 'of'), ('learner', 'of', 'AI'),
('of', 'AI', 'from'), ('AI', 'from', '4th'), ('from', '4th', 'May'), ('4th',
'May', '2025'), ('May', '2025', 'till'), ('2025', 'till', 'now')]
```

3.3 N-grams (n=8)

[12]:
```
# N-grams (n=8)
quotes_tokens_n = list(nltk.ngrams(quotes_tokens, 8))
print(quotes_tokens_n)
```

```
[('we', 'are', 'learner', 'of', 'AI', 'from', '4th', 'May'), ('are', 'learner',
'of', 'AI', 'from', '4th', 'May', '2025'), ('learner', 'of', 'AI', 'from',
'4th', 'May', '2025', 'till'), ('of', 'AI', 'from', '4th', 'May', '2025',
'till', 'now')]
```

4 Stemming

Definition: Reduces words to their root form, often by chopping off suffixes. Types:

- **Porter Stemmer** – Basic and widely used, but may not handle all words well.
- **Lancaster Stemmer** – More aggressive, sometimes over-stems words.
- **Snowball Stemmer** – Advanced, supports multiple languages.

[13]:
```
from nltk.stem import PorterStemmer, LancasterStemmer, SnowballStemmer

words_to_stem =␣
 ↪['given','give','giving','gave','thinking','loving','maximum','akshaybhujbal','gaved']
words_to_stem
```

[13]:
```
['given',
 'give',
 'giving',
 'gave',
 'thinking',
 'loving',
 'maximum',
 'akshaybhujbal',
 'gaved']
```

4.1 Porter Stemmer

[14]:
```
# Porter Stemmer
pst = PorterStemmer()
for word in words_to_stem:
    print(word + ' : ' + pst.stem(word))
```

```
given : given
give : give
giving : give
gave : gave
thinking : think
loving : love
maximum : maximum
akshaybhujbal : akshaybhujb
gaved : gave
```

4.2 Lancaster Stemmer

[15]:
```
# Lancaster Stemmer
lst = LancasterStemmer()
for word in words_to_stem:
    print(word + ' : ' + lst.stem(word))
```

```
given : giv
give : giv
giving : giv
gave : gav
thinking : think
loving : lov
maximum : maxim
akshaybhujbal : akshaybhujb
gaved : gav
```

4.3 Snowball Stemmer (English)

[16]:
```
# Snowball Stemmer (English)
sbst = SnowballStemmer('english')
for word in words_to_stem:
    print(word + ' : ' + sbst.stem(word))
```

```
given : given
give : give
giving : give
gave : gave
thinking : think
loving : love
maximum : maximum
```

```
akshaybhujbal : akshaybhujb
gaved : gave
```

[17]:
```
# Snowball Stemmer (German)
stemmer = SnowballStemmer('german')
print(stemmer.stem('samstag'))
```

```
samstag
```

5 Lemmatization

Definition: Reduces words to their dictionary (lemma) form, considering meaning and grammar.

- More accurate than stemming because it uses vocabulary and morphological analysis.

[18]:
```
from nltk.stem import WordNetLemmatizer

word_lem = WordNetLemmatizer()
for word in words_to_stem:
    print(word + ' : ' + word_lem.lemmatize(word))
```

```
given : given
give : give
giving : giving
gave : gave
thinking : thinking
loving : loving
maximum : maximum
akshaybhujbal : akshaybhujbal
gaved : gaved
```

6 Stopwords

Definition: Commonly used words (e.g., “the”, “is”, “in”) that are often removed in NLP tasks. - Removing stopwords helps focus on meaningful content.

[23]:
```
from nltk.corpus import stopwords

print(stopwords.words('english')[:10])
```

```
['a', 'about', 'above', 'after', 'again', 'against', 'ain', 'all', 'am', 'an']
```

[24]:
```
int(len(stopwords.words('english')))
```

[24]:
```
198
```

[25]:
```
from nltk.corpus import stopwords

print("Stopwords length for French is", len(stopwords.words('french')))
```

```
print("Stopwords length for German is", len(stopwords.words('german')))
print("Stopwords length for Chinese is", len(stopwords.words('chinese')))
```

```
Stopwords length for French is 157
Stopwords length for German is 232
Stopwords length for Chinese is 841
```

7 POS Tagging

Definition: Assigns grammatical labels (noun, verb, adjective, etc.) to each token in a sentence.

[26]:
```
import nltk
from nltk.tokenize import word_tokenize

sentence = "The quick brown fox jumps over the lazy dog"
tokens = word_tokenize(sentence)
pos_tags = nltk.pos_tag(tokens)

print(pos_tags)
```

```
[('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), ('jumps',
'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('dog', 'NN')]
```

8 Named Entity Recognition (NER)

Definition: Identifies and classifies entities in text (people, places, organizations, dates, etc.).

[27]:
```
from nltk import ne_chunk

NE_sent = "The US president stays in the WHITEHOUSE"
NE_tokens = word_tokenize(NE_sent)
NE_tags = nltk.pos_tag(NE_tokens)
NE_NER = ne_chunk(NE_tags)

print(NE_NER)
```

```
(S
  The/DT
  (GSP US/NNP)
  president/NN
  stays/NNS
  in/IN
  the/DT
  (ORGANIZATION WHITEHOUSE/NNP))
```

9 Syntax in NLP

Definition: The set of rules, principles, and processes that govern the structure of sentences in a language.

Example Rule: In English, a simple sentence follows Subject → Verb → Object.

Example: "She (S) eats (V) apples (O)."

In NLP: Syntax parsing helps a machine understand how words relate to each other grammatically, which is important for translation, question answering, and information extraction

Summary

- Learned WhitespaceTokenizer and wordpunct_tokenize for flexible token splitting.
- Explored n-grams for sequence-based analysis.
- Compared Porter, Lancaster, and Snowball stemmers.
- Understood lemmatization for accurate root forms.
- Reviewed stopwords in multiple languages.
- Applied POS tagging to identify grammatical roles of words.
- Performed Named Entity Recognition (NER) to detect people, places, and organizations.
- Discussed syntax as the set of rules and principles that define sentence structure in NLP.