Day63_NLP_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.

[2]:
```
# 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.'''
```

[3]:
```
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.']
```

[4]:
```
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.

[5]:
```
from nltk.tokenize import wordpunct_tokenize

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

[5]:
```
'Good apple cost $3.88 in Hyderabad. Please buy two of them. Thanks.'
```

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

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

[7]:
```
print(len(wordpunct_tokenize(s)))
```

```
18
```

[8]:
```
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

Tokenizer	Description
`word_tokenize`	Splits into words and punctuation (accurate).

Tokenizer	Description
`sent_tokenize`	Splits into sentences.
`blankline_tokenize`	Splits into paragraphs by blank lines.
`WhitespaceTokenizer`	Splits by spaces/tabs only.
`wordpunct_tokenize`	Splits words and punctuation separately.

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).

[13]:
```
import nltk
from nltk.util import bigrams, trigrams, ngrams

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

[13]:
```
'we are learner of AI from 4th May 2025 till now'
```

[14]:
```
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

[15]:
```
# 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

[17]:
```
# 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)

[16]:
```
# 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.

[19]:
```
from nltk.stem import PorterStemmer, LancasterStemmer, SnowballStemmer

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

[19]:
```
['given',
 'give',
 'giving',
 'gave',
 'thinking',
 'loving',
 'maximum',
 'akshaybhujbal',
 'gaved']
```

4.1 Porter Stemmer

[20]:
```
# 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

[21]:
```
# 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)

[22]:
```
# 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
```

[26]:
```
# 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.

[27]:
```
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.

[29]:
```
from nltk.corpus import stopwords

stopwords.words('english')
```

[29]:
```
['a',
 'about',
 'above',
 'after',
 'again',
 'against',
 'ain',
 'all',
 'am',
 'an',
 'and',
 'any',
 'are',
 'aren',
 "aren't",
 'as',
 'at',
```

```
'be',
'because',
'been',
'before',
'being',
'below',
'between',
'both',
'but',
'by',
'can',
'couldn',
"couldn't",
'd',
'did',
'didn',
"didn't",
'do',
'does',
'doesn',
"doesn't",
'doing',
'don',
"don't",
'down',
'during',
'each',
'few',
'for',
'from',
'further',
'had',
'hadn',
"hadn't",
'has',
'hasn',
"hasn't",
'have',
'haven',
"haven't",
'having',
'he',
"he'd",
"he'll",
'her',
'here',
'hers',
```

```
'herself',
"he's",
'him',
'himself',
'his',
'how',
'i',
"i'd",
'if',
"i'll",
"i'm",
'in',
'into',
'is',
'isn',
"isn't",
'it',
"it'd",
"it'll",
"it's",
'its',
'itself',
"i've",
'just',
'll',
'm',
'ma',
'me',
'mightn',
"mightn't",
'more',
'most',
'mustn',
"mustn't",
'my',
'myself',
'needn',
"needn't",
'no',
'nor',
'not',
'now',
'o',
'of',
'off',
'on',
'once',
```

```
'only',
'or',
'other',
'our',
'ours',
'ourselves',
'out',
'over',
'own',
're',
's',
'same',
'shan',
"shan't",
'she',
"she'd",
"she'll",
"she's",
'should',
'shouldn',
"shouldn't",
"should've",
'so',
'some',
'such',
't',
'than',
'that',
"that'll",
'the',
'their',
'theirs',
'them',
'themselves',
'then',
'there',
'these',
'they',
"they'd",
"they'll",
"they're",
"they've",
'this',
'those',
'through',
'to',
'too',
```

```
 'under',
 'until',
 'up',
 've',
 'very',
 'was',
 'wasn',
 "wasn't",
 'we',
 "we'd",
 "we'll",
 "we're",
 'were',
 'weren',
 "weren't",
 "we've",
 'what',
 'when',
 'where',
 'which',
 'while',
 'who',
 'whom',
 'why',
 'will',
 'with',
 'won',
 "won't",
 'wouldn',
 "wouldn't",
 'y',
 'you',
 "you'd",
 "you'll",
 'your',
 "you're",
 'yours',
 'yourself',
 'yourselves',
 "you've"]
```

[30]:
```
int(len(stopwords.words('english')))
```

[30]: 198

[34]:
```
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
```

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.

7 NLP Quick Reference – Tokenization, N-grams, Stemming, Lemmatization, Stopwords

7.1 Example Text

[36]:
```
text = """Natural Language Processing makes it possible for machines
to understand human language and perform useful tasks."""
```

7.2 Basic Tokenization

[40]:
```
from nltk.tokenize import word_tokenize, sent_tokenize, blankline_tokenize

word_tokenize(text)          # Words + punctuation
```

[40]:
```
['Natural',
 'Language',
 'Processing',
 'makes',
 'it',
 'possible',
 'for',
 'machines',
 'to',
 'understand',
 'human',
 'language',
 'and',
 'perform',
 'useful',
 'tasks',
 '.']
```

[41]:
```
sent_tokenize(text)          # Sentences
```

[41]:
```
['Natural Language Processing makes it possible for machines \nto understand
human language and perform useful tasks.']
```

[42]:
```
blankline_tokenize(text)     # Paragraphs
```

[42]:
```
['Natural Language Processing makes it possible for machines \nto understand
human language and perform useful tasks.']
```

7.3 Whitespace Tokenizer

Splits only on spaces/tabs, keeps punctuation attached.

[43]:
```
from nltk.tokenize import WhitespaceTokenizer
WhitespaceTokenizer().tokenize(text)
```

[43]:
```
['Natural',
 'Language',
 'Processing',
 'makes',
 'it',
 'possible',
 'for',
 'machines',
 'to',
 'understand',
 'human',
 'language',
 'and',
 'perform',
 'useful',
 'tasks.']
```

7.4 WordPunct Tokenizer

Splits words and punctuation separately.

[44]:
```
from nltk.tokenize import wordpunct_tokenize
wordpunct_tokenize(text)
```

[44]:
```
['Natural',
 'Language',
 'Processing',
 'makes',
 'it',
 'possible',
 'for',
```

```
 'machines',
 'to',
 'understand',
 'human',
 'language',
 'and',
 'perform',
 'useful',
 'tasks',
 '.']
```

7.5 N-grams

Create sequences of n consecutive tokens.

[46]:
```
import nltk
tokens = nltk.word_tokenize(text)

list(nltk.bigrams(tokens))        # Bigrams
```

[46]:
```
[('Natural', 'Language'),
 ('Language', 'Processing'),
 ('Processing', 'makes'),
 ('makes', 'it'),
 ('it', 'possible'),
 ('possible', 'for'),
 ('for', 'machines'),
 ('machines', 'to'),
 ('to', 'understand'),
 ('understand', 'human'),
 ('human', 'language'),
 ('language', 'and'),
 ('and', 'perform'),
 ('perform', 'useful'),
 ('useful', 'tasks'),
 ('tasks', '.')]
```

[47]:
```
list(nltk.trigrams(tokens))       # Trigrams
```

[47]:
```
[('Natural', 'Language', 'Processing'),
 ('Language', 'Processing', 'makes'),
 ('Processing', 'makes', 'it'),
 ('makes', 'it', 'possible'),
 ('it', 'possible', 'for'),
 ('possible', 'for', 'machines'),
 ('for', 'machines', 'to'),
 ('machines', 'to', 'understand'),
 ('to', 'understand', 'human'),
```

```
 ('understand', 'human', 'language'),
 ('human', 'language', 'and'),
 ('language', 'and', 'perform'),
 ('and', 'perform', 'useful'),
 ('perform', 'useful', 'tasks'),
 ('useful', 'tasks', '.')]
```

[48]:
```
list(nltk.ngrams(tokens, 4))      # 4-word n-grams
```

[48]:
```
[('Natural', 'Language', 'Processing', 'makes'),
 ('Language', 'Processing', 'makes', 'it'),
 ('Processing', 'makes', 'it', 'possible'),
 ('makes', 'it', 'possible', 'for'),
 ('it', 'possible', 'for', 'machines'),
 ('possible', 'for', 'machines', 'to'),
 ('for', 'machines', 'to', 'understand'),
 ('machines', 'to', 'understand', 'human'),
 ('to', 'understand', 'human', 'language'),
 ('understand', 'human', 'language', 'and'),
 ('human', 'language', 'and', 'perform'),
 ('language', 'and', 'perform', 'useful'),
 ('and', 'perform', 'useful', 'tasks'),
 ('perform', 'useful', 'tasks', '.')]
```

7.6 Stemming

Reduce words to root form (may not be real words).

[49]:
```
from nltk.stem import PorterStemmer, LancasterStemmer, SnowballStemmer
words = ['running', 'runs', 'easily', 'fairly']

PorterStemmer().stem('running')
```

[49]:
```
'run'
```

[50]:
```
LancasterStemmer().stem('running')
```

[50]:
```
'run'
```

[51]:
```
SnowballStemmer('english').stem('running')
```

[51]:
```
'run'
```

7.7 Lemmatization

Reduce to dictionary form, considering meaning.

[52]:
```
from nltk.stem import WordNetLemmatizer
WordNetLemmatizer().lemmatize('running', pos='v')  # 'run'
```

[52]:
```
'run'
```

7.8 Stopwords

Common words often removed in text processing.

[53]:
```
from nltk.corpus import stopwords
stopwords.words('english')[:10]   # First 10 stopwords
len(stopwords.words('english'))   # Count of English stopwords
```

[53]:
```
198
```

Closing Notes :

- Tokenization is the **first step** in almost every NLP task.
- The choice between **stemming** and **lemmatization** depends on whether you need speed (stemming) or accuracy (lemmatization).
- **Stopwords removal** helps focus on meaningful words, but always check if they are important for your specific task.
- **N-grams** capture word sequences and context, useful for tasks like text prediction or sentiment analysis.
- Always experiment with different tokenizers and preprocessing techniques to see what works best for your dataset.