Lab 02: Basic NLP Preprocessing Techniques

Course: ITAI 2373 - Natural Language Processing
Module: 02 - Text Preprocessing
Duration: 2-3 hours
Student Name: ____
Date: ____

🎯 Learning Objectives

By completing this lab, you will:

1. Understand the critical role of preprocessing in NLP pipelines
2. Master fundamental text preprocessing techniques
3. Compare different libraries and their approaches
4. Analyze the effects of preprocessing on text data
5. Build a complete preprocessing pipeline
6. Load and work with different types of text datasets

📖 Introduction to NLP Preprocessing

Natural Language Processing (NLP) preprocessing refers to the initial steps taken to clean and transform raw text data into a format that's more suitable for analysis by machine learning algorithms.

Why is preprocessing crucial?

1. **Standardization:** Ensures consistent text format across your dataset
2. **Noise Reduction:** Removes irrelevant information that could confuse algorithms
3. **Complexity Reduction:** Simplifies text to focus on meaningful patterns
4. **Performance Enhancement:** Improves the efficiency and accuracy of downstream tasks

Real-world Impact

Consider searching for "running shoes" vs "Running Shoes!" - without preprocessing, these might be treated as completely different queries. Preprocessing ensures they're recognized as equivalent.

__init__.py NameError ×

Please explain the error:

```
NameError: name 'nltk_tokens' is nc
```

Hypothesis

The `NameError: name 'nltk_tokens' is not defined` traceback indicates that the variable `nltk_tokens` was accessed before it was assigned a value.

Looking at the code, the line `#nltk_tokens = word_tokenize(simple_text)` is commented out. This line is responsible for assigning the list of tokens to the `nltk_tokens` variable. Consequently, when the code attempts to print `nltk_tokens` in the following line `print(f"\nWord tokens: {nltk_tokens}")`, the variable does not exist in the current scope, leading to the `NameError`.

Suggested Changes

To fix this error, uncomment the line that performs the word tokenization and assigns the result to `nltk_tokens`. Additionally, the code section after the first `print("NLTK Tokenization Results")` is a duplicate and can be removed.

```
# Step 4: Tokenization with NLTK

!pip install nltk

from nltk.tokenize import wor

# Test on simple text
print("NLTK Tokenization Results")
print("=" * 40)
print(f"Original: {simple_text}")

# Word tokenization
nltk_tokens = word_tokenize(simple_
print(f"\nWord tokens: {nltk_tokens
print(f"Number of tokens: {len(nltk
```

🤔 Conceptual Question 1

Before we start coding, think about your daily interactions with text processing systems (search engines, chatbots, translation apps). What challenges do you think these systems face when processing human language? List at least 3 specific challenges and explain why each is problematic.

Double-click this cell to write your answer:

Challenge 1:

1. Ambiguity in Language (Lexical & Syntactic Ambiguity) Problem: Words or phrases can possess various meanings based on the context. For instance, the term "bank" might refer to a financial institution or the edge of a river. Likewise, a statement like "I saw the man with the telescope" could imply that I used a telescope or that the man was in possession of a telescope.

Why it's problematic: Text processing systems need to accurately discern the intended meaning to deliver precise outcomes. Misinterpretation can result in erroneous responses from chatbots, irrelevant search results, or flawed translations.

Challenge 2:

2. Context Comprehension and Global Awareness Problem: Human interaction is significantly dependent on context, cultural allusions, and collective knowledge. For instance, the expression "That movie was fire" conveys a positive sentiment in contemporary slang, yet a literal interpretation by a system could lead to a misunderstanding as something harmful. Why it poses a challenge: In the absence of context or foundational knowledge, systems risk generating responses that are inappropriate, obsolete, or even offensive.

Challenge 3:

3. Managing Informal or Non-Standard Language Issue: Individuals frequently utilize slang, abbreviations, emojis, typographical errors, or regional dialects. For instance: "idk what u mean lol 🤷" significantly contrasts with formal grammatical structures. Reasons for concern: The majority of models are developed using standard language and may find it challenging to comprehend or respond appropriately to informal phrases, resulting in a frustrating experience for users.

🛠️ Part 1: Environment Setup

```
# Sentence tokenization
sentences = sent_tokenize(simple_te
print(f"\nSentences: {sentences}")
print(f"Number of sentences: {len(s

# Remove the duplicated code sectic
# print("NLTK Tokenization Results'
# print("=" * 40)
# print(f"Original: {simple_text}")

# # Word tokenization
# nltk_tokens = word_tokenize(simpl
# print(f"\nWord tokens: {nltk_toke
# print(f"Number of tokens: {len(nl

# # Sentence tokenization
# sentences = sent_tokenize(simple_
# print(f"\nSentences: {sentences}'
# print(f"Number of sentences: {ler
```

Use code with caution

Rate this answer

still not working

Rate this answer

We'll be working with two major NLP libraries:

- **NLTK (Natural Language Toolkit):** Comprehensive NLP library with extensive resources
- **spaCy:** Industrial-strength NLP with pre-trained models

⚠️ **Note:** Installation might take 2-3 minutes to complete.

```
# Step 1: Install Required Libraries
print("🔧 Installing NLP libraries...")

!pip install -q nltk spacy
!python -m spacy download en_core_web_sm

print("✅ Installation complete!")
```

```
🔧 Installing NLP libraries...
Collecting en-core-web-sm==3.8.0
  Downloading https://github.com/explosion/spacy-models/releas
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 44.
✓ 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 res
order to load all the package's dependencies. You can do this
'Restart kernel' or 'Restart runtime' option.
✅ Installation complete!
```

🤔 Conceptual Question 2

Why do you think we need to install a separate language model (en_core_web_sm) for spaCy? What components might this model contain that help with text processing? Think about what information a computer needs to understand English text.

Double-click this cell to write your answer:

It is essential to install `en_core_web_sm` in spaCy, as the framework alone does not encompass the language-specific data required for processing English. This model offers pre-trained components that facilitate the system's ability to comprehend and analyze text.

Key Components: Tokenizer: Divides text into words and punctuation. POS Tagger: Recognizes parts of speech such as nouns and verbs. Dependency Parser: Examines grammar and the relationships between words. NER Named Entity Recognizer: Identifies names, dates, locations, and more. Lemmatizer: Determines the base form of words for instance, "running" becomes "run".

These instruments enable spaCy to effectively interpret and engage with natural language.

```
# Step 2: Import Libraries and Download NLTK Data
import nltk
import spacy
import string
import re
from collections import Counter

# Download essential NLTK data
print("📦 Downloading NLTK data packages...")
nltk.download('punkt')      # For tokenization
nltk.download('stopwords')  # For stop word removal
nltk.download('wordnet')    # For lemmatization
nltk.download('averaged_perceptron_tagger')  # For POS tagging

print("\n✅ All imports and downloads completed!")
```

```
📦 Downloading NLTK data packages...

✅ All imports and downloads completed!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data..
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet 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
[nltk_data]       date!
```

📁 Part 2: Sample Text Data

We'll work with different types of text to understand how preprocessing affects various text styles:

- Simple text
- Academic text (with citations, URLs)
- Social media text (with emojis, hashtags)
- News text (formal writing)
- Product reviews (informal, ratings)

```
# Step 3: Load Sample Texts
simple_text = "Natural Language Processing is a fascinating field

academic_text = """
Dr. Smith's research on machine-learning algorithms is groundbreak
She published 3 papers in 2023, focusing on deep neural networks (
The results were amazing - accuracy improved by 15.7%!
"This is revolutionary," said Prof. Johnson.
Visit https://example.com for more info. #NLP #AI @university
"""

social_text = "OMG! Just tried the new coffee shop ☕ SO GOOD!!!

news_text = """
The stock market experienced significant volatility today, with te
```

```
Apple Inc. (AAPL) dropped 3.2%, while Microsoft Corp. fell 2.8%.
"We're seeing a rotation out of growth stocks," said analyst Jane
"""

review_text = """
This laptop is absolutely fantastic! I've been using it for 6 mont
The battery life is incredible - lasts 8-10 hours easily.
Only complaint: the keyboard could be better. Overall rating: 4.5/
"""

# Store all texts
sample_texts = {
    "Simple": simple_text,
    "Academic": academic_text.strip(),
    "Social Media": social_text,
    "News": news_text.strip(),
    "Product Review": review_text.strip()
}

print("📄 Sample texts loaded successfully!")
for name, text in sample_texts.items():
    preview = text[:80] + "..." if len(text) > 80 else text
    print(f"\n🏷️ {name}: {preview}")
```

```
📄 Sample texts loaded successfully!

🏷️ Simple: Natural Language Processing is a fascinating field

🏷️ Academic: Dr. Smith's research on machine-learning algorit
She publi...

🏷️ Social Media: OMG! Just tried the new coffee shop ☕ SO G(

🏷️ News: The stock market experienced significant volatility

🏷️ Product Review: This laptop is absolutely fantastic! I've
```

🤔 Conceptual Question 3

Looking at the different text types we've loaded, what preprocessing challenges do you anticipate for each type? For each text type below, identify at least 2 specific preprocessing challenges and explain why they might be problematic for NLP analysis.

Double-click this cell to write your answer:

Simple text challenges:

1. Absence of Context: Concise sentences hinder models from grasping meaning or connections.
2. Restricted Vocabulary: Fundamental words and structures diminish the complexity required for advanced NLP tasks.

Academic text challenges:

1. Citations and URLs: These elements introduce noise and may complicate tokenization or analysis.

2. Complex Structure: Extended sentences containing technical terminology increase the difficulty of parsing and entity recognition.

Social media text challenges:

1. Slang and Abbreviations: Informal expressions such as "LOL" or "idk" pose difficulties for models in terms of interpretation.
2. Emojis and Hashtags: Non-standard characters may skew sentiment or topic analysis if not appropriately managed.

News text challenges:

1. Formal and Dense Language: The use of intricate sentence constructions can hinder effective parsing and summarization.
2. Named Entities: The regular occurrence of individuals, locations, and organizations necessitates precise entity recognition for comprehensive understanding.

Product review challenges:

1. Mixed Sentiment: A review can encompass both favorable and unfavorable viewpoints, complicating the process of sentiment analysis.
2. Informal Language and Emojis: Misspellings, colloquialisms, and the use of emojis may perplex models unless they are adequately normalized.

🔤 Part 3: Tokenization

What is Tokenization?

Tokenization is the process of breaking down text into smaller, meaningful units called **tokens**. These tokens are typically words, but can also be sentences, characters, or subwords.

Why is it Important?

- Most NLP algorithms work with individual tokens, not entire texts
- It's the foundation for all subsequent preprocessing steps
- Different tokenization strategies can significantly impact results

Common Challenges:

- **Contractions:** "don't" → "do" + "n't" or "don't"?
- **Punctuation:** Keep with words or separate?
- **Special characters:** How to handle @, #, URLs?

```
!pip install --upgrade nltk
```

```
# 2) Import and inspect data paths
import nltk
print("NLTK version:", nltk.__version__)
print("NLTK data paths:", nltk.data.path)

# 3) Download the standard punkt tokenizer (and fallback punkt_tab)
nltk.download('punkt', quiet=False)
nltk.download('punkt_tab', quiet=False)


# Step 4: Tokenization with NLTK

from nltk.tokenize import word_tokenize, sent_tokenize

# Test on simple text
print("NLTK Tokenization Results")
print("=" * 40)
print(f"Original: {simple_text}")

# Word tokenization
nltk_tokens = word_tokenize(simple_text)
print(f"\nWord tokens: {nltk_tokens}")
print(f"Number of tokens: {len(nltk_tokens)}")

# Sentence tokenization
sentences = sent_tokenize(simple_text)
print(f"\nSentences: {sentences}")
print(f"Number of sentences: {len(sentences)}")
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.
Requirement already satisfied: click in /usr/local/lib/python3
Requirement already satisfied: joblib in /usr/local/lib/pythor
Requirement already satisfied: regex>=2021.8.3 in /usr/local/l
Requirement already satisfied: tqdm in /usr/local/lib/python3.
NLTK version: 3.9.1
NLTK data paths: ['/root/nltk_data', '/usr/nltk_data', '/usr/s
NLTK Tokenization Results
========================================
Original: Natural Language Processing is a fascinating field c

Word tokens: ['Natural', 'Language', 'Processing', 'is', 'a',
Number of tokens: 14

Sentences: ['Natural Language Processing is a fascinating fiel
Number of sentences: 2
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt 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!
```

🤔 Conceptual Question 4

Examine the NLTK tokenization results above. How did NLTK handle the contraction "It's"? What happened to the punctuation marks? Do you think this approach is appropriate for all NLP tasks? Explain your reasoning.

Double-click this cell to write your answer:

How "It's" was handled: NLTK divides "It's" into two distinct tokens: "It" and "'s". This indicates that NLTK processes contractions by isolating the root word from the suffix of the contraction.

Punctuation treatment: Punctuation symbols such as "." and "!" are regarded as **individual tokens**, separate from the words they follow.

Appropriateness for different tasks: This methodology is effective for **syntactic analysis**, **part-of-speech tagging**, and **grammar-oriented NLP tasks**, where the disaggregation of words and punctuation aids in comprehending structure. Conversely, for tasks such as **sentiment analysis** or **machine translation**, maintaining the integrity of contractions and punctuation may more accurately reflect tone, emotion, and meaning. Therefore, while NLTK's approach is advantageous for general preprocessing, certain tasks might gain from utilizing more context-sensitive tokenizers (such as spaCy or BERT-based models).

```
# Step 5: Tokenization with spaCy
nlp = spacy.load('en_core_web_sm')

print("🔍 spaCy Tokenization Results")
print("=" * 40)
print(f"Original: {simple_text}")

# Process with spaCy
doc = nlp(simple_text)

# Extract tokens
spacy_tokens = [token.text for token in doc]
print(f"\nWord tokens: {spacy_tokens}")
print(f"Number of tokens: {len(spacy_tokens)}")

# Show detailed token information
print(f"\n🔬 Detailed Token Analysis:")
print(f"{'Token':<12} {'POS':<8} {'Lemma':<12} {'Is Alpha':<8} {'Is
print("-" * 50)
for token in doc:
    print(f"{token.text:<12} {token.pos_:<8} {token.lemma_:<12} {to
```

```
🔍 spaCy Tokenization Results
========================================
Original: Natural Language Processing is a fascinating field c

Word tokens: ['Natural', 'Language', 'Processing', 'is', 'a',
Number of tokens: 14

🔬 Detailed Token Analysis:
Token        POS      Lemma        Is Alpha Is Stop
--------------------------------------------------
Natural      PROPN    Natural      1        0
Language     PROPN    Language     1        0
Processing   NOUN     processing   1        0
is           AUX      be           1        1
a            DET      a            1        1
```

```
fascinating  ADJ    fascinating  1     0
field        NOUN   field        1     0
of           ADP    of           1     1
AI           PROPN  AI           1     0
.            PUNCT  .            0     0
It           PRON   it           1     1
's           AUX    be           0     1
amazing      ADJ    amazing      1     0
!            PUNCT  !            0     0
```

🤔 Conceptual Question 5

Compare the NLTK and spaCy tokenization results. What differences do you notice? Which approach do you think would be better for different NLP tasks? Consider specific examples like sentiment analysis vs. information extraction.

Double-click this cell to write your answer:

Key differences observed: Similar to NLTK, spaCy divides "It's" into "It" and "'s" while also separating punctuation. Additionally, it incorporates part-of-speech, lemma, and stopword information.

Better for sentiment analysis: A tokenizer that retains contractions and punctuation (such as spaCy) might yield superior results by maintaining emotional nuance and context.

Better for information extraction: NLTK's precise token separation is helpful for tasks like part-of-speech tagging and entity recognition.spaCy demonstrates greater strength by offering structured linguistic features, including POS tags and named entities, readily available.

Overall assessment: NLTK is good for structured analysis, but may require customization for tasks needing more contextual understanding. SpaCy provides more profound insights and is more adaptable for complex NLP tasks when compared to the fundamental tokenization capabilities of NLTK

```
# Step 6: Test Tokenization on Complex Text
print("🧪 Testing on Social Media Text")
print("=" * 40)
print(f"Original: {social_text}")

# NLTK approach
social_nltk_tokens = word_tokenize(social_text)
print(f"\nNLTK tokens: {social_nltk_tokens}")

# spaCy approach
social_doc = nlp(social_text)
social_spacy_tokens = [token.text for token in social_doc]
print(f"spaCy tokens: {social_spacy_tokens}")
```

```
print(f"\n📊 Comparison:")
print(f"NLTK token count: {len(social_nltk_tokens)}")
print(f"spaCy token count: {len(social_spacy_tokens)}")
```

```
🧪 Testing on Social Media Text
========================================
Original: OMG! Just tried the new coffee shop ☕ SO GOOD!!! H

NLTK tokens: ['OMG', '!', 'Just', 'tried', 'the', 'new', 'coff
spaCy tokens: ['OMG', '!', 'Just', 'tried', 'the', 'new', 'cof

📊 Comparison:
NLTK token count: 22
spaCy token count: 23
```

🤔 Conceptual Question 6

Looking at how the libraries handled social media text (emojis, hashtags), which library seems more robust for handling "messy" real-world text? What specific advantages do you notice? How might this impact a real-world application like social media sentiment analysis?

Double-click this cell to write your answer:

More robust library: SpaCy

Specific advantages: Effectively manages emojis and hashtags with greater precision, maintains emoji tokens, and consistently differentiates symbols. Additionally, it offers part-of-speech tagging and lemmas for more comprehensive analysis.

Impact on sentiment analysis: The precise tokenization of emojis and hashtags by spaCy aids in capturing emotional nuances and current trends, resulting in more insightful and contextually relevant sentiment analysis in practical social media scenarios.

🛑 Part 4: Stop Words Removal

What are Stop Words?

Stop words are common words that appear frequently in a language but typically don't carry much meaningful information about the content. Examples include "the", "is", "at", "which", "on", etc.

Why Remove Stop Words?

1. **Reduce noise** in the data
2. **Improve efficiency** by reducing vocabulary size
3. **Focus on content words** that carry semantic meaning

When NOT to Remove Stop Words?

- **Sentiment analysis:** "not good" vs "good" - the "not" is crucial!
- **Question answering:** "What is the capital?" - "what" and "is" provide context

```
# Step 7: Explore Stop Words Lists
from nltk.corpus import stopwords

# Get NLTK English stop words
nltk_stopwords = set(stopwords.words('english'))
print(f"📊 NLTK has {len(nltk_stopwords)} English stop words")
print(f"First 20: {sorted(list(nltk_stopwords))[:20]}")

# Get spaCy stop words
spacy_stopwords = nlp.Defaults.stop_words
print(f"\n📊 spaCy has {len(spacy_stopwords)} English stop words"
print(f"First 20: {sorted(list(spacy_stopwords))[:20]}")

# Compare the lists
common_stopwords = nltk_stopwords.intersection(spacy_stopwords)
nltk_only = nltk_stopwords - spacy_stopwords
spacy_only = spacy_stopwords - nltk_stopwords

print(f"\n🔍 Comparison:")
print(f"Common stop words: {len(common_stopwords)}")
print(f"Only in NLTK: {len(nltk_only)} - Examples: {sorted(list(nl
print(f"Only in spaCy: {len(spacy_only)} - Examples: {sorted(list(
```

```
📊 NLTK has 198 English stop words
First 20: ['a', 'about', 'above', 'after', 'again', 'against',

📊 spaCy has 326 English stop words
First 20: ["'d", "'ll", "'m", "'re", "'s", "'ve", 'a', 'about'

🔍 Comparison:
Common stop words: 123
Only in NLTK: 75 - Examples: ['ain', 'aren', "aren't", 'couldr
Only in spaCy: 203 - Examples: ["'d", "'ll", "'m", "'re", "'s'
```

🤔 Conceptual Question 7

Why do you think NLTK and spaCy have different stop word lists? Look at the examples of words that are only in one list - do you agree with these choices? Can you think of scenarios where these differences might significantly impact your NLP results?

Double-click this cell to write your answer:

Reasons for differences: NLTK and spaCy adopt distinct criteria for identifying stop words. While spaCy encompasses a broader range of contractions and auxiliary forms relevant to contemporary language use, NLTK emphasizes traditional stop words that are prevalent in linguistic studies.

Agreement with choices: Indeed, spaCy's incorporation of contractions such as "'s" and "'ll" proves beneficial for informal and conversational text. Conversely, NLTK's inclusion of negations like "ain't" and "couldn't" might be less suitable for applications such as sentiment analysis, where these terms hold significant meaning.

Scenarios where differences matter: In the context of **sentiment analysis**, the removal of negations for instance, "not" or "couldn't" can alter the intended meaning. In **chatbot development** or **social media analysis**, it may be essential to retain contractions to enhance contextual comprehension. The selection of an appropriate stop word list is contingent upon the specific task and the nature of the text.

```
# Step 8: Remove Stop Words with NLTK
# Test on simple text
original_tokens = nltk_tokens  # From earlier tokenization
filtered_tokens = [word for word in original_tokens if word.lower(

print("🧪 NLTK Stop Word Removal")
print("=" * 40)
print(f"Original: {simple_text}")
print(f"\nOriginal tokens ({len(original_tokens)}): {original_toke
print(f"After removing stop words ({len(filtered_tokens)}): {filte

# Show which words were removed
removed_words = [word for word in original_tokens if word.lower()
print(f"\nRemoved words: {removed_words}")

# Calculate reduction percentage
reduction = (len(original_tokens) - len(filtered_tokens)) / len(or
print(f"Vocabulary reduction: {reduction:.1f}%")
```

```
🧪 NLTK Stop Word Removal
========================================
Original: Natural Language Processing is a fascinating field c

Original tokens (14): ['Natural', 'Language', 'Processing', 'i
After removing stop words (10): ['Natural', 'Language', 'Proce

Removed words: ['is', 'a', 'of', 'It']
Vocabulary reduction: 28.6%
```

```
# Step 9: Remove Stop Words with spaCy
doc = nlp(simple_text)
spacy_filtered = [token.text for token in doc if not token.is_stop

print("🧪 spaCy Stop Word Removal")
print("=" * 40)
print(f"Original: {simple_text}")
print(f"\nOriginal tokens ({len(spacy_tokens)}): {spacy_tokens}")
print(f"After removing stop words & punctuation ({len(spacy_filter

# Show which words were removed
spacy_removed = [token.text for token in doc if token.is_stop or t
print(f"\nRemoved words: {spacy_removed}")
```

```
# Calculate reduction percentage
spacy_reduction = (len(spacy_tokens) - len(spacy_filtered)) / len(
print(f"Vocabulary reduction: {spacy_reduction:.1f}%")
```

```
🧪 spaCy Stop Word Removal
========================================
Original: Natural Language Processing is a fascinating field c

Original tokens (14): ['Natural', 'Language', 'Processing', 'i
After removing stop words & punctuation (7): ['Natural', 'Lang

Removed words: ['is', 'a', 'of', '.', 'It', "'s", '!']
Vocabulary reduction: 50.0%
```

🤔 Conceptual Question 8

Compare the NLTK and spaCy stop word removal results. Which approach removed more words? Do you think removing punctuation (as spaCy did) is always a good idea? Give a specific example where keeping punctuation might be important for NLP analysis.

Double-click this cell to write your answer:

Which removed more: spaCy was more efficient in eliminating stop words and also took out punctuation, in contrast to NLTK

Punctuation removal assessment: The removal of punctuation is not always beneficial—it can eliminate significant indicators, particularly in tasks that involve tone or emotion.

Example where punctuation matters: In **sentiment analysis**, the phrases "Great!" and "Great." convey different levels of emotional intensity. The absence of the exclamation mark may diminish the accuracy of sentiment detection.

🌱 Part 5: Lemmatization and Stemming

What is Lemmatization?

Lemmatization reduces words to their base or dictionary form (called a **lemma**). It considers context and part of speech to ensure the result is a valid word.

What is Stemming?

Stemming reduces words to their root form by removing suffixes. It's faster but less accurate than lemmatization.

Key Differences:

Aspect	Stemming	Lemmatization
Speed	Fast	Slower
Accuracy	Lower	Higher
Output	May be non-words	Always valid words
Context	Ignores context	Considers context

Examples:

- **"running"** → Stem: "run", Lemma: "run"
- **"better"** → Stem: "better", Lemma: "good"
- **"was"** → Stem: "wa", Lemma: "be"

```
# Step 10: Stemming with NLTK
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

# Test words that demonstrate stemming challenges
test_words = ['running', 'runs', 'ran', 'better', 'good', 'best', '

print("🌿 Stemming Demonstration")
print("=" * 30)
print(f"{'Original':<12} {'Stemmed':<12}")
print("-" * 25)

for word in test_words:
    stemmed = stemmer.stem(word)
    print(f"{word:<12} {stemmed:<12}")

# Apply to our sample text
sample_tokens = [token for token in nltk_tokens if token.isalpha()]
stemmed_tokens = [stemmer.stem(token.lower()) for token in sample_t

print(f"\n🧪 Applied to sample text:")
print(f"Original: {sample_tokens}")
print(f"Stemmed: {stemmed_tokens}")
```

```
🌿 Stemming Demonstration
==============================
Original     Stemmed
-------------------------
running      run
runs         run
ran          ran
better       better
good         good
best         best
flying       fli
flies        fli
was          wa
were         were
cats         cat
dogs         dog

🧪 Applied to sample text:
Original: ['Natural', 'Language', 'Processing', 'is', 'a', 'fa
Stemmed: ['natur', 'languag', 'process', 'is', 'a', 'fascin',
```

🤔 Conceptual Question 9

Look at the stemming results above. Can you identify any cases where stemming produced questionable results? For example, how were "better" and "good" handled? Do you think this is problematic for NLP applications? Explain your reasoning.

Double-click this cell to write your answer:

Questionable results identified: Terms such as "better" and "good" may be incorrectly stemmed or not recognized as related due to their irregular forms.

Assessment of "better" and "good": Stemming may fail to reduce both terms to a common root, overlooking their shared meaning. For instance, "better" might remain unchanged while "good" is deemed unrelated.

Impact on NLP applications: This inconsistency can negatively affect tasks such as sentiment analysis or text classification, where semantically related words ought to be grouped together. Lemmatization is frequently more effective for maintaining meaning

---

```python
# Step 11: Lemmatization with spaCy
print("🌱 spaCy Lemmatization Demonstration")
print("=" * 40)

# Test on a complex sentence
complex_sentence = "The researchers were studying the effects of ru
doc = nlp(complex_sentence)

print(f"Original: {complex_sentence}")
print(f"\n{'Token':<15} {'Lemma':<15} {'POS':<10} {'Explanation':<2
print("-" * 65)

for token in doc:
    if token.is_alpha:
        explanation = "No change" if token.text.lower() == token.le
        print(f"{token.text:<15} {token.lemma_:<15} {token.pos_:<10

# Extract lemmas
lemmas = [token.lemma_.lower() for token in doc if token.is_alpha a
print(f"\n🔤 Lemmatized tokens (no stop words): {lemmas}")
```

```
🌱 spaCy Lemmatization Demonstration
========================================
Original: The researchers were studying the effects of running

Token           Lemma           POS        Explanation
-----------------------------------------------------------------
The             the             DET        No change
researchers     researcher      NOUN       Lemmatized
were            be              AUX        Lemmatized
studying        study           VERB       Lemmatized
```

```
the            the            DET        No change
effects        effect         NOUN       Lemmatized
of             of             ADP        No change
running        run            VERB       Lemmatized
and            and            CCONJ      No change
swimming       swim           VERB       Lemmatized
on             on             ADP        No change
better         well           ADJ        Lemmatized
performance    performance    NOUN       No change

🔤 Lemmatized tokens (no stop words): ['researcher', 'study',
```

```
# Step 12: Compare Stemming vs Lemmatization
comparison_words = ['better', 'running', 'studies', 'was', 'childre

print("⚖️ Stemming vs Lemmatization Comparison")
print("=" * 50)
print(f"{'Original':<12} {'Stemmed':<12} {'Lemmatized':<12}")
print("-" * 40)

for word in comparison_words:
    # Stemming
    stemmed = stemmer.stem(word)

    # Lemmatization with spaCy
    doc = nlp(word)
    lemmatized = doc[0].lemma_

    print(f"{word:<12} {stemmed:<12} {lemmatized:<12}")
```

```
⚖️ Stemming vs Lemmatization Comparison
==================================================
Original     Stemmed      Lemmatized
----------------------------------------
better       better       well
running      run          run
studies      studi        study
was          wa           be
children     children     child
feet         feet         foot
```

🤔 Conceptual Question 10

Compare the stemming and lemmatization results. Which approach do you think is more suitable for:

1. **A search engine** (where speed is crucial and you need to match variations of words)?
2. **A sentiment analysis system** (where accuracy and meaning preservation are important)?
3. **A real-time chatbot** (where both speed and accuracy matter)?

Explain your reasoning for each choice.

Double-click this cell to write your answer:

1. Search engine: Stemming: This method is quicker and sufficiently effective in matching various word forms (e.g., "running" → "run"), thereby enhancing recall in search outcomes.

2. Sentiment analysis: Lemmatization: This approach maintains the original meaning and manages irregular forms more effectively (example, "better" → "well"), resulting in a more precise interpretation of sentiment.

3. Real-time chatbot: Lemmatization (recommended, if optimized): It provides superior language comprehension, which is essential for generating natural responses. In scenarios where speed is of the essence and resources are constrained, light stemming may be employed; however, lemmatization remains the optimal choice for ensuring clarity and contextual relevance.

🧹 Part 6: Text Cleaning and Normalization

What is Text Cleaning?

Text cleaning involves removing or standardizing elements that might interfere with analysis:

- **Case normalization** (converting to lowercase)
- **Punctuation removal**
- **Number handling** (remove, replace, or normalize)
- **Special character handling** (URLs, emails, mentions)
- **Whitespace normalization**

Why is it Important?

- Ensures consistency across your dataset
- Reduces vocabulary size
- Improves model performance
- Handles edge cases in real-world data

```
# Step 13: Basic Text Cleaning
def basic_clean_text(text):
    """Apply basic text cleaning operations"""
    # Convert to lowercase
    text = text.lower()

    # Remove extra whitespace
    text = re.sub(r'\s+', ' ', text).strip()

    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuatior

    # Remove numbers
    text = re.sub(r'\d+', '', text)
```

```
    # Remove extra spaces again
    text = re.sub(r'\s+', ' ', text).strip()

    return text

# Test basic cleaning
test_text = "   Hello WORLD!!! This has 123 numbers and   extra sp
cleaned = basic_clean_text(test_text)

print("🧹 Basic Text Cleaning")
print("=" * 30)
print(f"Original: '{test_text}'")
print(f"Cleaned: '{cleaned}'")
print(f"Length reduction: {(len(test_text) - len(cleaned))/len(tes
```

```
🧹 Basic Text Cleaning
==============================
Original: '   Hello WORLD!!! This has 123 numbers and   extra
Cleaned: 'hello world this has numbers and extra spaces'
Length reduction: 26.2%
```

```
# Step 14: Advanced Cleaning for Social Media
def advanced_clean_text(text):
    """Apply advanced cleaning for social media and web text"""
    # Remove URLs
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.M

    # Remove email addresses
    text = re.sub(r'\S+@\S+', '', text)

    # Remove mentions (@username)
    text = re.sub(r'@\w+', '', text)

    # Convert hashtags (keep the word, remove #)
    text = re.sub(r'#(\w+)', r'\1', text)

    # Remove emojis (basic approach)
    emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticc
                           u"\U0001F300-\U0001F5FF"  # symbols
                           u"\U0001F680-\U0001F6FF"  # transpc
                           u"\U0001F1E0-\U0001F1FF"  # flags
                           "]+", flags=re.UNICODE)
    text = emoji_pattern.sub(r'', text)

    # Convert to lowercase and normalize whitespace
    text = text.lower()
    text = re.sub(r'\s+', ' ', text).strip()

    return text

# Test on social media text
print("🚀 Advanced Cleaning on Social Media Text")
print("=" * 45)
print(f"Original: {social_text}")

cleaned_social = advanced_clean_text(social_text)
```

```
print(f"Cleaned: {cleaned_social}")
print(f"Length reduction: {(len(social_text) - len(cleaned_social)
```

```
🚀 Advanced Cleaning on Social Media Text
=============================================
Original: OMG! Just tried the new coffee shop ☕ SO GOOD!!! H
Cleaned: omg! just tried the new coffee shop ☕ so good!!! hi
Length reduction: 7.2%
```

🤔 Conceptual Question 11

Look at the advanced cleaning results for the social media text. What information was lost during cleaning? Can you think of scenarios where removing emojis and hashtags might actually hurt your NLP application? What about scenarios where keeping them would be beneficial?

Double-click this cell to write your answer:

Information lost: Emojis such as " 👍 " and "😍 ", along with hashtags (for instance, "#coffee", "#yum"), have forfeited their emotional and topical cues.

Scenarios where removal hurts: In the context of "sentiment analysis", the elimination of emojis and hashtags can diminish the detection of emotional tone and impair the model's capacity to comprehend user intent or enthusiasm.

Scenarios where keeping helps: In areas like "trend detection" or "brand monitoring", hashtags play a crucial role in identifying topics, while emojis express user sentiment both of which are valuable for monitoring public responses or gathering product feedback.

🔧 Part 7: Building a Complete Preprocessing Pipeline

Now let's combine everything into a comprehensive preprocessing pipeline that you can customize based on your needs.

Pipeline Components:

1. **Text cleaning** (basic or advanced)
2. **Tokenization** (NLTK or spaCy)
3. **Stop word removal** (optional)
4. **Lemmatization/Stemming** (optional)
5. **Additional filtering** (length, etc.)

```
# Step 15: Complete Preprocessing Pipeline
def preprocess_text(text,
                   clean_level='basic',     # 'basic' or 'advancec
                   remove_stopwords=True,
                   use_lemmatization=True,
                   use_stemming=False,
                   min_length=2):
    """
    Complete text preprocessing pipeline
    """
    # Step 1: Clean text
    if clean_level == 'basic':
        cleaned_text = basic_clean_text(text)
    else:
        cleaned_text = advanced_clean_text(text)

    # Step 2: Tokenize
    if use_lemmatization:
        # Use spaCy for lemmatization
        doc = nlp(cleaned_text)
        tokens = [token.lemma_.lower() for token in doc if token.i
    else:
        # Use NLTK for basic tokenization
        tokens = word_tokenize(cleaned_text)
        tokens = [token for token in tokens if token.isalpha()]

    # Step 3: Remove stop words
    if remove_stopwords:
        if use_lemmatization:
            tokens = [token for token in tokens if token not in sp
        else:
            tokens = [token.lower() for token in tokens if token.l

    # Step 4: Apply stemming if requested
    if use_stemming and not use_lemmatization:
        tokens = [stemmer.stem(token.lower()) for token in tokens]

    # Step 5: Filter by length
    tokens = [token for token in tokens if len(token) >= min_lengt

    return tokens

print("🔧 Preprocessing Pipeline Created!")
print("✅ Ready to test different configurations.")
```

```
🔧 Preprocessing Pipeline Created!
✅ Ready to test different configurations.
```

```
# Step 16: Test Different Pipeline Configurations
test_text = sample_texts["Product Review"]
print(f"🎯 Testing on: {test_text[:100]}...")
print("=" * 60)

# Configuration 1: Minimal processing
minimal = preprocess_text(test_text,
                         clean_level='basic',
                         remove_stopwords=False,
                         use_lemmatization=False,
                         use_stemming=False)
```

```
print(f"\n1. Minimal processing ({len(minimal)} tokens):")
print(f"   {minimal[:10]}...")

# Configuration 2: Standard processing
standard = preprocess_text(test_text,
                          clean_level='basic',
                          remove_stopwords=True,
                          use_lemmatization=True)
print(f"\n2. Standard processing ({len(standard)} tokens):")
print(f"   {standard[:10]}...")

# Configuration 3: Aggressive processing
aggressive = preprocess_text(test_text,
                            clean_level='advanced',
                            remove_stopwords=True,
                            use_lemmatization=False,
                            use_stemming=True,
                            min_length=3)
print(f"\n3. Aggressive processing ({len(aggressive)} tokens):")
print(f"   {aggressive[:10]}...")

# Show reduction percentages
original_count = len(word_tokenize(test_text))
print(f"\n📊 Token Reduction Summary:")
print(f"   Original: {original_count} tokens")
print(f"   Minimal: {len(minimal)} ({(original_count-len(minimal))/
print(f"   Standard: {len(standard)} ({(original_count-len(standard
print(f"   Aggressive: {len(aggressive)} ({(original_count-len(aggr
```

```
🎯 Testing on: This laptop is absolutely fantastic! I've been
The ...
============================================================

1. Minimal processing (34 tokens):
   ['this', 'laptop', 'is', 'absolutely', 'fantastic', 'ive',

2. Standard processing (18 tokens):
   ['laptop', 'absolutely', 'fantastic', 've', 'use', 'month',

3. Aggressive processing (21 tokens):
   ['laptop', 'absolut', 'fantast', 'use', 'month', 'still', '

📊 Token Reduction Summary:
   Original: 47 tokens
   Minimal: 34 (27.7% reduction)
   Standard: 18 (61.7% reduction)
   Aggressive: 21 (55.3% reduction)
```

🤔 Conceptual Question 12

Compare the three pipeline configurations (Minimal, Standard, Aggressive). For each configuration, analyze:

1. **What information was preserved?**
2. **What information was lost?**
3. **What type of NLP task would this configuration be best suited for?**

Double-click this cell to write your answer:

Minimal Processing:

- Preserved: The majority of the original text, including contractions and modifiers example,"I've," "absolutely"
- Lost: Very few stopwords; minimal cleaning involved
- Best for: "Sentiment analysis" or "review interpretation", where emotional tone and nuance hold significance

Standard Processing:

- Preserved: Essential content words and cleaned lemmas example, "use," "fast," "battery"
- Lost: Stopwords, function words, and contractions
- Best for: "Topic modeling" or "keyword extraction", where the emphasis is on fundamental ideas

Aggressive Processing:

- Preserved: Only root forms example, "fantast," "batteri"
- Lost: Original word forms, modifiers, and emotional intensity
- Best for: Search indexing or document clustering, where rapid grouping of word variations is necessary

---

```
# Step 17: Comprehensive Analysis Across Text Types
print("🔬 Comprehensive Preprocessing Analysis")
print("=" * 50)

# Test standard preprocessing on all text types
results = {}
for name, text in sample_texts.items():
    original_tokens = len(word_tokenize(text))
    processed_tokens = preprocess_text(text,
                                      clean_level='basic',
                                      remove_stopwords=True,
                                      use_lemmatization=True)

    reduction = (original_tokens - len(processed_tokens)) / origir
    results[name] = {
        'original': original_tokens,
        'processed': len(processed_tokens),
        'reduction': reduction,
        'sample': processed_tokens[:8]
    }

    print(f"\n📄 {name}:")
    print(f"   Original: {original_tokens} tokens")
    print(f"   Processed: {len(processed_tokens)} tokens ({reducti
    print(f"   Sample: {processed_tokens[:8]}")

# Summary table
print(f"\n\n📋 Summary Table")
print(f"{'Text Type':<15} {'Original':<10} {'Processed':<10} {'Rec
print("-" * 50)
```

```
for name, data in results.items():
    print(f"{name:<15} {data['original']:<10} {data['processed']:<
```

```
🔬 Comprehensive Preprocessing Analysis
==================================================

📄 Simple:
  Original: 14 tokens
  Processed: 7 tokens (50.0% reduction)
  Sample: ['natural', 'language', 'processing', 'fascinating'

📄 Academic:
  Original: 61 tokens
  Processed: 26 tokens (57.4% reduction)
  Sample: ['dr', 'smith', 'research', 'machinelearning', 'alg

📄 Social Media:
  Original: 22 tokens
  Processed: 10 tokens (54.5% reduction)
  Sample: ['omg', 'try', 'new', 'coffee', 'shop', 'good', 'hi

📄 News:
  Original: 51 tokens
  Processed: 25 tokens (51.0% reduction)
  Sample: ['stock', 'market', 'experience', 'significant', 'v

📄 Product Review:
  Original: 47 tokens
  Processed: 18 tokens (61.7% reduction)
  Sample: ['laptop', 'absolutely', 'fantastic', 've', 'use',


📋 Summary Table
Text Type       Original   Processed  Reduction
--------------------------------------------------
Simple          14         7          50.0      %
Academic        61         26         57.4      %
Social Media    22         10         54.5      %
News            51         25         51.0      %
Product Review  47         18         61.7      %
```

🤔 Final Conceptual Question 13

Looking at the comprehensive analysis results across all text types:

1. **Which text type was most affected by preprocessing?** Why do you think this happened?
2. **Which text type was least affected?** What does this tell you about the nature of that text?
3. **If you were building an NLP system to analyze customer reviews for a business, which preprocessing approach would you choose and why?**
4. **What are the main trade-offs you need to consider when choosing preprocessing techniques for any NLP project?**

Double-click this cell to write your answer:

---

1. Most affected text type: Product Review: This category saw the greatest decrease (61.7%) due to the elimination of informal language, superfluous words, and numerous modifiers.

2. Least affected text type: Simple Text: With merely 14 original tokens, there is less content to eliminate. This indicates that it was already clear, straightforward, and devoid of unnecessary elements.

3. For customer review analysis: Employ "standard preprocessing" techniques to eliminate noise while preserving essential sentiment words, modifiers, and certain punctuation marks to maintain the emotional tone.

4. Main trade-offs to consider: Accuracy versus Speed: A more rigorous cleaning process is quicker but may jeopardize the preservation of meaning. Context versus Simplicity: Excessive removal can diminish nuance. Task Relevance: Preprocessing should align with the objectives, such as sentiment analysis versus topic extraction.

🎯 Lab Summary and Reflection

Congratulations! You've completed a comprehensive exploration of NLP preprocessing techniques.

🔑 Key Concepts You've Mastered:

1. **Text Preprocessing Fundamentals** - Understanding why preprocessing is crucial
2. **Tokenization Techniques** - NLTK vs spaCy approaches and their trade-offs
3. **Stop Word Management** - When to remove them and when to keep them
4. **Morphological Processing** - Stemming vs lemmatization for different use cases
5. **Text Cleaning Strategies** - Basic vs advanced cleaning for different text types
6. **Pipeline Design** - Building modular, configurable preprocessing systems

🎓 Real-World Applications:

These techniques form the foundation for search engines, chatbots, sentiment analysis, document classification, machine translation, and information extraction systems.

💡 Key Insights to Remember:

- **No Universal Solution**: Different NLP tasks require different preprocessing approaches
- **Trade-offs Are Everywhere**: Balance information preservation with noise reduction
- **Context Matters**: The same technique can help or hurt depending on your use case
- **Experimentation Is Key**: Always test and measure impact on your specific task

---

Excellent work completing Lab 02! 🎉

For your reflection journal, focus on the insights you gained about when and why to use different techniques, the challenges you encountered, and connections you made to real-world applications.