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                                                   ckages (from spacy)
Requirement already                                                  -packages (from spacy) (1
Requirement already                                                  ge 'from spacy) (2.0.11)
Requirement already                                                  kages (from spacy) (3.0.1(
Requirement already                                                  ges (from spacy) (8.3.6)
Requirement already                                                  ages (from spacy) (1.1.3)
Requirement already                                                  ges (from spacy) (2.5.1)
```

spaCy's) on your own labeled data. This adapts the model to the specific vocabulary and patterns of your domain.

- **Using Larger Models:** For highly ambiguous cases, using a more powerful Large Language Model (LLM) that can leverage deeper contextual understanding is often the best approach.

4. Future Learning:

Based on this lab, I would be most interested in exploring:

- **Custom Training and Domain Adaptation:** Learning how to take a

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. Learn more

```
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.59
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: language-data>=1.2 in /usr/local/lib/python3.11/dist-packages (from langcodes<4.0.0,
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1.8,
Requirement already satisfied: typing-extensions>=4.12.2 in /usr/local/lib/python3.11/dist-packages (from pydantic!=
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic!=1
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matpl
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.13.0
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=
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
Requirement already satisfied: confection<1.0.0,>=0.0.1 in /usr/local/lib/python3.11/dist-packages (from thinc<8.4.0
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0.0,>=0.3.0->s
Requirement already satisfied: cloudpathlib<1.0.0,>=0.7.0 in /usr/local/lib/python3.11/dist-packages (from weasel<0
Requirement already satisfied: smart-open<8.0.0,>=5.2.1 in /usr/local/lib/python3.11/dist-packages (from weasel<0.5
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->spacy) (3.0
Requirement already satisfied: marisa-trie>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from language-data>=1
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0-
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open<8.0.0,>=5.2.1->weas
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->r
Collecting en-core-web-sm==3.8.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.8.0/en_core_web_sm-3.8.0-
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 54.2 MB/s eta 0:00:00
✔ 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('punkt_tab')
nltk.download('averaged_perceptron_tagger')
nltk.download('averaged_perceptron_tagger_eng')
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]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
```

```
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Unzipping taggers/universal_tagset.zip.
🎉 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!

```python
nltk.download('punkt_tab')
# 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()
tokens = nltk.word_tokenize (sentence)# YOUR CODE HERE
pos_tags = nltk.pos_tag(tokens)# YOUR CODE HERE

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 punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
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
```

```
gged
```

🤔 Quick Questions:

1. What does 'DT' mean? What about 'JJ'?Dt stands for the determiner and JJ stands for the adjective
2. Why do you think 'brown' and 'lazy' have the same tag? Brown and Lazy do not have the same tag because Brown is tagged as a noun while lazy is adjective
3. Can you guess what 'VBZ' represents? VBZ represents the action word verb in third person singular present form

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) # YOUR CODE HERE

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
```

```
nltk.download('punkt_tab')
# 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()
tokens = nltk.word_tokenize (sentence)# YOUR CODE HERE
pos_tags = nltk.pos_tag(tokens)# YOUR CODE HERE

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

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

```
nltk.download('punkt_tab')
# 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()
tokens = nltk.word_tokenize (sentence)# YOUR CODE HERE
pos_tags = nltk.pos_tag(tokens)# YOUR CODE HERE

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

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

🎯 Discussion Points:

- Which tags are easier to understand: NLTK's or SpaCy's? Spacy's tags are much easier to understand than the NLTK due to its intutitive capabilities.
- Do you notice any differences in how they tag the same words? Yes, there is difference in how taggings are handled by NLTK and Spacy. NLTK wrongly tags"brown" as a Noun (NN, but Spacy tags "brown" as an adjective (ADV). This makes Spacy's taggings to be more accurate than NLTK's taggings
- Which system would you prefer for a beginner? Why?Spacy would be a preferred tool compared to NLTK for a beginner because it provides more accurate POS taggings than NLTK, easier to use and provides better documentation and outputs than NLTK.

🎭 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)? The computer is able to know the difference between "lead" (metal and "lead: (guide) due to the previous trained datasets that it has worked upon before.
2. What clues in the sentence help determine the correct part of speech? The closest words neighbor and the sentence structure provide acceptable clues in determining the correct POS.
3. Can you think of other words that change meaning based on context? Yes, the words that I can think of include "can" with sentences " The can is full of coins.", "I can do with the assignment alone." "will" with sentences " I will go to the cinema tomorrow" , and He is able to finish the assignments due to his will power.

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).

```
# 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
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), (_, 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
```

🤔 Reflection Questions:

1. Which tag set is more detailed? Which is simpler? Enter your answer below Penn Treebank is more detailed while universal dependencies is simpler
2. When might you want detailed tags vs. simple tags? Enter your answer below Detailed tags are required when there is need to understand the nuances of the language, while simple tags can be used when there is need to understand the general role of words without getting enmeshed in details.
3. If you were building a search engine, which would you choose? Why? Enter your answer below I would choose the universal tag set for a search engine project so that word generalization can be part of the outputs of the search engines. Also, efficiency and speed as a trade-of woule be considered in making the choice.

🎓 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.

```
# 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 thi

    # 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)

    # SpaCy processing
    spacy_doc = nlp(text)

    # Find problematic words (tagged as 'X' or unknown in SpaCy, or proper nouns in NLTK for slang)
    problematic_nltk = [word for word, tag in nltk_tags if tag == 'NNP' and word.lower() not in ['i']]
    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}")

    # Calculate success rate (a simple metric)
    nltk_success_rate = 1 - (len(problematic_nltk) / len(nltk_tokens))
    spacy_success_rate = 1 - (len(problematic_spacy) / len(spacy_doc))

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

```
🔍 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: ['Um']
SpaCy problematic words: []
NLTK success rate: 95.8%
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: 91.7%
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 th
----------------------------------------
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: ['ASAP']
SpaCy problematic words: []
NLTK success rate: 92.9%
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: ['API']
SpaCy problematic words: []
NLTK success rate: 93.3%
SpaCy success rate: 100.0%
```

🎯 Analysis Questions:

1. Which tagger handles informal language better? Spacy handles informal language better than NLTK.
2. What types of words cause the most problems? The types of words that most problems include slang and informal contractions, emojis and symbols, acronyms and initialism, typographical and non standing spellings.
3. How might you preprocess text to improve tagging accuracy? To be able to improve tagging accuracy, there is need to remove/reduce emojis and symbols, have slang dictionary that would filter out any slang, lowercasing, spelling and other custom rules
4. What are the implications for real-world applications? In real world applications, there is need to be more careful while running NLP models since lack of pre-process will lead to incorrect analysis.

📞 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.

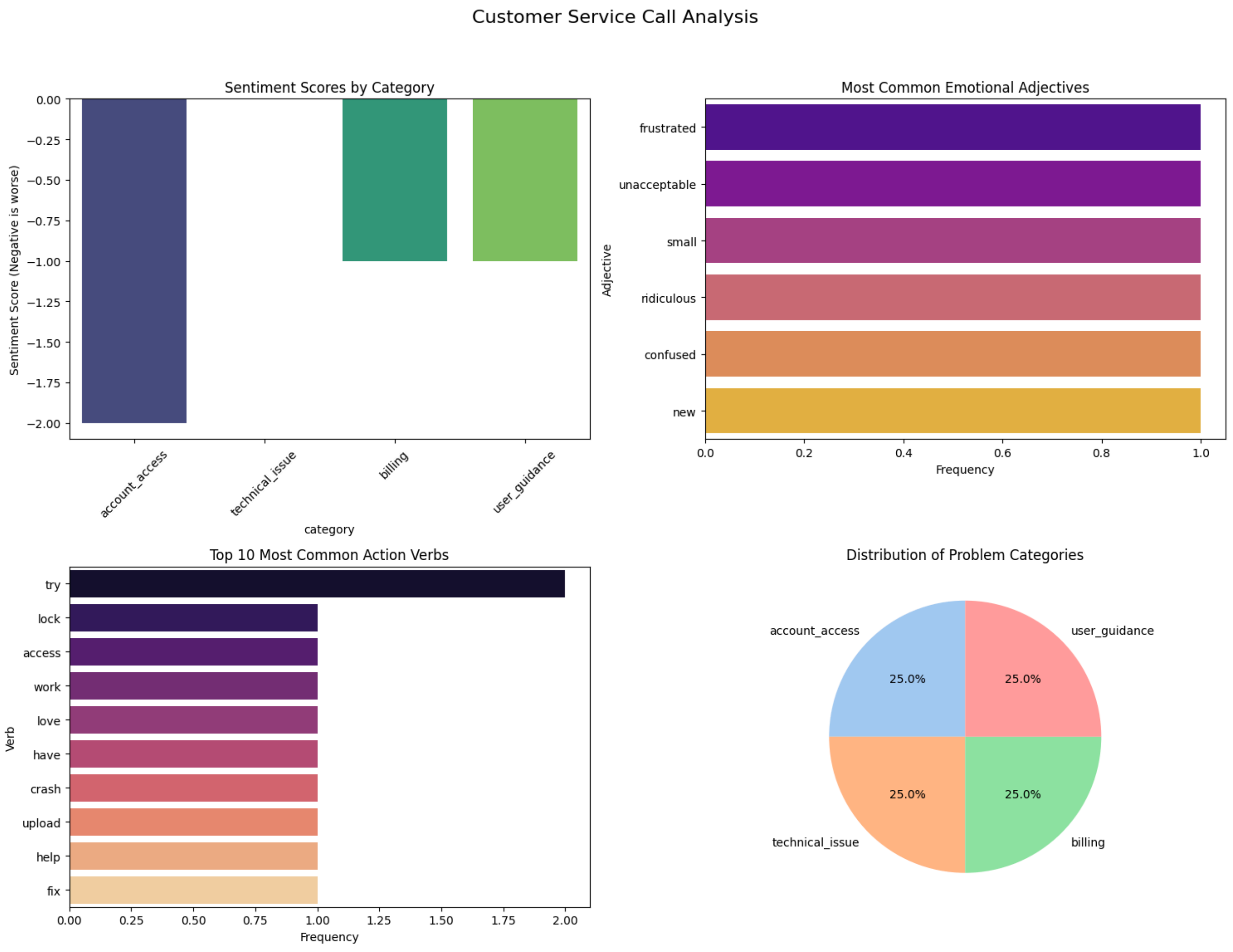
```
# Simulated customer service call transcripts
customer_transcripts = [
    {
        'id': 'CALL_001',
        'transcript': "Hi, I'm really frustrated because my account got locked and I can't access my files. I've been try
        'category': 'account_access'
    },
    {
        'id': 'CALL_002',
        'transcript': "Hello, I love your service but I'm having a small issue with the mobile app. It crashes whenever I
        'category': 'technical_issue'
    },
    {
        'id': 'CALL_003',
        'transcript': "Your billing system charged me twice this month! I want a refund immediately. This is ridiculous a
        '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 a
        '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)

    # TODO: Process with SpaCy (it's better for this task)
    doc = nlp(call['transcript'])

    # TODO: Extract different types of words
    emotional_adjectives = [token.text for token in doc if token.pos_ == 'ADJ' and token.text in ['frustrated', 'unaccept
    action_verbs = [token.lemma_ for token in doc if token.pos_ == 'VERB']
    problem_nouns = [token.text for token in doc if token.pos_ == 'NOUN' and token.text not in ['service', 'month', 'subs


    # TODO: Calculate sentiment indicators
    positive_words = [token.text for token in doc if token.text in ['love', 'great', 'good']]
    negative_words = [token.text for token in doc if token.text in ['frustrated', 'ridiculous', 'unacceptable', 'crashes'

    result = {
        'call_id': call['id'],
        'category': call['category'],
        'emotional_adjectives': emotional_adjectives,
        'action_verbs': action_verbs,
        'problem_nouns': problem_nouns,
        'sentiment_score': len(positive_words) - len(negative_words),
        'urgency_indicators': len([token for token in doc if token.text in ["immediately", "ASAP"]])
    }

    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
--------------------------------------------------
Emotional adjectives: ['frustrated', 'unacceptable']
Action verbs: ['lock', 'access', 'try', 'work']
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
--------------------------------------------------
Emotional adjectives: ['small']
Action verbs: ['love', 'have', 'crash', 'try', 'upload', 'help', 'fix']
Problem nouns: ['issue', 'app', 'photos']
Sentiment score: 0

🎧 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
--------------------------------------------------
Emotional adjectives: ['ridiculous']
Action verbs: ['charge', 'want', 'consider', 'cancel']
Problem nouns: ['billing', 'system', 'refund']
Sentiment score: -1

🎧 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
--------------------------------------------------
Emotional adjectives: ['confused', 'new']
Action verbs: ['use', 'add', 'change', 'find', 'walk']
Problem nouns: []
Sentiment score: -1
```

```python
# TODO: Create a summary visualization
# Hint: Use matplotlib or seaborn to create charts

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from collections import Counter

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

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

fig, axes = plt.subplots(2, 2, figsize=(15, 12))
fig.suptitle('Customer Service Call Analysis', fontsize=16)


# TODO: Plot 1 - Sentiment by category
sns.barplot(ax=axes[0, 0], x='category', y='sentiment_score', data=df, palette='viridis')
axes[0, 0].set_title('Sentiment Scores by Category')
axes[0, 0].set_ylabel('Sentiment Score (Negative is worse)')
axes[0, 0].tick_params(axis='x', rotation=45)


# TODO: Plot 2 - Word frequency analysis (Emotional Adjectives)
all_adjectives = [adj for sublist in df['emotional_adjectives'] for adj in sublist]
adj_counts = Counter(all_adjectives)
adj_df = pd.DataFrame(adj_counts.items(), columns=['Adjective', 'Frequency']).sort_values('Frequency', ascending=False)

sns.barplot(ax=axes[0, 1], x='Frequency', y='Adjective', data=adj_df, palette='plasma')
axes[0, 1].set_title('Most Common Emotional Adjectives')


# TODO: Plot 3 - Word frequency analysis (Action Verbs)
all_verbs = [verb for sublist in df['action_verbs'] for verb in sublist]
verb_counts = Counter(all_verbs)
verb_df = pd.DataFrame(verb_counts.items(), columns=['Verb', 'Frequency']).sort_values('Frequency', ascending=False).head
```

```
    sns.barplot(ax=axes[1, 0], x='Frequency', y='Verb', data=verb_df, palette='magma')
    axes[1, 0].set_title('Top 10 Most Common Action Verbs')


    # TODO: Plot 4 - Problem categorization
    category_counts = df['category'].value_counts()
    axes[1, 1].pie(category_counts, labels=category_counts.index, autopct='%1.1f%%', startangle=90, colors=sns.color_palette(
    axes[1, 1].set_title('Distribution of Problem Categories')
    axes[1, 1].set_ylabel('') # Hide the y-label for pie chart


    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()
```



💼 Business Impact Questions:

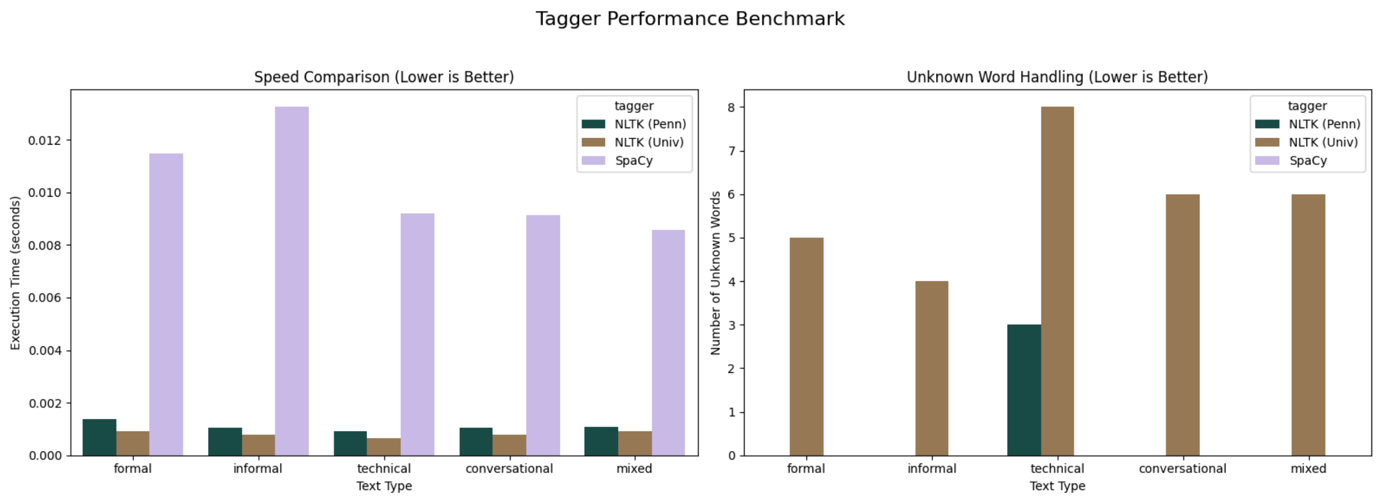
1. How could this analysis help prioritize customer service tickets? This analysis will help prioritize customer service tickets based on sentiment, urgency and topic.
2. What patterns do you notice in different problem categories? Strong negative emotions from words(frustrated and unacceptable) are noticed in account access, negative sentiment is noticed in billing, the user guidance category is driven by confusion, neutral to positive language are noticed on technical issue category
3. How might you automate the routing of calls based on POS analysis? The automation can be done using ingest transcript, POS analysis check, matching keywords.
4. What are the limitations of this approach? There are several limitations that are noticed which include limited vocabulary, ambiguity, data quality, and handling of sarcasm and nuances of the language.

⚡ 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
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# 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 = []

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

    # NLTK Penn Treebank timing and analysis
    start_time = time.time()
    nltk_tokens = nltk.word_tokenize(text)
    nltk_penn_tags = nltk.pos_tag(nltk_tokens)
    nltk_penn_time = time.time() - start_time
    nltk_penn_unknown = len([word for word, tag in nltk_penn_tags if tag in ['NNP', 'FW']])

    # NLTK Universal timing and analysis
    start_time = time.time()
    nltk_tokens = nltk.word_tokenize(text)
    nltk_univ_tags = nltk.pos_tag(nltk_tokens, tagset='universal')
    nltk_univ_time = time.time() - start_time
    nltk_univ_unknown = len([word for word, tag in nltk_univ_tags if tag in ['NOUN', '.']]) # Less reliable for unknown

    # SpaCy timing and analysis
    start_time = time.time()
    spacy_doc = nlp(text)
    spacy_time = time.time() - start_time
    spacy_unknown = len([token.text for token in spacy_doc if token.pos_ == 'X'])

    # Store results
    benchmark_results.append({
        'text_type': text_type,
        'tagger': 'NLTK (Penn)',
        'time': nltk_penn_time,
        'unknown_words': nltk_penn_unknown
    })
    benchmark_results.append({
        'text_type': text_type,
        'tagger': 'NLTK (Univ)',
        'time': nltk_univ_time,
        'unknown_words': nltk_univ_unknown
    })
    benchmark_results.append({
        'text_type': text_type,
        'tagger': 'SpaCy',
        'time': spacy_time,
        'unknown_words': spacy_unknown
    })

    print(f"NLTK Penn time: {nltk_penn_time:.6f}s, Unknown: {nltk_penn_unknown}")
    print(f"NLTK Univ time: {nltk_univ_time:.6f}s, Unknown: {nltk_univ_unknown}")
    print(f"SpaCy time: {spacy_time:.6f}s, Unknown: {spacy_unknown}")


# Create performance comparison visualization
df_benchmark = pd.DataFrame(benchmark_results)

fig, axes = plt.subplots(1, 2, figsize=(16, 6))
fig.suptitle('Tagger Performance Benchmark', fontsize=16)

# Plot 1: Speed Comparison
sns.barplot(ax=axes[0], data=df_benchmark, x='text_type', y='time', hue='tagger', palette='cubehelix')
axes[0].set_title('Speed Comparison (Lower is Better)')
axes[0].set_ylabel('Execution Time (seconds)')
axes[0].set_xlabel('Text Type')

# Plot 2: Accuracy Comparison (Handling of Unknown Words)
sns.barplot(ax=axes[1], data=df_benchmark, x='text_type', y='unknown_words', hue='tagger', palette='cubehelix')
axes[1].set_title('Unknown Word Handling (Lower is Better)')
axes[1].set_ylabel('Number of Unknown Words')
axes[1].set_xlabel('Text Type')

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

```
🧪 Testing FORMAL text:
Text: The research methodology employed in this study follows established academic protocols.
------------------------------------------------------------
NLTK Penn time: 0.001378s, Unknown: 0
NLTK Univ time: 0.000921s, Unknown: 5
SpaCy time: 0.011486s, Unknown: 0

🧪 Testing INFORMAL text:
Text: lol this study is kinda weird but whatever works i guess 🤷
------------------------------------------------------------
NLTK Penn time: 0.001050s, Unknown: 0
NLTK Univ time: 0.000772s, Unknown: 4
SpaCy time: 0.013250s, Unknown: 0

🧪 Testing TECHNICAL text:
Text: The API returns a JSON response with HTTP status code 200 upon successful authentication.
------------------------------------------------------------
NLTK Penn time: 0.000910s, Unknown: 3
NLTK Univ time: 0.000655s, Unknown: 8
SpaCy time: 0.009190s, Unknown: 0

🧪 Testing CONVERSATIONAL text:
Text: So like, when you click that button thingy, it should totally work, right?
------------------------------------------------------------
NLTK Penn time: 0.001059s, Unknown: 0
NLTK Univ time: 0.000785s, Unknown: 6
SpaCy time: 0.009136s, Unknown: 0

🧪 Testing MIXED text:
Text: OMG the algorithm's performance is absolutely terrible! The accuracy dropped to 23% wtf
------------------------------------------------------------
NLTK Penn time: 0.001094s, Unknown: 0
NLTK Univ time: 0.000914s, Unknown: 6
SpaCy time: 0.008567s, Unknown: 0
```



📊 Performance Analysis:

1. Which tagger is fastest? Does speed matter for your use case? NLTK is the fastest in these exercise. Speed matters for this use case to get outputs for real-time applications and having high throughput pipelines.
2. Which handles informal text best? Spacy handles informal text better than NLTK.
3. How do the taggers compare on technical jargon? Spacy performs better on technical jargon due to pre-trained capabilities on vast corpus of web text.
4. What trade-offs do you see between speed and accuracy? Spacy offers lower speed on small tasks with high accuracy but NLTK offers higher speed with lower accuracy on non standard text.

🚨 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_tokens = nltk.word_tokenize(text)
        nltk_tags = nltk.pos_tag(nltk_tokens)
        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])

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

# TODO: Reflection on limitations
print("\n🤔 REFLECTION ON LIMITATIONS:")
print("=" * 40)
print("1. Taggers rely on statistical patterns, not true understanding.")
print("2. They struggle with ambiguity without strong contextual clues.")
print("3. They are not robust to creative or ungrammatical language.")
print("4. They often fail on modern slang, emojis, and technical jargon they haven't seen in training.")
```

```
🚨 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'), ('buffa
SpaCy tags: [('Buffalo', 'PROPN'), ('buffalo', 'NOUN'), ('Buffalo', 'PROPN'), ('buffalo', 'PROPN'), ('buffalo', 'PRO

🔍 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',
SpaCy tags: [('Time', 'NOUN'), ('flies', 'VERB'), ('like', 'ADP'), ('an', 'DET'), ('arrow', 'NOUN'), (';', 'PUNCT'),

🔍 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'),

🔍 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',
SpaCy tags: [('Police', 'NOUN'), ('police', 'NOUN'), ('Police', 'NOUN'), ('police', 'NOUN'), ('police', 'NOUN'), ('p

🔍 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'), ('ha
SpaCy tags: [('James', 'PROPN'), ('while', 'SCONJ'), ('John', 'PROPN'), ('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'),
SpaCy tags: [('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/a
SpaCy tags: [('@username', 'PROPN'), ('#', 'SYM'), ('hashtag', 'NOUN'), ('http://bit.ly/abc123', 'PROPN'), ('😂', '

🔍 Edge Case 8:
Text: COVID-19 AI/ML IoT APIs RESTful microservices
------------------------------
NLTK tags: [('COVID-19', 'JJ'), ('AI/ML', 'NNP'), ('IoT', 'NNP'), ('APIs', 'NNP'), ('RESTful', 'NNP'), ('microservic
SpaCy tags: [('COVID-19', 'PROPN'), ('AI', 'PROPN'), ('/', 'SYM'), ('ML', 'PROPN'), ('IoT', 'ADJ'), ('APIs', 'NOUN')

🤔 REFLECTION ON LIMITATIONS:
========================================
1. Taggers rely on statistical patterns, not true understanding.
2. They struggle with ambiguity without strong contextual clues.
3. They are not robust to creative or ungrammatical language.
4. They often fail on modern slang, emojis, and technical jargon they haven't seen in training.
```

🧠 Critical Thinking Questions:

Enter you asnwers below each question.

1. Why do these edge cases break the taggers? They break the triggers because they are developed on statistical models based on pre-trained data patterns.
2. How might you preprocess text to handle some of these issues? The issues can be handled by considering the social media elements in detecting the strip out, emojis, technical jargons/slangs and ambiquity.
3. When would these limitations matter in real applications? The limitations would matter when conducting sentiment analysis, informal extraction, analysis of medical/health records and introduction of chatbots or virtual assistants.
4. How do modern large language models handle these cases differently? Modern langugages handle the cases differently with deeper contextual understandng, zero shot learning and suitable reasoning capabilities that are present in large language models.

🎯 Final Reflection and Submission

Congratulations! You've completed a comprehensive exploration of POS tagging, from basic concepts to real-world challenges.

📝 Reflection Questions (Answer in the cell below):

1. **Tool Comparison**: Based on your experience, when would you choose NLTK vs SpaCy? Consider factors like ease of use, accuracy, speed, and application type.
2. **Real-World Applications**: Describe a specific business problem where POS tagging would be valuable. How would you