# ITAI 2373 Module 05: Part-of-Speech Tagging

## In-Class Exercise & Homework Lab

Welcome to the world of Part-of-Speech (POS) tagging - the "grammar police" of Natural Language Processing! 🚔 📝

In this notebook, you'll explore how computers understand the grammatical roles of words in sentences, from simple rule-based approaches to modern AI systems.

### What You'll Learn:

- **Understand POS tagging fundamentals** and why it matters in daily apps
- **Use NLTK and SpaCy** for practical text analysis
- **Navigate different tag sets** and understand their trade-offs
- **Handle real-world messy text** like speech transcripts and social media
- **Apply POS tagging** to solve actual business problems

### Structure:

- **Part 1**: In-Class Exercise (30-45 minutes) - Basic concepts and hands-on practice
- **Part 2**: Homework Lab - Real-world applications and advanced challenges

---

💡 ***Pro Tip**: POS tagging is everywhere! It helps search engines understand "Apple stock" vs "apple pie", helps Siri understand your commands, and powers autocorrect on your phone.*

## 🛠️ Setup and Installation

Let's get our tools ready! We'll use two powerful libraries:

- **NLTK**: The "Swiss Army knife" of NLP - comprehensive but requires setup
- **SpaCy**: The "speed demon" - built for production, cleaner output

Run the cells below to install and set up everything we need.

```
# Install required libraries (run this first!)
!pip install nltk spacy matplotlib seaborn pandas
!python -m spacy download en_core_web_sm

print("✅ Installation complete!")
```

```
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.0.12)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.0.13)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.11)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.0.10)
Requirement already satisfied: thinc<8.4.0,>=8.3.4 in /usr/local/lib/python3.11/dist-packages (from spacy) (8.3.6)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.1.3)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.5.1)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.10)
Requirement already satisfied: weasel<0.5.0,>=0.1.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (0.4.1)
Requirement already satisfied: typer<1.0.0,>=0.3.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (0.16.0)
Requirement already satisfied: numpy>=1.19.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.2)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.32.3)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.11.
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.1.6)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from spacy) (75.2.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (24.2)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2)
```

```
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.0.0
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,!=1.8.1,<3.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.0
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy) (3.10
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0->spacy)
Requirement already satisfied: blis<1.4.0,>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from thinc<8.4.0,>=8.3.4->spacy) (1.
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.11/dist-packages (from thinc<8.4.0,>=8.3.4->spac
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0->spacy) (1.
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0->spacy) (13.9.4)
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from weasel<0.5.0,>=0.1.0->s
Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in /usr/local/lib/python3.11/dist-packages (from weasel<0.5.0,>=0.1.0->spa
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->spacy) (3.0.2)
Requirement already satisfied: marisa-trie>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from language-data>=1.2->langcodes<4
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0.0,>
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0.0
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open<8.0.0,>=5.2.1->weasel<0.5.0,>=0.1
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->t
Collecting en-core-web-sm==3.8.0
  Using cached https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.8.0/en_core_web_sm-3.8.0-py3-none-any.
✔ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in
order to load all the package's dependencies. You can do this by selecting the
'Restart kernel' or 'Restart runtime' option.
```

```python
# Import all the libraries we'll need
import nltk
import spacy
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import warnings
warnings.filterwarnings('ignore')

# Download NLTK data (this might take a moment)
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('universal_tagset')

# Load SpaCy model
nlp = spacy.load('en_core_web_sm')

print("🎉 All libraries loaded successfully!")
print("📚 NLTK version:", nltk.__version__)
print("🚀 SpaCy version:", spacy.__version__)
```

```
[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!
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Package universal_tagset is already up-to-date!
🎉 All libraries loaded successfully!
📚 NLTK version: 3.9.1
🚀 SpaCy version: 3.8.7
```

---

# 🎯 PART 1: IN-CLASS EXERCISE (30-45 minutes)

Welcome to the hands-on portion! We'll start with the basics and build up your understanding step by step.

## Learning Goals for Part 1:

1. Understand what POS tagging does
2. Use NLTK and SpaCy for basic tagging
3. Interpret and compare different tag outputs

4. Explore word ambiguity with real examples
5. Compare different tagging approaches

## 🔍 Activity 1: Your First POS Tags (10 minutes)

Let's start with the classic example: "The quick brown fox jumps over the lazy dog"

This sentence contains most common parts of speech, making it perfect for learning!

```
# Let's start with a classic example
sentence = "The quick brown fox jumps over the lazy dog"

# TODO: Use NLTK to tokenize and tag the sentence
# Hint: Use nltk.word_tokenize() and nltk.pos_tag()
import nltk
nltk.download('averaged_perceptron_tagger_eng') # Download the missing resource
tokens = nltk.word_tokenize(sentence)
pos_tags = nltk.pos_tag(tokens)


print("Original sentence:", sentence)
print("\nTokens:", tokens)
print("\nPOS Tags:")
for word, tag in pos_tags:
    print(f"  {word:8} -> {tag}")
```

```
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
Original sentence: The quick brown fox jumps over the lazy dog

Tokens: ['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']

POS Tags:
  The      -> DT
  quick    -> JJ
  brown    -> NN
  fox      -> NN
  jumps    -> VBZ
  over     -> IN
  the      -> DT
  lazy     -> JJ
  dog      -> NN
```

### 🤔 Quick Questions:

1. What does 'DT' mean? What about 'JJ'? DT signifies Determiner Words such as "the," "a," and "an" that initiate noun phrases are labeled as DT
2. Why do you think 'brown' and 'lazy' have the same tag? JJ denotes Adjective Descriptive terms that modify nouns (for instance: "quick," "brown," "lazy") are classified as JJ. Since "brown" and "lazy" both describe the nouns "fox" and "dog," respectively, they are each assigned the adjective tag JJ
3. Can you guess what 'VBZ' represents? VBZ indicates a Verb, 3rd-person singular present This is why "jumps" is categorized as VBZ: it is a present-tense verb with the subject "fox" being third-person singular.

*Hint: Think about the grammatical role each word plays in the sentence!*

## 🚀 Activity 2: SpaCy vs NLTK Showdown (10 minutes)

Now let's see how SpaCy handles the same sentence. SpaCy uses cleaner, more intuitive tag names.

```
# TODO: Process the same sentence with SpaCy
# Hint: Use nlp(sentence) and access .text and .pos_ attributes
doc = nlp(sentence)

print("SpaCy POS Tags:")
for token in doc:
    print(f"  {token.text:8} -> {token.pos_:6} ({token.tag_})")
```

```
print("\n" + "="*50)
print("COMPARISON:")
print("="*50)

# Let's compare side by side
nltk_tags = nltk.pos_tag(nltk.word_tokenize(sentence))
spacy_doc = nlp(sentence)

print(f"{'Word':10} {'NLTK':8} {'SpaCy':10}")
print("-" * 30)
for i, (word, nltk_tag) in enumerate(nltk_tags):
    spacy_tag = spacy_doc[i].pos_
    print(f"{word:10} {nltk_tag:8} {spacy_tag:10}")
```

```
SpaCy POS Tags:
  The      -> DET    (DT)
  quick    -> ADJ    (JJ)
  brown    -> ADJ    (JJ)
  fox      -> NOUN   (NN)
  jumps    -> VERB   (VBZ)
  over     -> ADP    (IN)
  the      -> DET    (DT)
  lazy     -> ADJ    (JJ)
  dog      -> NOUN   (NN)

==================================================
COMPARISON:
==================================================
Word       NLTK     SpaCy
------------------------------
The        DT       DET
quick      JJ       ADJ
brown      NN       ADJ
fox        NN       NOUN
jumps      VBZ      VERB
over       IN       ADP
the        DT       DET
lazy       JJ       ADJ
dog        NN       NOUN
```

## 🎯 Discussion Points:

- Which tags are easier to understand: NLTK's or SpaCy's?
- Do you notice any differences in how they tag the same words?
- Which system would you prefer for a beginner? Why?

## 🎭 Activity 3: The Ambiguity Challenge (15 minutes)

Here's where things get interesting! Many words can be different parts of speech depending on context. Let's explore this with some tricky examples.

```
# Ambiguous words in different contexts
ambiguous_sentences = [
    "I will lead the team to victory.",          # lead = verb
    "The lead pipe is heavy.",                   # lead = noun (metal)
    "She took the lead in the race.",            # lead = noun (position)
    "The bank approved my loan.",                # bank = noun (financial)
    "We sat by the river bank.",                 # bank = noun (shore)
    "I bank with Chase.",                        # bank = verb
]

print("🎭 AMBIGUITY EXPLORATION")
print("=" * 40)

for sentence in ambiguous_sentences:
    print(f"\nSentence: {sentence}")

    # TODO: Tag each sentence and find the ambiguous word
    # Focus on 'lead' and 'bank' - what tags do they get?
    tokens = nltk.word_tokenize(sentence)
    tags = nltk.pos_tag(tokens)
```

```
    # Find and highlight the key word
    for word, tag in tags:
        if word.lower() in ['lead', 'bank']:
            print(f"   🎯 '{word}' is tagged as: {tag}")
```

```
🎭 AMBIGUITY EXPLORATION
========================================

Sentence: I will lead the team to victory.
  🎯 'lead' is tagged as: VB

Sentence: The lead pipe is heavy.
  🎯 'lead' is tagged as: NN

Sentence: She took the lead in the race.
  🎯 'lead' is tagged as: NN

Sentence: The bank approved my loan.
  🎯 'bank' is tagged as: NN

Sentence: We sat by the river bank.
  🎯 'bank' is tagged as: NN

Sentence: I bank with Chase.
  🎯 'bank' is tagged as: NN
```

### 🧠 Think About It:

1. How does the computer know the difference between "lead" (metal) and "lead" (guide)?
2. What clues in the sentence help determine the correct part of speech?
3. Can you think of other words that change meaning based on context?

**Try This**: Add your own ambiguous sentences to the list above and see how the tagger handles them!

## 📊 Activity 4: Tag Set Showdown (10 minutes)

NLTK can use different tag sets. Let's compare the detailed Penn Treebank tags (~~45 tags) with the simpler Universal Dependencies tags (~~17 tags).

```python
# Compare different tag sets
test_sentence = "The brilliant students quickly solved the challenging programming assignment."

# TODO: Get tags using both Penn Treebank and Universal tagsets
# Hint: Use tagset='universal' parameter for universal tags

import nltk
nltk.download('averaged_perceptron_tagger')
nltk.download('universal_tagset')
nltk.download('punkt_tab') # Download the missing resource
nltk.download('averaged_perceptron_tagger_eng') # Download the missing resource
from collections import Counter # Import Counter

tokens = nltk.word_tokenize(test_sentence)

penn_tags = nltk.pos_tag(tokens)
universal_tags = nltk.pos_tag(tokens, tagset='universal')

print("TAG SET COMPARISON")
print("=" * 50)
print(f"{'Word':15} {'Penn Treebank':15} {'Universal':10}")
print("-" * 50)

# TODO: Print comparison table
# Hint: Zip the two tag lists together
for (word, penn_tag), (word, univ_tag) in zip(penn_tags, universal_tags):
    print(f"{word:15} {penn_tag:15} {univ_tag:10}")

# Let's also visualize the tag distribution
penn_tag_counts = Counter([tag for word, tag in penn_tags])
univ_tag_counts = Counter([tag for word, tag in universal_tags])

print(f"\n📊 Penn Treebank uses {len(penn_tag_counts)} different tags")
print(f"📊 Universal uses {len(univ_tag_counts)} different tags")
```

```
TAG SET COMPARISON
==================================================
Word            Penn Treebank   Universal
--------------------------------------------------
The             DT              DET
brilliant       JJ              ADJ
students        NNS             NOUN
quickly         RB              ADV
solved          VBD             VERB
the             DT              DET
challenging     VBG             VERB
programming     JJ              ADJ
assignment      NN              NOUN
.               .               .

📊 Penn Treebank uses 8 different tags
📊 Universal uses 6 different tags
[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!
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Package universal_tagset is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger_eng is already up-to-
[nltk_data]       date!
```

## 🤔 Reflection Questions:

1. Which tag set is more detailed? Which is simpler? Enter your answer below Comprehensive:Penn Treebank approximately 45 tags; Basic: Universal Dependencies 17 tags.
2. When might you want detailed tags vs. simple tags? Enter your answer below Comprehensive for linguistic analysis or precise parsing; Basic for quick, reliable NLP in practical applications.
3. If you were building a search engine, which would you choose? Why? Enter your answer below I would choose Universal, it is quicker, adaptable across various fields, and provides sufficient information for managing queries.

---

# 🎓 End of Part 1: In-Class Exercise

Great work! You've learned the fundamentals of POS tagging and gotten hands-on experience with both NLTK and SpaCy.

## What You've Accomplished:

✅ Used NLTK and SpaCy for basic POS tagging
✅ Interpreted different tag systems
✅ Explored word ambiguity and context
✅ Compared different tagging approaches

## 🏠 Ready for Part 2?

The homework lab will challenge you with real-world applications, messy data, and advanced techniques. You'll analyze customer service transcripts, handle informal language, and benchmark different taggers.

**Take a break, then dive into Part 2 when you're ready!**

---

# 🏠 PART 2: HOMEWORK LAB

## Real-World POS Tagging Challenges

Welcome to the advanced section! Here you'll tackle the messy, complex world of real text data. This is where POS tagging gets interesting (and challenging)!

## Learning Goals for Part 2:

1. Process real-world, messy text data
2. Handle speech transcripts and informal language
3. Analyze customer service scenarios
4. Benchmark and compare different taggers
5. Understand limitations and edge cases

## 📋 Submission Requirements:

- Complete all exercises with working code
- Answer all reflection questions
- Include at least one visualization
- Submit your completed notebook file

---

# 🌍 Lab Exercise 1: Messy Text Challenge (25 minutes)

Real-world text is nothing like textbook examples! Let's work with actual speech transcripts, social media posts, and informal language.

```
import nltk
import spacy
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('universal_tagset')


nltk.download('averaged_perceptron_tagger_eng')

# Load spaCy’s English model
nlp = spacy.load("en_core_web_sm")

# Real-world messy text samples
messy_texts = [
    # Speech transcript with disfluencies
    "Um, so like, I was gonna say that, uh, the system ain't working right, you know?",

    # Social media style
```

```
    "OMG this app is sooo buggy rn 😤 cant even login smh",

    # Customer service transcript
    "Yeah hi um I'm calling because my internet's been down since like yesterday and I've tried unplugging the router thingy but it's st

    # Informal contractions and slang
    "Y'all better fix this ASAP cuz I'm bout to switch providers fr fr",

    # Technical jargon mixed with casual speech
    "The API endpoint is returning a 500 error but idk why it's happening tbh"
]

print("🔍 PROCESSING MESSY TEXT")
print("=" * 60)

# TODO: Process each messy text sample
# 1. Use both NLTK and SpaCy
# 2. Count how many words each tagger fails to recognize properly
# 3. Identify problematic words (slang, contractions, etc.)

for i, text in enumerate(messy_texts, 1):
    print(f"\n📝 Sample {i}: {text}")
    print("-" * 40)

    # NLTK processing
    nltk_tokens = nltk.word_tokenize(text)
    nltk_tags = nltk.pos_tag(nltk_tokens)

    # TODO: SpaCy processing
    spacy_doc = nlp(text)

    # TODO: Find problematic words (tagged as 'X' or unknown)
    problematic_nltk = [token[0] for token in nltk_tags if token[1] == 'X']
    problematic_spacy = [token.text for token in spacy_doc if token.pos_ == 'X']

    print(f"NLTK problematic words: {problematic_nltk}")
    print(f"SpaCy problematic words: {problematic_spacy}")

    # TODO: Calculate success rate
    nltk_success_rate = (len(nltk_tokens) - len(problematic_nltk)) / len(nltk_tokens) if len(nltk_tokens) > 0 else 0
    spacy_success_rate = (len(spacy_doc) - len(problematic_spacy)) / len(spacy_doc) if len(spacy_doc) > 0 else 0

    print(f"NLTK success rate: {nltk_success_rate:.1%}")
    print(f"SpaCy success rate: {spacy_success_rate:.1%}")
```

```
[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!
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Package universal_tagset is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger_eng is already up-to-
[nltk_data]       date!
🔍 PROCESSING MESSY TEXT
============================================================

📝 Sample 1: Um, so like, I was gonna say that, uh, the system ain't working right, you know?
----------------------------------------
NLTK problematic words: []
SpaCy problematic words: []
NLTK success rate: 100.0%
SpaCy success rate: 100.0%

📝 Sample 2: OMG this app is sooo buggy rn 😤 cant even login smh
----------------------------------------
NLTK problematic words: []
SpaCy problematic words: []
NLTK success rate: 100.0%
SpaCy success rate: 100.0%

📝 Sample 3: Yeah hi um I'm calling because my internet's been down since like yesterday and I've tried unplugging the router thingy
----------------------------------------
NLTK problematic words: []
SpaCy problematic words: []
NLTK success rate: 100.0%
SpaCy success rate: 100.0%
```

```
📝 Sample 4: Y'all better fix this ASAP cuz I'm bout to switch providers fr fr
----------------------------------------
NLTK problematic words: []
SpaCy problematic words: []
NLTK success rate: 100.0%
SpaCy success rate: 100.0%

📝 Sample 5: The API endpoint is returning a 500 error but idk why it's happening tbh
----------------------------------------
NLTK problematic words: []
SpaCy problematic words: []
NLTK success rate: 100.0%
SpaCy success rate: 100.0%
```

## 🎯 Analysis Questions:

1. Which tagger handles informal language better?
2. What types of words cause the most problems?
3. How might you preprocess text to improve tagging accuracy?
4. What are the implications for real-world applications?

## 📞 Lab Exercise 2: Customer Service Analysis Case Study (30 minutes)

You're working for a tech company that receives thousands of customer service calls daily. Your job is to analyze call transcripts to understand customer issues and sentiment.

**Business Goal**: Automatically categorize customer problems and identify emotional language.

```
# Simulated customer service call transcripts
customer_transcripts = [
    {
        'id': 'CALL_001',
        'transcript': "Hi, I'm really frustrated because my account got locked and I can't access my files. I've been trying for hours a
        'category': 'account_access'
    },
    {
        'id': 'CALL_002',
        'transcript': "Hello, I love your service but I'm having a small issue with the mobile app. It crashes whenever I try to upload
        'category': 'technical_issue'
    },
    {
        'id': 'CALL_003',
        'transcript': "Your billing system charged me twice this month! I want a refund immediately. This is ridiculous and I'm consider
        'category': 'billing'
    },
    {
        'id': 'CALL_004',
        'transcript': "I'm confused about how to use the new features you added. The interface changed and I can't find anything. Can so
        'category': 'user_guidance'
    }
]

# TODO: Analyze each transcript for:
# 1. Emotional language (adjectives that indicate sentiment)
# 2. Action words (verbs that indicate what customer wants)
# 3. Problem indicators (nouns related to issues)

analysis_results = []

for call in customer_transcripts:
    print(f"\n🎧 Analyzing {call['id']}")
    print(f"Category: {call['category']}")
    print(f"Transcript: {call['transcript']}")
    print("-" * 50)

    # Process with SpaCy
    doc = nlp(call['transcript'])

    # Extract different types of words
    emotional_adjectives = [token.text for token in doc if token.pos_ == 'ADJ']
    action_verbs = [token.text for token in doc if token.pos_ == 'VERB']
```

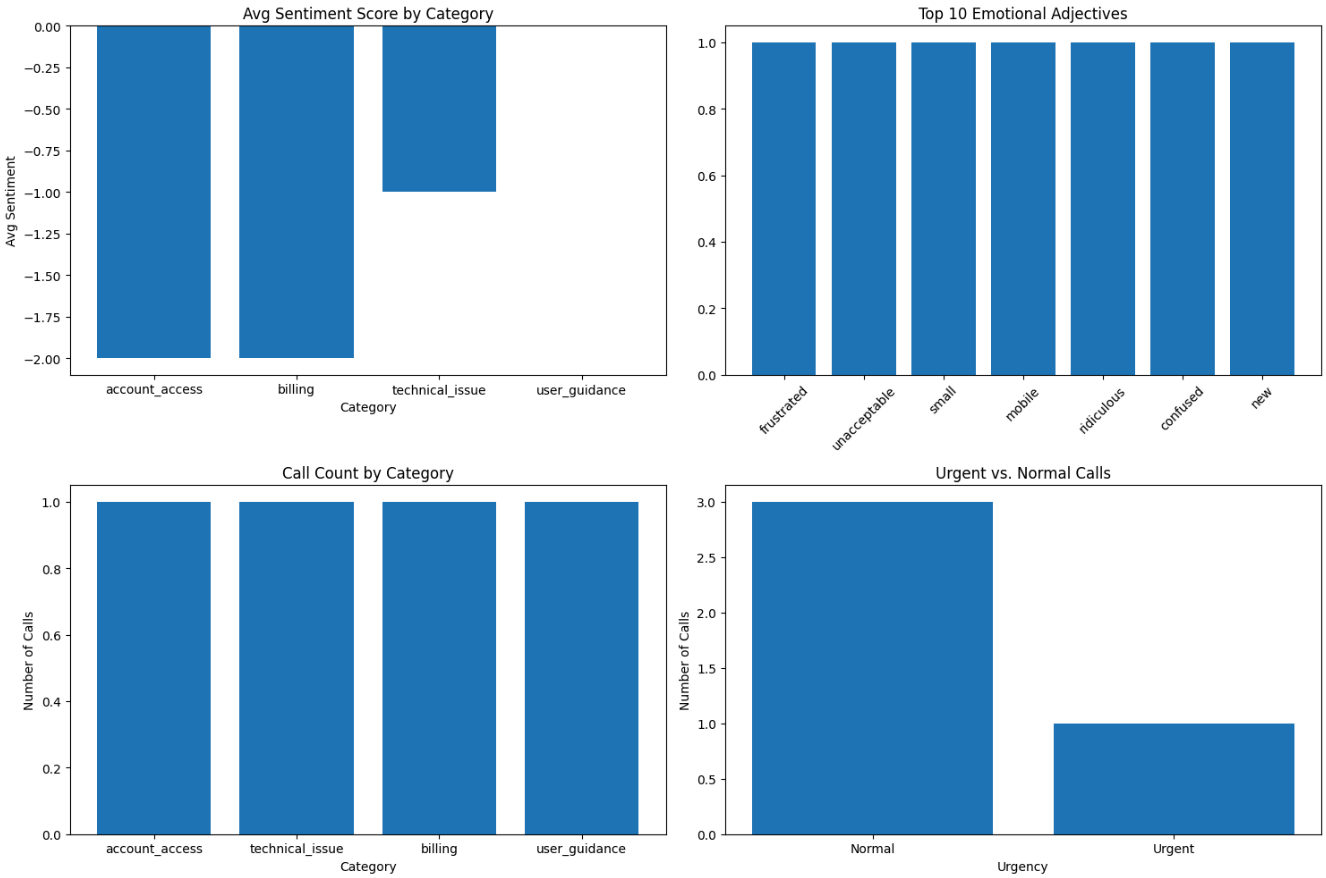
```
    problem_nouns = [token.text for token in doc if token.pos_ == 'NOUN'] # This is a basic approach; more advanced would involve checki

    # Calculate sentiment indicators
    positive_words = [token.text for token in doc if token.text.lower() in ['love', 'great', 'good', 'happy', 'pleased']]
    negative_words = [token.text for token in doc if token.text.lower() in ['frustrated', 'ridiculous', 'unacceptable', 'issue', 'proble

    result = {
        'call_id': call['id'],
        'category': call['category'],
        'transcript': call['transcript'], # Include transcript in the results
        'emotional_adjectives': emotional_adjectives,
        'action_verbs': action_verbs,
        'problem_nouns': problem_nouns,
        'sentiment_score': len(positive_words) - len(negative_words),
        'urgency_indicators': len([token.text for token in doc if token.text.lower() in ['immediately', 'asap', 'urgent']])
    }

    analysis_results.append(result)

    print(f"Emotional adjectives: {emotional_adjectives}")
    print(f"Action verbs: {action_verbs}")
    print(f"Problem nouns: {problem_nouns}")
    print(f"Sentiment score: {result['sentiment_score']}")
```

```
🎧 Analyzing CALL_001
Category: account_access
Transcript: Hi, I'm really frustrated because my account got locked and I can't access my files. I've been trying for hours and noth
--------------------------------------------------
Emotional adjectives: ['frustrated', 'unacceptable']
Action verbs: ['locked', 'access', 'trying', 'works']
Problem nouns: ['account', 'files', 'hours']
Sentiment score: -2

🎧 Analyzing CALL_002
Category: technical_issue
Transcript: Hello, I love your service but I'm having a small issue with the mobile app. It crashes whenever I try to upload photos.
--------------------------------------------------
Emotional adjectives: ['small', 'mobile']
Action verbs: ['love', 'having', 'crashes', 'try', 'upload', 'help', 'fix']
Problem nouns: ['service', 'issue', 'app', 'photos']
Sentiment score: -1

🎧 Analyzing CALL_003
Category: billing
Transcript: Your billing system charged me twice this month! I want a refund immediately. This is ridiculous and I'm considering can
--------------------------------------------------
Emotional adjectives: ['ridiculous']
Action verbs: ['charged', 'want', 'considering', 'canceling']
Problem nouns: ['billing', 'system', 'month', 'refund', 'subscription']
Sentiment score: -2

🎧 Analyzing CALL_004
Category: user_guidance
Transcript: I'm confused about how to use the new features you added. The interface changed and I can't find anything. Can someone w
--------------------------------------------------
Emotional adjectives: ['confused', 'new']
Action verbs: ['use', 'added', 'changed', 'find', 'walk']
Problem nouns: ['features', 'interface']
Sentiment score: 0
```

```
# Create a summary visualization

import matplotlib.pyplot as plt
import pandas as pd

# Convert results to DataFrame for easier analysis
df = pd.DataFrame(analysis_results)

# Create visualizations
# 1. Sentiment scores by category
# 2. Most common emotional adjectives
# 3. Action verbs frequency

fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# Plot 1 - Sentiment by category
sent_by_cat = df.groupby('category')['sentiment_score'].mean()
axes[0, 0].bar(sent_by_cat.index, sent_by_cat.values)
```

```
axes[0, 0].set_title('Avg Sentiment Score by Category')
axes[0, 0].set_xlabel('Category')
axes[0, 0].set_ylabel('Avg Sentiment')

# Plot 2 - Word frequency analysis
adj_counts = pd.Series([adj for adjs in df['emotional_adjectives'] for adj in adjs]).value_counts().head(10)
axes[0, 1].bar(adj_counts.index, adj_counts.values)
axes[0, 1].set_title('Top 10 Emotional Adjectives')
axes[0, 1].tick_params(axis='x', rotation=45)

# Plot 3 - Problem categorization
cat_counts = df['category'].value_counts()
axes[1, 0].bar(cat_counts.index, cat_counts.values)
axes[1, 0].set_title('Call Count by Category')
axes[1, 0].set_xlabel('Category')
axes[1, 0].set_ylabel('Number of Calls')

# Plot 4 - Urgency analysis
df['urgent'] = df['transcript'].str.contains(r'\b(immediately|asap|urgent)\b', case=False)
urg_counts = df['urgent'].map({True: 'Urgent', False: 'Normal'}).value_counts()
axes[1, 1].bar(urg_counts.index, urg_counts.values)
axes[1, 1].set_title('Urgent vs. Normal Calls')
axes[1, 1].set_xlabel('Urgency')
axes[1, 1].set_ylabel('Number of Calls')

plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-89-4197360677.py:37: UserWarning: This pattern is interpreted as a regular expression, and has match groups. To
  df['urgent'] = df['transcript'].str.contains(r'\b(immediately|asap|urgent)\b', case=False)
```




## 💼 Business Impact Questions:

1. How could this analysis help prioritize customer service tickets?
2. What patterns do you notice in different problem categories?
3. How might you automate the routing of calls based on POS analysis?
4. What are the limitations of this approach?

## ⚡ Lab Exercise 3: Tagger Performance Benchmarking (20 minutes)

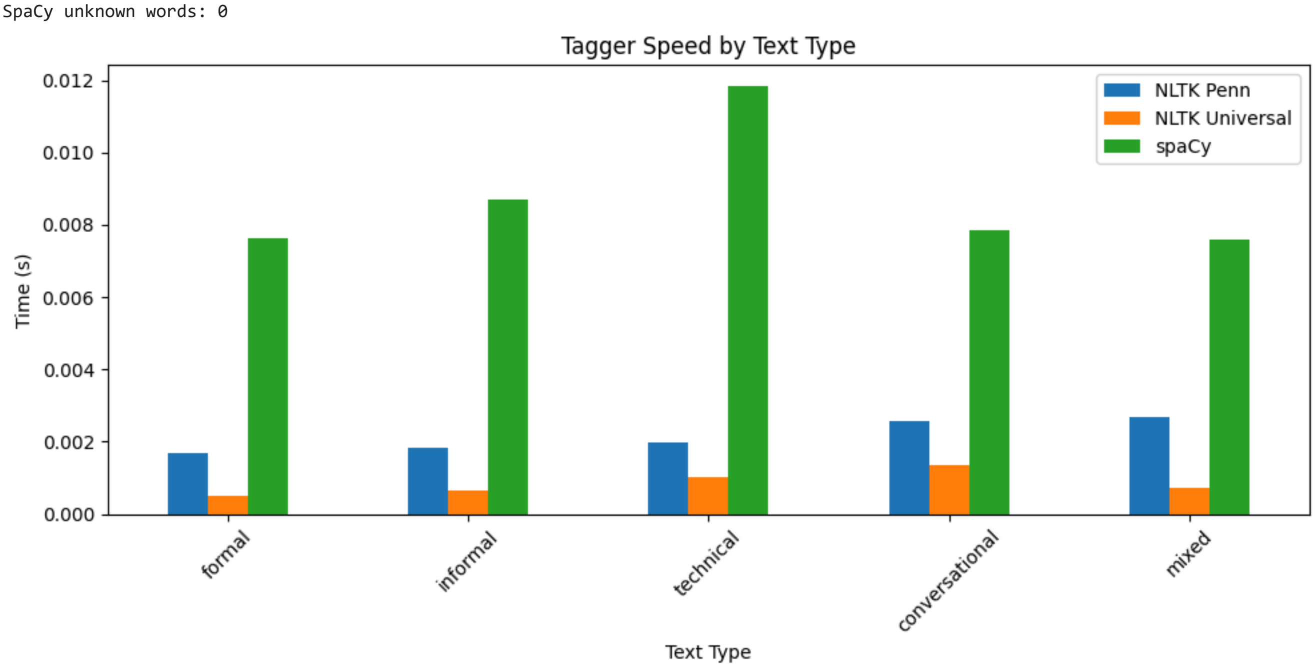
Let's scientifically compare different POS taggers on various types of text. This will help you understand when to use which tool.

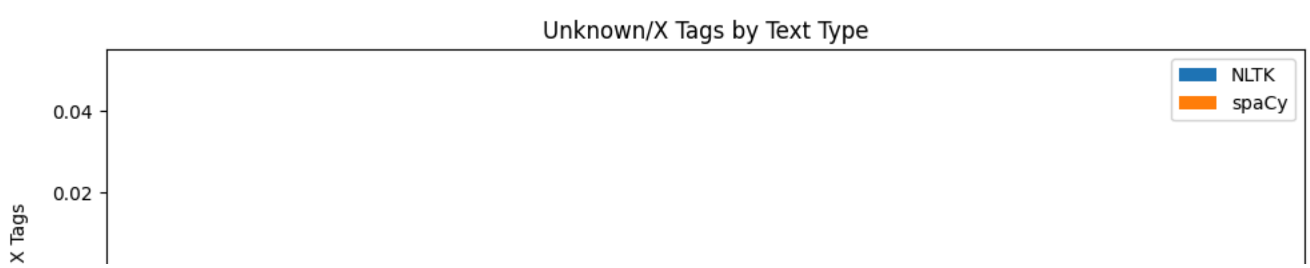
```
import time
from collections import defaultdict

# Different text types for testing
test_texts = {
```

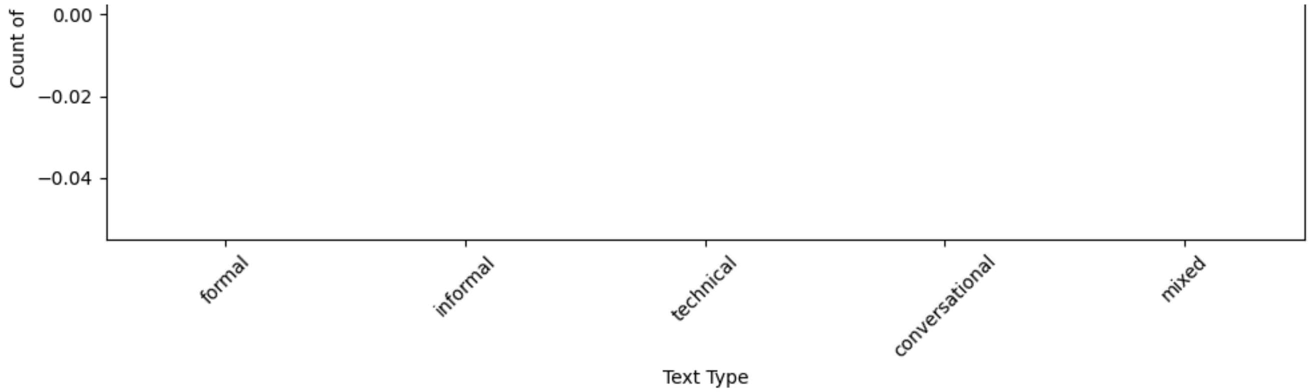
```
    'formal': "The research methodology employed in this study follows established academic protocols.",
    'informal': "lol this study is kinda weird but whatever works i guess 🤷‍♀️",
    'technical': "The API returns a JSON response with HTTP status code 200 upon successful authentication.",
    'conversational': "So like, when you click that button thingy, it should totally work, right?",
    'mixed': "OMG the algorithm's performance is absolutely terrible! The accuracy dropped to 23% wtf"
}

# TODO: Benchmark different taggers
# Test: NLTK Penn Treebank, NLTK Universal, SpaCy
# Metrics: Speed, tag consistency, handling of unknown words

benchmark_results = defaultdict(list)

for text_type, text in test_texts.items():
    print(f"\n🧪 Testing {text_type.upper()} text:")
    print(f"Text: {text}")
    print("-" * 60)

    # TODO: NLTK Penn Treebank timing
    start_time = time.time()
    nltk_penn_tags = nltk.pos_tag(nltk.word_tokenize(text))
    nltk_penn_time = time.time() - start_time  # leave this line in place
    nltk_penn_time = time.time() - start_time

    # TODO: NLTK Universal timing
    start_time = time.time()
    nltk_univ_tags = nltk.pos_tag(nltk.word_tokenize(text), tagset='universal')
    nltk_univ_time = time.time() - start_time
    start_time = time.time()
    nltk_univ_tags = nltk.pos_tag(nltk.word_tokenize(text), tagset='universal')
    nltk_univ_time = time.time() - start_time  # leave this line in place


    # TODO: SpaCy timing
    start_time = time.time()
    start_time = time.time()
    spacy_doc = nlp(text)
    spacy_tags = [(tok.text, tok.pos_) for tok in spacy_doc]
    spacy_time = time.time() - start_time  # leave this line in place
    start_time = time.time()
    spacy_doc = nlp(text)
    spacy_tags = [(tok.text, tok.pos_) for tok in spacy_doc]
    spacy_time = time.time() - start_time

    # TODO: Count unknown/problematic tags
    nltk_unknown  = sum(1 for _w, t in nltk_univ_tags if t == 'X')
    spacy_unknown = sum(1 for tok in spacy_doc   if tok.pos_ == 'X')

    # Store results
    benchmark_results[text_type] = {
        'nltk_penn_time': nltk_penn_time,
        'nltk_univ_time': nltk_univ_time,
        'spacy_time': spacy_time,
        'nltk_unknown': nltk_unknown,
        'spacy_unknown': spacy_unknown
    }

    print(f"NLTK Penn time: {nltk_penn_time:.4f}s")
    print(f"NLTK Univ time: {nltk_univ_time:.4f}s")
    print(f"SpaCy time: {spacy_time:.4f}s")
    print(f"NLTK unknown words: {nltk_unknown}")
    print(f"SpaCy unknown words: {spacy_unknown}")

# TODO: Create performance comparison visualization
import pandas as pd
import matplotlib.pyplot as plt

perf_df = pd.DataFrame.from_dict(benchmark_results, orient='index')


# Speed comparison
perf_df[['nltk_penn_time','nltk_univ_time','spacy_time']] \
    .rename(columns={
        'nltk_penn_time':'NLTK Penn',
        'nltk_univ_time':'NLTK Universal',
        'spacy_time':'spaCy'
    }) \
```

```
        .plot(kind='bar', figsize=(10,5))
    plt.title('Tagger Speed by Text Type')
    plt.ylabel('Time (s)')
    plt.xlabel('Text Type')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

    # Unknown-tag comparison
    perf_df[['nltk_unknown','spacy_unknown']] \
        .rename(columns={'nltk_unknown':'NLTK','spacy_unknown':'spaCy'}) \
        .plot(kind='bar', figsize=(10,5))
    plt.title('Unknown/X Tags by Text Type')
    plt.ylabel('Count of X Tags')
    plt.xlabel('Text Type')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

```
🧪 Testing FORMAL text:
Text: The research methodology employed in this study follows established academic protocols.
------------------------------------------------------------
NLTK Penn time: 0.0017s
NLTK Univ time: 0.0005s
SpaCy time: 0.0076s
NLTK unknown words: 0
SpaCy unknown words: 0

🧪 Testing INFORMAL text:
Text: lol this study is kinda weird but whatever works i guess 🤷
------------------------------------------------------------
NLTK Penn time: 0.0018s
NLTK Univ time: 0.0006s
SpaCy time: 0.0087s
NLTK unknown words: 0
SpaCy unknown words: 0

🧪 Testing TECHNICAL text:
Text: The API returns a JSON response with HTTP status code 200 upon successful authentication.
------------------------------------------------------------
NLTK Penn time: 0.0020s
NLTK Univ time: 0.0010s
SpaCy time: 0.0118s
NLTK unknown words: 0
SpaCy unknown words: 0

🧪 Testing CONVERSATIONAL text:
Text: So like, when you click that button thingy, it should totally work, right?
------------------------------------------------------------
NLTK Penn time: 0.0026s
NLTK Univ time: 0.0014s
SpaCy time: 0.0078s
NLTK unknown words: 0
SpaCy unknown words: 0

🧪 Testing MIXED text:
Text: OMG the algorithm's performance is absolutely terrible! The accuracy dropped to 23% wtf
------------------------------------------------------------
NLTK Penn time: 0.0027s
NLTK Univ time: 0.0007s
SpaCy time: 0.0076s
NLTK unknown words: 0
SpaCy unknown words: 0
```






### 📊 Performance Analysis:

1. Which tagger is fastest? Does speed matter for your use case?
2. Which handles informal text best?
3. How do the taggers compare on technical jargon?
4. What trade-offs do you see between speed and accuracy?

## 🚨 Lab Exercise 4: Edge Cases and Error Analysis (15 minutes)

Every system has limitations. Let's explore the edge cases where POS taggers struggle and understand why.

```
# Challenging edge cases
edge_cases = [
    "Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.",  # Famous ambiguous sentence
    "Time flies like an arrow; fruit flies like a banana.",              # Classic ambiguity
    "The man the boat the river.",                                       # Garden path sentence
    "Police police Police police police police Police police.",          # Recursive structure
    "James while John had had had had had had had had had had had a better effect on the teacher.",  # Had had had..
    "Can can can can can can can can can can.",                         # Modal/noun ambiguity
    "@username #hashtag http://bit.ly/abc123 😂🔥💯",                    # Social media elements
    "COVID-19 AI/ML IoT APIs RESTful microservices",                    # Modern technical terms
]

print("🚨 EDGE CASE ANALYSIS")
print("=" * 50)

# TODO: Process each edge case and analyze failures
for i, text in enumerate(edge_cases, 1):
    print(f"\n🔍 Edge Case {i}:")
    print(f"Text: {text}")
    print("-" * 30)

    try:
        # TODO: Process with both taggers
        nltk_tags = nltk.pos_tag(nltk.word_tokenize(text))
        spacy_doc  = nlp(text)


        # TODO: Identify potential errors or weird tags
        # Look for: repeated tags, unusual patterns, X tags, etc.

        print("NLTK tags:", [(w, t) for w, t in nltk_tags])
        print("SpaCy tags:", [(token.text, token.pos_) for token in spacy_doc])

        # TODO: Analyze what went wrong
        # YOUR ANALYSIS CODE HERE

    except Exception as e:
        print(f"❌ Error processing: {e}")

# TODO: Reflection on limitations
print("\n🤔 REFLECTION ON LIMITATIONS:")
print("=" * 40)
```

```
# YOUR REFLECTION CODE HERE
```

```
🚨 EDGE CASE ANALYSIS
==================================================

🔍 Edge Case 1:
Text: Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo.
------------------------------
NLTK tags: [('Buffalo', 'NNP'), ('buffalo', 'NN'), ('Buffalo', 'NNP'), ('buffalo', 'NN'), ('buffalo', 'NN'), ('buffalo', 'NN'), ('Bu
SpaCy tags: [('Buffalo', 'PROPN'), ('buffalo', 'NOUN'), ('Buffalo', 'PROPN'), ('buffalo', 'PROPN'), ('buffalo', 'PROPN'), ('buffalo'

🔍 Edge Case 2:
Text: Time flies like an arrow; fruit flies like a banana.
------------------------------
NLTK tags: [('Time', 'NNP'), ('flies', 'NNS'), ('like', 'IN'), ('an', 'DT'), ('arrow', 'NN'), (';', ':'), ('fruit', 'CC'), ('flies',
SpaCy tags: [('Time', 'NOUN'), ('flies', 'VERB'), ('like', 'ADP'), ('an', 'DET'), ('arrow', 'NOUN'), (';', 'PUNCT'), ('fruit', 'NOUN

🔍 Edge Case 3:
Text: The man the boat the river.
------------------------------
NLTK tags: [('The', 'DT'), ('man', 'NN'), ('the', 'DT'), ('boat', 'NN'), ('the', 'DT'), ('river', 'NN'), ('.', '.')]
SpaCy tags: [('The', 'DET'), ('man', 'NOUN'), ('the', 'DET'), ('boat', 'NOUN'), ('the', 'DET'), ('river', 'NOUN'), ('.', 'PUNCT')]

🔍 Edge Case 4:
Text: Police police Police police police police Police police.
------------------------------
NLTK tags: [('Police', 'NNP'), ('police', 'NNS'), ('Police', 'NNP'), ('police', 'NNS'), ('police', 'NN'), ('police', 'NN'), ('Police
SpaCy tags: [('Police', 'NOUN'), ('police', 'NOUN'), ('Police', 'NOUN'), ('police', 'NOUN'), ('police', 'NOUN'), ('police', 'NOUN'),

🔍 Edge Case 5:
Text: James while John had had had had had had had had had had had a better effect on the teacher.
------------------------------
NLTK tags: [('James', 'NNP'), ('while', 'IN'), ('John', 'NNP'), ('had', 'VBD'), ('had', 'VBN'), ('had', 'VBN'), ('had', 'VBN'), ('ha
SpaCy tags: [('James', 'PROPN'), ('while', 'SCONJ'), ('John', 'PROPN'), ('had', 'AUX'), ('had', 'AUX'), ('had', 'AUX'), ('had', 'AUX

🔍 Edge Case 6:
Text: Can can can can can can can can can can.
------------------------------
NLTK tags: [('Can', 'MD'), ('can', 'MD'), ('can', 'MD'), ('can', 'MD'), ('can', 'MD'), ('can', 'MD'), ('can', 'MD'), ('can', 'MD'),
SpaCy tags: [('Can', 'AUX'), ('can', 'AUX'), ('can', 'AUX'), ('can', 'AUX'), ('can', 'AUX'), ('can', 'AUX'), ('can', 'AUX'), ('can',

🔍 Edge Case 7:
Text: @username #hashtag http://bit.ly/abc123 😂 🔥 💯
------------------------------
NLTK tags: [('@', 'JJ'), ('username', 'JJ'), ('#', '#'), ('hashtag', 'JJ'), ('http', 'NN'), (':', ':'), ('//bit.ly/abc123', 'NN'), (
SpaCy tags: [('@username', 'PROPN'), ('#', 'SYM'), ('hashtag', 'NOUN'), ('http://bit.ly/abc123', 'PROPN'), ('😂', 'PROPN'), ('🔥',

🔍 Edge Case 8:
Text: COVID-19 AI/ML IoT APIs RESTful microservices
------------------------------
```