NLP Self-Learning Assignment

Topics: Lemmatization vs Stemming & Unicode Normalization  
This document covers:  
- 📍 Lemmatization vs Stemming: Concepts and Python code  
- 📍 Unicode Normalization: Handling special characters and emojis

# 1. Lemmatization vs Stemming

Both are text preprocessing techniques in NLP used to reduce words to their base or root form.  
  
🔍 Why we do this?  
When analyzing text, words like 'study', 'studies', 'studying', and 'studied' all represent the same concept. To make text analysis easier and more consistent, we reduce them to a common root.

## Stemming

- Definition: Stemming chops off word endings to find the root form.  
- Pros: Fast and simple.  
- Cons: May produce non-dictionary words.  
📍 Example: "studies" → "studi"

from nltk.stem import PorterStemmer  
  
stemmer = PorterStemmer()  
words = ["studies", "studying", "studied", "happier", "flying"]  
for word in words:  
 print(f"{word} --> {stemmer.stem(word)}")

## Lemmatization

- Definition: Lemmatization uses vocabulary and grammar to find the base or dictionary word (lemma).  
- Pros: Produces real words.  
- Cons: Slightly slower.  
📍 Example: "studies" → "study"

import nltk

from nltk.stem import WordNetLemmatizer

from nltk.corpus import wordnet

# Download required resources (only once)

nltk.download('wordnet')

nltk.download('omw-1.4')

lemmatizer = WordNetLemmatizer()

words = ["studies", "studying", "studied", "happier", "flying"]

for word in words:

print(f"{word} --> {lemmatizer.lemmatize(word, pos='v')}")

## 📊 Comparison Table

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | Stemming | Lemmatization | Example |
| Output | May not be real word | Always a real word | studies → study |
| Speed | Faster | Slower |  |

# Unicode Normalization

When dealing with text from different languages or websites, the same-looking character might have different underlying representations.  
  
📍 Example:  
- 'café' (single composed character é)  
- 'cafe\u0301' (e + accent)  
  
They look the same but are treated as different strings.

s1 = "café"   
s2 = "cafe\u0301"   
  
print(s1 == s2) # False

**normalising**import unicodedata  
  
s1\_norm = unicodedata.normalize("NFC", s1)  
s2\_norm = unicodedata.normalize("NFC", s2)  
  
print(s1\_norm == s2\_norm) # True

## Real world uses:

cleaning user inputs(emojis,special characters)

comparing texts from different languages

preparing data for NLP models

## Unicode Normalization Forms

- \*\*NFC:\*\* Canonical Composition – most common for storage  
- \*\*NFD:\*\* Canonical Decomposition – useful for text analysis  
- \*\*NFKC:\*\* Compatibility Composition – ignores formatting differences  
- \*\*NFKD:\*\* Compatibility Decomposition – for strict comparisons

# Summary

- \*\*Stemming\*\*: Quick and simple, but may produce non-words.  
- \*\*Lemmatization\*\*: More accurate and meaningful.  
- \*\*Unicode Normalization\*\*: Ensures special characters are stored and compared consistently.