Lab 02: Basic NLP Preprocessing Techniques

Course: ITAI 2373 - Natural Language Processing
Module: 02 - Text Preprocessing
Duration: 2-3 hours
Student Name: Codie Munos__________
Date: 06/08/2025__________

🎯 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.

🤔 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: Translation apps do not always guve the best meaning for certain words. For example, some English words do not translate to the meaning intended when trying to get a Spanish translation. Translation apps focus more on proper dialect while most human conversations do not follow the same rules. This causes issue when trying to have a real life conversation witht the help of translation.

Challenge 2: Search engines have the ability to process human language, but sometimes does not understand the context of what is being typed. They require a bit more direction on what specifically is being asked. For example, googling certain locations for food will pop up information from all over, so it promts to share location in order to get more specific information. This can create issues when location cannot be shared.

Challenge 3: Chatbots summarize a lot of sources to come up with a few answers. It is usefull for needing to filter through many sources with short descriptions of what they offer. Though there is a lot of good use for it, unfortunately it does not always give the exact answer that you are looking for. I believe it has so many different options to give as a reply, therefore you have to ask a few follow up questions to get a deeper answer.

🛠️ Part 1: Environment Setup

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/releases/download/en_core_web_sm
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 77.0 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.
✅ 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.

I think we need to install the separate language because SpaCy does not fully comprehend human language on its own. It is more of the foundation for the language model to build upon. The model is able to breakdown context, definitions, and form a relationship between words which helps with text processing. It can do this by the data packages being downloaded that include tokenization, stop word removal, lemmatization, and POS tagging.

```
# 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-to-
[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 of AI. It's amazing!"

academic_text = """
Dr. Smith's research on machine-learning algorithms is groundbreaking!
She published 3 papers in 2023, focusing on deep neural networks (DNNs).
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!!! Highly recommend 👍 #coffe

news_text = """
The stock market experienced significant volatility today, with tech stocks leading the decl
Apple Inc. (AAPL) dropped 3.2%, while Microsoft Corp. fell 2.8%.
"We're seeing a rotation out of growth stocks," said analyst Jane Doe from XYZ Capital.
"""

review_text = """
This laptop is absolutely fantastic! I've been using it for 6 months and it's still super fa
The battery life is incredible - lasts 8-10 hours easily.
Only complaint: the keyboard could be better. Overall rating: 4.5/5 stars.
"""

# 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 of AI. It's amazing!

🏷️ Academic: Dr. Smith's research on machine-learning algorithms is groundbreaking!
She publi...

🏷️ Social Media: OMG! Just tried the new coffee shop ☕ SO GOOD!!! Highly recommend 👍

🏷️ News: The stock market experienced significant volatility today, with tech stocks le

🏷️ Product Review: This laptop is absolutely fantastic! I've been using it for 6 months
```

🤔 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. I antitipate that there will be preprocessing errors such as tokenization complications. While using two or more words to describe one idea, the interpretation can be wrong due to the simple text not giving more context.
2. Another challenge I see happening involve words needing to be captitalized in certain text, while in others needing to be left lowercase. An example being "sergeant" being used as a common noun and remaining lowercased. While if "Sergeant" is followed with a name, then it becomes a capitalized word. This can become problematic whenver there are many rules in the human language to follow.

Academic text challenges:

1. Academic text usually has more advanced language used. The format is different along with the content included. Even in different areas of study, the format may be similar along with the words, but the context can be regarding an entirely different subject. This is problematic for NLP analysis as it may need more direction to get the results desired.
2. These text also include citations which can become tricky whenever you are tyoing a paper and it autocorrects to something that seems more gramatically correct. This makes the

attention to detail important whenever writing any type of paper on a word processor.

Social media text challenges:

1. There are many forms of communication on social media, including hashtags, emojis, slang, videos and more. The example above in "Load Sample Text" shows "OMG" which makes the text processing unreliable. Along with the continuous use of capital letters in the sentence which can also disrupt text processing abilities.
2. The example above also uses hashtags and emojis which are not included in the traditional text category. It can be problematic whenever te model is trying to process it as something important or ignorable.

News text challenges:

1. News text can pose challenges whenever the news topic becomes worldwide knowledge. There can be many different articles in different languages which makes reporting on certain topics a bit harder. It can be an article being translated via an online translator and the model may not give the correct translation. This can lead to misinformation being reported on worldwide.
2. When searching for specific news articles, it can be hard to find the right one due to misleading titles. The use of the titles puts that website into a generalized category of what seems similar. It is hard for the models to filter out past the plain text due to it not being aware of the click bait mentality humans have.

Product review challenges:

1. Reviews online are sketchy due to the inability to always differentiate a review completed by a human versus one that was made by bots. The lack of ability to filter them out is due to NLP not seeing the two as different. This can lead to a problematic experience for customers purchasing based on written online reviews.

2. There is a struggle for NLP to recognize words that have been created over the past few years in the slang department. Sometimes to relate to more people on social media, there is slang used in reviews to make a connection to another reader. It shows authenticity to humans while still making it difficult for the NLP model to understand.

🔤 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?

```
# 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)}")
```

```
🔍 NLTK Tokenization Results
========================================
Original: Natural Language Processing is a fascinating field of AI. It's amazing!
---------------------------------------------------------------------------
LookupError                               Traceback (most recent call last)
<ipython-input-5-4218255646> in <cell line: 0>()
      8
      9 # Word tokenization
---> 10 nltk_tokens = word_tokenize(simple_text)
     11 print(f"\nWord tokens: {nltk_tokens}")
     12 print(f"Number of tokens: {len(nltk_tokens)}")

                              5 frames
/usr/local/lib/python3.11/dist-packages/nltk/data.py in find(resource_name, paths)
    577     sep = "*" * 70
    578     resource_not_found = f"\n{sep}\n{msg}\n{sep}\n"
--> 579     raise LookupError(resource_not_found)
    580
    581

LookupError:
**********************************************************************
  Resource punkt_tab not found.
  Please use the NLTK Downloader to obtain the resource:

  >>> import nltk
  >>> nltk.download('punkt_tab')

  For more information see: https://www.nltk.org/data.html

  Attempted to load tokenizers/punkt_tab/english/

  Searched in:
    - '/root/nltk_data'
    - '/usr/nltk_data'
    - '/usr/share/nltk_data'
    - '/usr/lib/nltk_data'
    - '/usr/share/nltk_data'
    - '/usr/local/share/nltk_data'
    - '/usr/lib/nltk_data'
    - '/usr/local/lib/nltk_data'
**********************************************************************
```

🤔 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: "it's" was not handled well due to the "punkt" library not being introduced.

Punctuation treatment: It split the punctuation into different tokens.

Appropriateness for different tasks: Depending on the task, the right approach varies. For speech to text situations, it might be best for the tokenization to include interpretaion as one such as "it's" rather in sentiment analysis, it is better to keep them separate such as "it", "'s".

```
# 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 Stop':<8}")
print("-" * 50)
for token in doc:
    print(f"{token.text:<12} {token.pos_:<8} {token.lemma_:<12} {token.is_alpha:<8} {token.i
```

```
🔍 spaCy Tokenization Results
========================================
Original: Natural Language Processing is a fascinating field of AI. It's amazing!

Word tokens: ['Natural', 'Language', 'Processing', 'is', 'a', 'fascinating', 'field', 'c
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: SpaCy calls upon the pretrained NLP library which helps with processing. It also keeps "it's" as one token after being executed. It also appears to utilize lemmatization rather than the stemming approach that NLTK uses. I also see that the punctuation is not separated into individual tokens in SpaCy.

Better for sentiment analysis: SpaCy is better for sentiment analysis due to keeping tokens together.

Better for information extraction: SpaCy is also better for information extraction.

Overall assessment: NLTK is good for certain uses that require less gramatical knowledge. SpaCy is better at keeping the tokens together that need to remain intact to deliver the message in the intended way.

```python
# 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!!! Highly recommend 👍 #coffee
---------------------------------------------------------------------------
LookupError                               Traceback (most recent call last)
<ipython-input-7-3786859953> in <cell line: 0>()
      5 
      6 # NLTK approach
----> 7 social_nltk_tokens = word_tokenize(social_text)
      8 print(f"\nNLTK tokens: {social_nltk_tokens}")
      9 

                              5 frames
/usr/local/lib/python3.11/dist-packages/nltk/data.py in find(resource_name, paths)
    577     sep = "*" * 70
    578     resource_not_found = f"\n{sep}\n{msg}\n{sep}\n"
--> 579     raise LookupError(resource_not_found)
    580 
    581 

LookupError: 
**********************************************************************
  Resource punkt_tab not found.
  Please use the NLTK Downloader to obtain the resource:

  >>> import nltk
  >>> nltk.download('punkt_tab')
  
  For more information see: https://www.nltk.org/data.html

  Attempted to load tokenizers/punkt_tab/english/

  Searched in:
    - '/root/nltk_data'
    - '/usr/nltk_data'
    - '/usr/share/nltk_data'
    - '/usr/lib/nltk_data'
    - '/usr/share/nltk_data'
    - '/usr/local/share/nltk_data'
    - '/usr/lib/nltk_data'
    - '/usr/local/lib/nltk_data'
**********************************************************************
```

🤔 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 is better off handling "messy" text.

Specific advantages: The advantages include SpaCy as keeping hashtags intact with what follows rather than separating them the way that NTLK does. Another advantage is the use of the pre trained model that SpaCy uses.

Impact on sentiment analysis: The impact is that SpaCy delivers better outputs by having the deeper connection to the meaning of what is being said. It makes the experience feel more natural and allows for the emotions to stay with the words rather than becomming chopped into different parts of a sentence.

🛑 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(nltk_only))[:5]}")
print(f"Only in spaCy: {len(spacy_only)} - Examples: {sorted(list(spacy_only))[:5]}")
```

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

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

🔍 Comparison:
Common stop words: 123
Only in NLTK: 75 - Examples: ['ain', 'aren', "aren't", 'couldn', "couldn't"]
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: The different amount of stop words stem from SpaCy and NTLK having different goals. The SpaCy goal is to be more modernized and able to keep up with societies' current interpretations.

Agreement with choices: The choices for the most part are the same and I agree that they are just fluff words that will not impact too many scenarious.

Scenarios where differences matter:

I can think of two words on here that can cause issues, those include "any" and "because". If these words are not plugged into the understanding, it can throw the whole sentence or conversation off. "Because" is a bridge word which starts off by explaining the result and ends by connecting via using "because" to jump to the how.

```
# 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() not in nltk_stopwords]

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

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

# Calculate reduction percentage
reduction = (len(original_tokens) - len(filtered_tokens)) / len(original_tokens) * 100
print(f"Vocabulary reduction: {reduction:.1f}%")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-1459279799> in <cell line: 0>()
      1 # Step 8: Remove Stop Words with NLTK
      2 # Test on simple text
----> 3 original_tokens = nltk_tokens  # From earlier tokenization
      4 filtered_tokens = [word for word in original_tokens if word.lower() not in
nltk_stopwords]
      5

NameError: name 'nltk_tokens' is not defined
```

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

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_filtered)}): {spacy_filtered}")

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

# Calculate reduction percentage
spacy_reduction = (len(spacy_tokens) - len(spacy_filtered)) / len(spacy_tokens) * 100
print(f"Vocabulary reduction: {spacy_reduction:.1f}%")
```

```
🧪 spaCy Stop Word Removal
========================================
Original: Natural Language Processing is a fascinating field of AI. It's amazing!

Original tokens (14): ['Natural', 'Language', 'Processing', 'is', 'a', 'fascinating', 'f
After removing stop words & punctuation (7): ['Natural', 'Language', 'Processing', 'fasc

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: NTLK removed more than SpaCy did.

Punctuation removal assessment: I do not think removing punctuation all the time is a good idea. There is a lot of emotion in one character especially when trying to express tone over message.

Example where punctuation matters: In contracts, there are a lot of formal sentences. The use of punction in these scenarios is crucial for formality.

🌱 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

Aspect	Stemming	Lemmatization
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', 'flying', 'flies', 'was',

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_tokens]

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
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-3960853977> in <cell line: 0>()
     17
     18 # Apply to our sample text
---> 19 sample_tokens = [token for token in nltk_tokens if token.isalpha()]
     20 stemmed_tokens = [stemmer.stem(token.lower()) for token in sample_tokens]
     21

NameError: name 'nltk_tokens' is not defined
```

🤔 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: A few questionalble results included "fli", "better", and "was". "fli and "was" are nonword outputs and better should have resulted in "good". **Assessment of "better" and "good":** "better" should have gave the result of "good" because they are similar in context. **Impact on NLP applications:**

With words being cut off and others not being linked to other definition, this can cause errors in NLP applications. The translation can become inconsistent and search results can become altered by not grasping the root of different words being used that are meant for the same results.

```
# 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 running and swimming on bet
doc = nlp(complex_sentence)

print(f"Original: {complex_sentence}")
print(f"\n{'Token':<15} {'Lemma':<15} {'POS':<10} {'Explanation':<20}")
print("-" * 65)

for token in doc:
    if token.is_alpha:
        explanation = "No change" if token.text.lower() == token.lemma_ else "Lemmatized"
        print(f"{token.text:<15} {token.lemma_:<15} {token.pos_:<10} {explanation:<20}")

# Extract lemmas
lemmas = [token.lemma_.lower() for token in doc if token.is_alpha and not token.is_stop]
print(f"\n🔤 Lemmatized tokens (no stop words): {lemmas}")
```

```
🌱 spaCy Lemmatization Demonstration
========================================
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-883759909> in <cell line: 0>()
      5 # Test on a complex sentence
      6 complex_sentence = "The researchers were studying the effects of running and
swimming on better performance."
----> 7 doc = nlp(complex_sentence)
      8 
      9 print(f"Original: {complex_sentence}")

NameError: name 'nlp' is not defined
```

Next steps: Explain error

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

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
----------------------------------------
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-528577304> in <cell line: 0>()
      9 for word in comparison_words:
     10     # Stemming
---> 11     stemmed = stemmer.stem(word)
     12
     13     # Lemmatization with spaCy

NameError: name 'stemmer' is not defined
```

Next steps: Explain error

🤔 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)? Stemming approach is better.
2. **A sentiment analysis system** (where accuracy and meaning preservation are important)? Lemmatization is the better option.
3. **A real-time chatbot** (where both speed and accuracy matter)? Lemmatization is better for chat bots.

Explain your reasoning for each choice.

Double-click this cell to write your answer:

1. Search engine: I think that stemming would work better due to it searching a larger range of sites that are similar but not exact to the word being used. **2. Sentiment analysis:** Lemmatization is more consistent and accurate while stemming can be unreliable for definitions.

3. Real-time chatbot:

Lemmatization is not as fast as stemming, but it does focus more on word relationships which can lead to a better answer. It depends as well as what the chat bot is being used for. Lemmatization works better for customer sevice situations and other similar duties.

🧹 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.punctuation))

    # 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 spaces.   "
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(test_text)*100:.1f}%")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-643039145> in <cell line: 0>()
     21 # Test basic cleaning
     22 test_text = "   Hello WORLD!!! This has 123 numbers and   extra spaces.   "
---> 23 cleaned = basic_clean_text(test_text)
     24 
     25 print("🧹 Basic Text Cleaning")

<ipython-input-1-643039145> in basic_clean_text(text)
      6 
      7     # Remove extra whitespace
----> 8     text = re.sub(r'\s+', ' ', text).strip()
      9 
     10     # Remove punctuation

NameError: name 're' is not defined
```

Next steps: Explain error

```
# 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.MULTILINE)

    # 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"  # emoticons
                               u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                               u"\U0001F680-\U0001F6FF"  # transport & map symbols
                               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))/len(social_text)*100:.1f
```

```
🚀 Advanced Cleaning on Social Media Text
=============================================
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-369960869> in <cell line: 0>()
     32 print("🚀 Advanced Cleaning on Social Media Text")
     33 print("=" * 45)
---> 34 print(f"Original: {social_text}")
     35 
     36 cleaned_social = advanced_clean_text(social_text)

NameError: name 'social_text' is not defined
```

Next steps: Explain error

🤔 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: The first example does not remove emojis, while the second one does and it is the same for the hashtags.

Scenarios where removal hurts: By removing emojis, you lose sentiniment analysis. Sometimes the words being said are not actually meant to be understood at surface level depending on how it is used. Sarcasm can be taken differently for instance and if being used with emojis, can set it apart from a different type of attitude in the text. With the removal of hashtags, and keeping of the text that follows, it can cause the sentence structure to not flow. The use of hashtags enables networking for the same type of information being shared along with promoting oneself in a certain community.

Scenarios where keeping helps:

Keeping hashtags and emojis help with self-expression, cross-communication with others around the world, and brand promotion. Not everyone speaks the same written language, but emojis can be used to communicate a lot when there is lack of translation.

🔧 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 'advanced'
                   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.is_alpha]
    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 spacy_stopwords]
        else:
            tokens = [token.lower() for token in tokens if token.lower() not in nltk_stopwor

    # 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_length]

    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))/original_count*100:.1f}%
print(f"   Standard: {len(standard)} ({(original_count-len(standard))/original_count*100:.1f
print(f"   Aggressive: {len(aggressive)} ({(original_count-len(aggressive))/original_count*1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-2552558083> in <cell line: 0>()
      1 # Step 16: Test Different Pipeline Configurations
----> 2 test_text = sample_texts["Product Review"]
      3 print(f"🎯 Testing on: {test_text[:100]}...")
      4 print("=" * 60)
      5 

NameError: name 'sample_texts' is not defined
```

Next steps: Explain error

🤔 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: Stop worrds and raw word forms were preserved.
- Lost: There was no cleaning used which can cause issues with noise. Noise includes typos, irrelevant words/letters, and will not be able to use sentiment analysis.
- Best for: This is best for text summarization.

Standard Processing:

- Preserved: Lemmatization was used which means the definitions/context of words being used will form relationships and remain preserved.
- Lost: It will not keep stop words which can filter out words that are needed to build relationships between sentences and clauses.
- Best for: Standard processing is best used for text classification and sentiment analysis.

Aggressive Processing:

- Preserved: Agressive processing uses stemming and adcanced cleaning. This allows there to be less stop words, and core content left are stemming.
- Lost: If the stemming is too much, it can redefine the meaning of words and cause bad sentence flow.
- Best for: This is best used when using search engines suchh as google or bing.

```python
# 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)) / original_tokens * 100
    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 ({reduction:.1f}% reduction)")
    print(f"   Sample: {processed_tokens[:8]}")

# Summary table
print(f"\n\n📋 Summary Table")
print(f"{'Text Type':<15} {'Original':<10} {'Processed':<10} {'Reduction':<10}")
print("-" * 50)
for name, data in results.items():
    print(f"{name:<15} {data['original']:<10} {data['processed']:<10} {data['reduction']:<1
```

```
🔬 Comprehensive Preprocessing Analysis
==================================================
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-1232917379> in <cell line: 0>()
      5 # Test standard preprocessing on all text types
      6 results = {}
----> 7 for name, text in sample_texts.items():
      8     original_tokens = len(word_tokenize(text))
      9     processed_tokens = preprocess_text(text,

NameError: name 'sample_texts' is not defined
```

Next steps: Explain error

🤔 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: Informal text types are the most affected by preprocessing. This is due to the high usage of slang, run on sentences, emojis, hashtags, and trendy phrases.

2. Least affected text type: Least affected types would be formal text types. These text tend to have less noise and stop words in the writings. This allows for low usages of reduction.

3. For customer review analysis: For the use of NLP in customer servce reviews, lemmatization is better. This is due to the ability to retain the meaning of what is being said. It also allows room for stop words that are crucial to understanding to be kept. More sentiment analysis creates a better customer service experience.

4. Main trade-offs to consider: For NLP projects, there is a lot of consideration needed for preprocessing techniques. The risks at hand include choose between speed or accuracy, sentiment analysis, over-cleaning, and context of the field the project is being used for. All of these factors need to be considered while deciding what to use for each situation.

🎯 Lab Summary and Reflection

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

💡 Key Concepts You've Mastered: