2.9 Practical Task: A Step-by-Step Practical Guide

Introduction

This practical exercise focuses on text preprocessing using Python's NLTK and Pandas libraries. The goal is to take raw hotel reviews from TripAdvisor and transform them into a format suitable for Natural Language Processing (NLP) tasks such as sentiment analysis or topic modeling.

The steps below represent a typical preprocessing pipeline: (1) importing and understanding the data; (2) cleaning text (lowercasing, removing punctuation and stopwords); (3) tokenizing words; (4) applying stemming and lemmatization; and (5) creating and analyzing n-grams.

1. Importing the Required Libraries

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.corpus import stopwords
import re
import pandas as pd
```

• nltk: Natural Language Toolkit — provides tokenizers, stemmers, lemmatizers, and stopword lists.
• pandas: Handles data in tabular format using DataFrames.
• re: The regular expressions module for advanced text cleaning and pattern matching.

Important NLTK Components

• word_tokenize: splits text into individual words or tokens.
• PorterStemmer: reduces words to their root forms (e.g., "running" → "run").
• WordNetLemmatizer: converts words to their meaningful base forms using vocabulary and grammar context (e.g., "better" → "good").
• stopwords: common English words like "the", "is", "and" that are usually removed.

2. Downloading Required NLTK Resources

```
nltk.download('punkt_tab')
nltk.download('stopwords')
```

These downloads ensure the environment has the linguistic data required for tokenizing and filtering words.

3. Loading the Dataset

```
data = pd.read_csv("tripadvisor_hotel_reviews.csv")
```

The dataset contains hotel reviews and ratings from TripAdvisor. Each row represents one review. You can confirm the structure using:

```
data.info()
data.head()
```

4. Inspecting the First Review

```
data['Review'][0]
```

This displays the first review in the dataset. It is usually a raw text string — possibly mixed with uppercase letters, punctuation, and filler words.

5. Converting Text to Lowercase

```
data['review_lowercase'] = data['Review'].str.lower()
```

This converts all text to lowercase to maintain consistency. Without this step, words like “Hotel” and “hotel” would be treated as different tokens.

6. Removing Stopwords (Except “not”)

```
en_stopwords = stopwords.words('english')
en_stopwords.remove("not")
```

Stopwords are common words that carry little meaning. By removing them, you keep only the informative words that express content and sentiment. However, the word “not” is kept because it can change sentiment entirely (e.g., “good” vs. “not good”).

7. Removing Stopwords from the Text

```
data['review_no_stopwords'] = data['review_lowercase'] \
.apply(lambda x: ' '.join([word for word in x.split() if word not in
en_stopwords]))
```

This line performs three things: (1) splits each review into words using .split(); (2) filters out any word found in the stopword list; and (3) rejoins the remaining words into a single string using ' '.join().

Python Concept: Lambda Functions and .apply()

A lambda function is an inline anonymous function for short, one-off use (e.g., *lambda x: x + 1*). The .apply() method in Pandas applies a function to every element in a column or row. Here, it applies the lambda function to each review. This combination is powerful for transforming entire text columns efficiently.

8. Replacing Special Characters (Punctuation Cleaning)

```
data['review_no_stopwords_no_punct'] = data.apply(
lambda x: re.sub(r"[*]", "star", x['review_no_stopwords']), axis=1)
```

This replaces asterisks (*) with the word “star.” For example, if a review said “5 * hotel”, it becomes “5 star hotel.” This ensures symbols expressing ratings are converted into meaningful words.

```
data['review_no_stopwords_no_punct'] = data.apply(
lambda x: re.sub(r"([^\w\s])", "", x['review_no_stopwords_no_punct']), axis=1)
```

Python Concept: Regular Expressions (re.sub)

re.sub(pattern, replacement, text) searches for the pattern in the text and replaces it with replacement. In r"([^\w\s])", ^ means “not”, \w matches any word character, and \s matches whitespace. So this removes punctuation marks, emojis, and other non-alphanumeric symbols.

9. Tokenization – Splitting Text into Words

```
data['tokenized'] = data.apply(
lambda x: word_tokenize(x['review_no_stopwords_no_punct']), axis=1)
```

Tokenization splits text into words or tokens that NLP models can work with.

10. Stemming – Reducing Words to Root Forms

```
ps = PorterStemmer()
data["stemmed"] = data["tokenized"].apply(lambda tokens: [ps.stem(token) for
token in tokens])
```

Stemming removes suffixes to get word roots (not always grammatically correct). This helps standardize words before analysis.

Python Concept: List Comprehension

The expression *[ps.stem(token) for token in tokens]* means: “For every token in the list tokens, apply ps.stem() to it, and collect the results in a new list.” It is a compact alternative to building a list with a for-loop and append().

11. Lemmatization – Meaningful Root Forms

```
lemmatizer = WordNetLemmatizer()
data["lemmatized"] = data["tokenized"].apply(lambda tokens:
[lemmatizer.lemmatize(token) for token in tokens])
```

Lemmatization uses a vocabulary and grammar rules to return meaningful base forms (e.g., “running” → “run”, “better” → “good”). This keeps valid words that make sense linguistically.

12. Building a Clean Token List

```
tokens_clean = sum(data['lemmatized'], [])
```

This flattens all lists of tokens (from all reviews) into one long list using Python’s sum() with an empty list as the start value.

13. Creating n-Grams

n-Grams are sequences of n consecutive words, used to capture patterns in language.

Unigrams (n=1)

```
unigrams = pd.Series(nltk.ngrams(tokens_clean, 1)).value_counts()
```

Bigrams (n=2)

```
bigrams = pd.Series(nltk.ngrams(tokens_clean, 2)).value_counts()
```

4-Grams (n=4)

```
ngrams_4 = pd.Series(nltk.ngrams(tokens_clean, 4)).value_counts()
```

When N > 3, we generally call them N-grams.

14. Summary of the Workflow

Step	Transformation	Example
1	Lowercasing	“Clean Room” → “clean room”
2	Stopword removal	“the hotel was not clean” → “hotel not clean”
3	Punctuation removal	“hotel!” → “hotel”
4	Tokenization	“hotel not clean” → ['hotel', 'not', 'clean']
5	Stemming	“cleaned” → “clean”
6	Lemmatization	“better” → “good”
7	n-Grams	“friendly staff” → ('friendly', 'staff')