Natural Language Processing using Python Programming

Notebook 04.1: Introduction to Part-of-Speech (POS) Tagging

Part of the comprehensive learning series: Natural Language Processing using Python Programming

Learning Objectives:

- Master Part-of-Speech (POS) tagging using NLTK's statistical models
- Understand the Penn Treebank tagset and common grammatical categories
- Resolve word ambiguity through contextual grammatical analysis
- Implement content word filtering for improved text processing
- Compare POS-based filtering with traditional stopword removal techniques

---

- **Part-of-Speech (POS) Tagging** is the process of labeling each word in a sentence with its corresponding grammatical category (e.g., noun, verb, adjective).
- This is a crucial step in NLP as it helps the computer understand the **role** a word plays in the sentence structure, enabling more complex analysis like dependency parsing and named entity recognition.

1. Setting up: Libraries and Sample Text

- We will use NLTK for its clear, sequential POS tagging and introduce the standard tagset.

In [1]:
```
# Import necessary libraries
import nltk
from nltk.tokenize import word_tokenize

# Importing the function to get tag descriptions
from nltk.help import upenn_tagset

# Ensure the necessary resource for POS tagging is downloaded
# averaged_perceptron_tagger: A pre-trained model for POS tagging
# tagsets: Provides descriptions for the POS tags
# tagsets_json: JSON format descriptions for the POS tags
nltk.download('averaged_perceptron_tagger', quiet=True)
nltk.download('tagsets_json', quiet=True)
```

Out[1]:
```
True
```

In [2]:
```
sample_text = "The developers are constantly building new solutions, which is real
print(f"Sample Text: {sample_text}")
```

```
Sample Text: The developers are constantly building new solutions, which is really
great.
```

---

2. POS Tagging with NLTK

- NLTK provides the `pos_tag` function, which first tokenizes the text (or accepts pre-tokenized words) and then applies a trained model (the averaged perceptron tagger) to assign the appropriate tag.

In [3]:
```
# 1. Tokenize the text
tokens = word_tokenize(sample_text)

# 2. Apply NLTK's POS tagger
tagged_tokens = nltk.pos_tag(tokens)

print("NLTK POS Tagging Result:")
for token, tag in tagged_tokens:
    # Display token, its tag, and the description of the tag
    # print(f"token: {token:<10} | tagged: {tag} | description: {upenn_tagset(tag)
    print(f"token: {token:<10} | tagged: {tag}")
```

```
NLTK POS Tagging Result:
token: The        | tagged: DT
token: developers | tagged: NNS
token: are        | tagged: VBP
token: constantly | tagged: RB
token: building   | tagged: VBG
token: new        | tagged: JJ
token: solutions  | tagged: NNS
token: ,          | tagged: ,
token: which      | tagged: WDT
token: is         | tagged: VBZ
token: really     | tagged: RB
token: great      | tagged: JJ
token: .          | tagged: .
```

Why is POS Tagging Important?

- Consider the word **'address'**. Without context, it could be:
    1. A **Noun** (location): "What is your shipping **address**?" (NN)
    2. A **Verb** (to speak to): "The CEO will **address** the board." (VB)
- POS tagging resolves this ambiguity, which is crucial for subsequent analysis, such as only extracting Nouns for feature generation.

In [4]:
```
# Example to show context-based POS tagging
# The word 'address' can be a Noun or a Verb based on context
sentence1 = word_tokenize("What is your shipping address?")
sentence2 = word_tokenize("The CEO will address the board.")
```

```
print("Sentence 1 Tags:", nltk.pos_tag(sentence1))
print("Sentence 2 Tags:", nltk.pos_tag(sentence2))

# Output Observation: 'address' is correctly identified as a Noun (NN) in S1 and d
```

```
Sentence 1 Tags: [('What', 'WP'), ('is', 'VBZ'), ('your', 'PRP$'), ('shipping', 'VB
G'), ('address', 'NN'), ('?', '.')]
Sentence 2 Tags: [('The', 'DT'), ('CEO', 'NNP'), ('will', 'MD'), ('address', 'VB'),
('the', 'DT'), ('board', 'NN'), ('.', '.')]
```

---

3. Understanding the Penn Treebank Tagset

- The tags outputted by NLTK (like `DT`, `NN`, `VBG`) are part of the **Penn Treebank Tagset**, the most commonly used standard in English NLP.
- It contains 36 tags for parts-of-speech and 12 tags for punctuation and currency.
- Knowing the common tags is essential for feature engineering.

Key POS Tag Examples

In [5]:
```
print("Common Tag Examples:")
print("----------------------")
nltk.help.upenn_tagset('NN')   # Noun, singular or mass
nltk.help.upenn_tagset('NNS')  # Noun, plural
nltk.help.upenn_tagset('NNP')  # Proper noun, singular (e.g., 'Apple', 'London')
nltk.help.upenn_tagset('VB')   # Verb, base form (e.g., 'see', 'write')
nltk.help.upenn_tagset('VBG')  # Verb, gerund or present participle (e.g., 'runnir
nltk.help.upenn_tagset('VBD')  # Verb, past tense (e.g., 'saw', 'wrote')
nltk.help.upenn_tagset('RB')   # Adverb (e.g., 'quickly', 'very', 'well')
nltk.help.upenn_tagset('JJ')   # Adjective (e.g., 'big', 'new', 'amazing')
nltk.help.upenn_tagset('DT')   # Determiner (e.g., 'the', 'a', 'an')
```

```
Common Tag Examples:
----------------------
NN: noun, common, singular or mass
    common-carrier cabbage knuckle-duster Casino afghan shed thermostat
    investment slide humour falloff slick wind hyena override subhumanity
    machinist ...
NNS: noun, common, plural
    undergraduates scotches bric-a-brac products bodyguards facets coasts
    divestitures storehouses designs clubs fragrances averages
    subjectivists apprehensions muses factory-jobs ...
NNP: noun, proper, singular
    Motown Venneboerger Czestochwa Ranzer Conchita Trumplane Christos
    Oceanside Escobar Kreisler Sawyer Cougar Yvette Ervin ODI Darryl CTCA
    Shannon A.K.C. Meltex Liverpool ...
VB: verb, base form
    ask assemble assess assign assume atone attention avoid bake balkanize
    bank begin behold believe bend benefit bevel beware bless boil bomb
    boost brace break bring broil brush build ...
VBG: verb, present participle or gerund
    telegraphing stirring focusing angering judging stalling lactating
    hankerin' alleging veering capping approaching traveling besieging
    encrypting interrupting erasing wincing ...
VBD: verb, past tense
    dipped pleaded swiped regummed soaked tidied convened halted registered
    cushioned exacted snubbed strode aimed adopted belied figgered
    speculated wore appreciated contemplated ...
RB: adverb
    occasionally unabatingly maddeningly adventurously professedly
    stirringly prominently technologically magisterially predominately
    swiftly fiscally pitilessly ...
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 ...
DT: determiner
    all an another any both del each either every half la many much nary
    neither no some such that the them these this those
```

Practical Use Case: Filtering for Content Words

- For tasks like Text Classification or Information Retrieval, we often only care about **content words** (Nouns, Verbs, Adjectives, Adverbs), and ignore **function words** (Determiners, Prepositions, Conjunctions).
- POS tagging makes this filtering easy.

In [6]:
```python
# Practical Use Case: Filtering for Content Words
# Content words: Nouns, Verbs, Adjectives, Adverbs
# Function words: Determiners, Prepositions, Conjunctions
text_to_filter = "The extremely fast runner easily won the long marathon."
tagged_text = nltk.pos_tag(word_tokenize(text_to_filter))

# Define tags we want to keep (Content Words)
content_tags = ['NN', 'NNS', 'NNP', 'VB', 'VBD', 'VBG', 'JJ', 'RB'] # Nouns, Verbs

content_words = [word for word, tag in tagged_text if tag in content_tags]

print(f"Original Text: {text_to_filter}")
```

```
print(f"\nTagged Tokens: {tagged_text}")
print(f"\nContent Words Only: {content_words}")
```

```
Original Text: The extremely fast runner easily won the long marathon.

Tagged Tokens: [('The', 'DT'), ('extremely', 'RB'), ('fast', 'JJ'), ('runner', 'N
N'), ('easily', 'RB'), ('won', 'VBD'), ('the', 'DT'), ('long', 'JJ'), ('marathon',
'NN'), ('.', '.')]

Content Words Only: ['extremely', 'fast', 'runner', 'easily', 'won', 'long', 'marat
hon']
```

---

4. Comparing POS Tagging to Simple N-grams (Content Filtering)

- We can contrast this sophisticated linguistic filtering with the crude N-gram creation we will see later.
- The difference highlights the value of POS tagging.

In [7]:
```
# Importing for comparison with simple stopword filtering
# ngrams: For creating n-grams (not used here but for context)
# n-grams: contiguous sequences of n items from a given sample of text
from nltk.util import ngrams
from nltk.corpus import stopwords

# 1. Simple Stopword Filtering (Chapter 2 method)
stop_words = set(stopwords.words('english'))
tokens = [w.lower() for w in word_tokenize(text_to_filter) if w.isalpha()]
stopword_filtered = [w for w in tokens if w not in stop_words]

# 2. POS Tagging Filtered Result (from above)
pos_filtered = content_words

print(f"Stopword Filtered: {stopword_filtered}")
print(f"POS Tag Filtered:  {pos_filtered}")

# Observation: POS filtering includes 'extremely' (Adverb, RB) and 'long' (Adject
# which are meaningful content words, and also preserves 'won' (past tense verb, \
# POS tagging provides a more linguistically informed, targeted list of words cru
```

```
Stopword Filtered: ['extremely', 'fast', 'runner', 'easily', 'long', 'marathon']
POS Tag Filtered:  ['extremely', 'fast', 'runner', 'easily', 'won', 'long', 'marath
on']
```

5. Summary and Next Steps

- POS tagging is a foundational step for linguistic analysis, providing the grammatical context needed to disambiguate words and filter text effectively.
- We've mastered NLTK's `pos_tag` and understood the Penn Treebank tags.
- In the next notebook (**4.2**), we will explore the more advanced structural analysis provided by **SpaCy**: **Dependency Parsing**, which uses these POS tags to build a syntactic graph of the entire sentence.

Key Takeaways

- **POS Tagging Mastery:** We successfully implemented Part-of-Speech tagging using NLTK, learning to assign grammatical categories to words in context.
- **Penn Treebank Understanding:** We mastered the standard Penn Treebank tagset, understanding the meaning and application of common POS tags like NN, VB, JJ, and RB.
- **Word Disambiguation:** We demonstrated how POS tagging resolves word ambiguity (like 'address' as noun vs. verb) through contextual analysis.
- **Content Filtering:** We implemented sophisticated content word filtering using POS tags, showing advantages over traditional stopword removal methods.

Next Notebook Preview

- With POS tagging mastered, we're ready to explore **advanced syntactic analysis**.
- The next notebook will dive into **Dependency Parsing with SpaCy**, building syntactic graphs that show relationships between words in sentences.

About This Project

This notebook is part of the **Natural Language Processing using Python Programming for Beginners** repository - a comprehensive, beginner-friendly guide for mastering NLP using Python, NLTK, and SpaCy.

Repository: `NLP`

Author

Prakash Ukhalkar

GitHub prakash-ukhalkar
