Lecture-2

Text Preprocessing

Text preprocessing is essential for handling variations, noise, and inconsistencies in text data. Techniques like cleaning, normalization, tokenization, stemming, and lemmatization make text consistent and machine-readable. Preprocessing enables accurate analysis and understanding in applications like sentiment analysis and chatbots.

Question:

"You are building a sentiment analysis tool for movie reviews. You notice that users write reviews in many different ways, such as:

- *'The movie was fantastic!!!'*
- *'It was the worst movie ever :('*
- *'meh, the movie was ok...'*
- *'The movi was gr8' (with typos and abbreviations)*

How would you ensure that your tool correctly understands the sentiment of these reviews, even with variations in writing style, typos, and abbreviations?"

1. **Text Cleaning**:
    - Remove punctuation, special characters, and emojis:
        - `'The movie was fantastic!!!'` → `'the movie was fantastic'`
        - `'It was the worst movie ever :('` → `'it was the worst movie ever'`
    - Correct typos and standardize abbreviations:
        - `'The movi was gr8'` → `'the movie was great'`
2. **Normalization**:
    - Convert text to lowercase to ensure consistency:
        - `'The movie was fantastic'` → `'the movie was fantastic'`
3. **Tokenization**:
    - Split text into individual words or tokens:
        - `'the movie was fantastic'` → `['the', 'movie', 'was', 'fantastic']`
4. **Stopword Removal**:
    - Remove common words that don't contribute much to sentiment (e.g., "the," "was"):
        - `['the', 'movie', 'was', 'fantastic']` → `['movie', 'fantastic']`
5. **Stemming/Lemmatization**:
    - Reduce words to their base forms:
        - `'fantastic'` → `'fantastic'` (no change)
        - `'worst'` → `'worst'` (no change)
        - `'amazing'` → `'amaz'` (stemming) or `'amazing'` (lemmatization)
6. **Handling Negations**:
    - Preserve negations to capture sentiment accurately:
        - `'not amazing'` → Treat as a negative sentiment.
7. **Feature Extraction**:
    - Convert text into numerical features (e.g., using Bag of Words, TF-IDF, or word embeddings).

8. **Sentiment Analysis**:
    - Use a machine learning model (e.g., Naive Bayes, LSTM) to classify the sentiment as positive, negative, or neutral.

Final Output:
After preprocessing, the reviews are cleaned, normalized, and ready for sentiment analysis. For example:

- `'The movie was fantastic!!!'` → Positive
- `'It was the worst movie ever :('` → Negative
- `'meh, the movie was ok...'` → Neutral
- `'The movi was gr8'` → Positive

Question?

You are designing a chatbot for a pizza delivery service. Customers might type their orders in many different ways, such as:

- *'I want a large pepperoni pizza.'*
- *'Can I get a pepperoni pizza, large?'*
- *'plz giv me L pep pizza' (with abbreviations and typos)*
- *'I'd like a big pepperoni pizza!!!'*

How would you teach the chatbot to understand these variations and process the orders correctly?"

Steps to Address the Challenge:

1. **Text Cleaning**:
   - Remove punctuation, special characters, and emojis:
     - `'I want a large pepperoni pizza.'` → `'i want a large pepperoni pizza'`
     - `'plz giv me L pep pizza'` → `'plz giv me l pep pizza'`
2. **Normalization**:
   - Convert text to lowercase:
     - `'I want a large pepperoni pizza.'` → `'i want a large pepperoni pizza'`
   - Expand abbreviations and correct typos:
     - `'plz giv me L pep pizza'` → `'please give me large pepperoni pizza'`
3. **Tokenization**:
   - Split text into individual words or tokens:
     - `'i want a large pepperoni pizza'` → `['i', 'want', 'a', 'large', 'pepperoni', 'pizza']`
4. **Stopword Removal**:
   - Remove common words that don't contribute much to the order (e.g., "i," "want," "a"):
     - `['i', 'want', 'a', 'large', 'pepperoni', 'pizza']` → `['large', 'pepperoni', 'pizza']`
5. **Stemming/Lemmatization**:
   - Reduce words to their base forms (optional, depending on the use case):
     - `'pepperoni'` → `'pepperoni'` (no change)
     - `'pizzas'` → `'pizza'` (lemmatization)
6. **Intent Recognition**:
   - Use Natural Language Understanding (NLU) to identify the intent (e.g., "order pizza") and extract key information:
     - Size: `'large'`
     - Topping: `'pepperoni'`
     - Item: `'pizza'`
7. **Slot Filling**:
   - Map extracted information to predefined slots in the chatbot:
     - `{'size': 'large', 'topping': 'pepperoni', 'item': 'pizza'}`
8. **Response Generation**:
   - Generate a response based on the extracted information:
     - `'You ordered a large pepperoni pizza. Is that correct?'`

Final Output:

After preprocessing, the chatbot can understand and process orders accurately, regardless of how they are written. For example:

- `'I want a large pepperoni pizza.'` → `'You ordered a large pepperoni pizza. Is that correct?'`
- `'plz giv me L pep pizza'` → `'You ordered a large pepperoni pizza. Is that correct?'`
- `'Can I get a pepperoni pizza, large?'` → `'You ordered a large pepperoni pizza. Is that correct?'`

Text Preprocessing: Cleaning and Structuring Text for Analysis

Before understanding text, a machine must **prepare it in a structured form**. This step is crucial because raw text contains inconsistencies, extra spaces, punctuation, and irrelevant words that may affect analysis.

Why Do Machines Need Clean Data Before Analysis?

Before a machine can understand and process human language, the text needs to be **cleaned and structured**. Unlike humans, who can **ignore typos, slang, and grammatical mistakes**, machines require **structured input** to perform well.

📌 **Key Reasons for Text Preprocessing:**
✅ **Removes Noise** → Extra spaces, punctuation, numbers, etc.
✅ **Standardizes Data** → Converts words to their base form for uniformity.
✅ **Reduces Data Size** → Eliminates unnecessary words (stopwords).
✅ **Improves NLP Model Accuracy** → Helps models understand text better.

Without preprocessing, an NLP model might misinterpret text and fail to produce **accurate results**.

Tokenization

Tokenization is the process of breaking a stream of text into meaningful elements (tokens), such as words, sentences, or subwords, for computational processing in NLP.

📌 Example:

Input Text:
"Natural Language Processing is amazing!"

Word Tokenization Output:
["Natural", "Language", "Processing", "is", "amazing", "!"]

Sentence Tokenization Output:
["Natural Language Processing is amazing!"]

Tokenization types:

1️⃣ Word Tokenization

- Splits text into individual words.
- Removes punctuation but retains meaningful words.

📌 **Example:**
Input:
"NLP is amazing, and I love learning it!"
Output (Word Tokens):
["NLP", "is", "amazing", "and", "I", "love", "learning", "it"]

✅ **Use Case:** Text classification, Named Entity Recognition (NER).

2️⃣ Sentence Tokenization

- Splits text into sentences.
- Helps in understanding sentence-level meaning.

📌 **Example:**
Input:
"NLP is exciting. It powers chatbots and search engines!"
Output (Sentence Tokens):
["NLP is exciting.", "It powers chatbots and search engines!"]

✅ **Use Case:** Machine translation, summarization.

3️⃣ Subword Tokenization

- Splits words into smaller subword units.
- Useful for handling **rare words, new words, and morphologically rich languages**.

["understand", "understanding", "misunderstand", "misunderstanding", "misunderstandings"]

Total unique tokens required: 5

❌ **Problem: If we add more variations (e.g., misunderstood, misunderstandable), the vocabulary size keeps increasing.**

a) Byte Pair Encoding (BPE) Tokenization

📌 Subword Tokenization (Smaller Vocabulary)

Subword tokenization breaks words into **smaller meaningful units** that can be **reused** in different contexts.

Breaking "Misunderstandings" into Subwords (BPE/WordPiece Output):

["mis", "##under", "##stand", "##ing", "##s"]

➡ Total unique tokens required: 5

Full Word	Subword Tokens (BPE/WordPiece)	Meaning / Context
misunderstandings	`["mis", "##under", "##stand", "##ing", "##s"]`	Multiple failures to understand something
misunderstanding	`["mis", "##under", "##stand", "##ing"]`	A single instance of failing to understand
misunderstood	`["mis", "##under", "##stood"]`	Past tense of misunderstanding
misunderstand	`["mis", "##under", "##stand"]`	To fail to understand something correctly
understand	`["under", "##stand"]`	To comprehend something
understanding	`["under", "##stand", "##ing"]`	The act of comprehension
understands	`["under", "##stand", "##s"]`	Third-person singular form of "understand"

- Combines frequent character pairs iteratively.
- Used in **GPT, BERT, and modern NLP models**.

✅ **Use Case:** Transformer models, text generation.

b) WordPiece Tokenization

- Used in **BERT** and **Google's NLP models**.
- Similar to BPE but with more probabilistic splitting.

📌 **Example:**
Word: *"unhappiness"*
Output: ["un", "##happiness"] (## indicates a subword)

✅ **Use Case:** Pre-trained deep learning models (e.g., BERT).

4️⃣ Character Tokenization

Project (Explained step by step): A character-level RNN reads words as a series of characters an d predict the class the word belongs to.

https://www.youtube.com/watch?v=XxcpWBb4Ijc&list=PLuwnGjM1nRi3SbSqyo0zydC3w58iF0p0S&index=3

- Splits text into individual characters.
- Used for **handling typos and languages without spaces (e.g., Chinese, Japanese).**

📌 **Example:**
Input: *"Hello"*
Output: ["H", "e", "l", "l", "o"]

✅ **Use Case:** Spell checking, character-level text generation.

5️⃣ N-gram Tokenization

- Splits text into **overlapping sequences of words** (N-grams).
- Useful for **context understanding** in **language models**.

📌 **Example (Bi-gram Tokenization - N=2):**
Input: *"Natural Language Processing is powerful"*
Output:
["Natural Language", "Language Processing", "Processing is", "is powerful"]

✅ **Use Case:** Text prediction, autocomplete, topic modeling.

Tokenization Type	Splits Into	Example Output	Use Cases
Word Tokenization	Words	`["NLP", "is", "fun"]`	Text classification, Named Entity Recognition
Sentence Tokenization	Sentences	`["NLP is fun.", "It is useful."]`	Machine translation, summarization
Subword Tokenization (BPE/WordPiece)	Subwords	`["un", "##happiness"]`	Transformer models (BERT, GPT)
Character Tokenization	Characters	`["N", "L", "P"]`	Spell checking, text generation
N-gram Tokenization	Word sequences	`["Natural Language", "Language Processing"]`	Context modeling, autocomplete

Stemming vs. Lemmatization: Key Differences

Both **stemming** and **lemmatization** are text preprocessing techniques used to reduce words to their root form. However, they differ in approach, accuracy, and usability.

🔷 1. Stemming (Rule-Based, Fast, Less Accurate)

- **Definition:** Reduces words to their base/root form by **removing suffixes** using **rule-based heuristics**.
- **Approach:** Uses **simple chopping rules** to remove common suffixes (e.g., *ing, ed, es*).
- **Output:** May produce **non-real words** (e.g., *running* → *runn*).

📌 **Example:**

Word	Stemming Output (Porter Stemmer)
Running	Runn
Studies	Studi
Better	Better (No change)
Organization	Organiz

✅ **Pros:**
✔ Fast and computationally efficient.
✔ Useful when word meaning is **not critical**.

❌ **Cons:**
❌ Can generate non-dictionary words.
❌ May cause **over-stemming** (incorrect truncation).

Applications of stemming-

1. Search Engines (Information Retrieval): Search engines like Google, Bing, and Elasticsearch need to match user queries to relevant web pages. Stemming helps reduce variations of a word to a common root, increasing search recall.

📌 Example:

User Query: "running shoes"

Indexed Keywords (After Stemming): "run shoe"

Now, even if a webpage contains "runn" or "running", the search engine still retrieves it.

Stemming is useful when **speed and computational efficiency** are more important than accuracy. Since stemming is a **rule-based** technique that simply chops off word endings, it is **faster** but may not always return a proper dictionary word. Here are specific use cases where stemming is helpful:

✅ 1. Search Engines (Information Retrieval)

🔹 **Why?**

- Search engines like **Google, Bing, and Elasticsearch** need to match user queries to relevant web pages.
- Stemming helps **reduce variations of a word** to a common root, increasing search recall.

📌 **Example:**
User Query: *"running shoes"*
Indexed Keywords (After Stemming): "run shoe"

- Now, even if a webpage contains "runn" or "running", the search engine still retrieves it.

✅ 2. Keyword-Based Text Matching (Plagiarism Checkers, Document Retrieval)

🔹 **Why?**

- Stemming helps **match similar words in different forms**.
- Used in **plagiarism detection, legal document retrieval, and research paper indexing**.

📌 **Example:**
If a **legal search system** stems *"arguing, argued, argues"* to "argu", it ensures that **all forms** of the word are matched in court documents.

✅ 3. Spam Detection & Text Classification (Fast Processing Required)

🔹 **Why?**

- Spam filters analyze email/text content quickly.
- Stemming allows the model to **process words faster** and detect spam-related words.

📌 **Example:**
Spam words: *"clicking, clicked, clicks"* → Stemmed to "click"
Now, if an email contains **"click here to win!"**, the system can match it to known spam patterns.

✅ 4. Search Query Expansion (Autocomplete, Recommendations)

🔹 **Why?**

- Stemming expands **short search queries** by mapping different word variations.
- Helps improve **autocomplete suggestions** and **product recommendations**.

📌 **Example:**

- A user types **"play games"** in a gaming website search.
- The system stems **"playing" and "played"** to "play", returning **broader results**.

✅ 5. Topic Modeling & Text Mining (Faster Processing for Large Datasets)

🔹 **Why?**

- When analyzing **millions of text documents**, stemming helps **reduce vocabulary size**.
- This speeds up **topic modeling, clustering, and text analysis**.

📌 **Example:**

- In **Twitter sentiment analysis**, instead of treating **"loved, loving, loves"** as different words, stemming reduces them to "love", simplifying processing.

📌 When to Use Stemming vs. Lemmatization?

Use Case	Use Stemming?	Use Lemmatization?
Search engines	✅ Yes	❌ No (Too slow)
Spam detection	✅ Yes	❌ No
Fast text matching	✅ Yes	❌ No
Chatbots & NLP assistants	❌ No	✅ Yes
Sentiment analysis	✅ Yes (for speed)	✅ Yes (for accuracy)

Note:

Do We Use Stemming on Subword Tokenization? Is It Required?

🚀 **Short Answer: No, stemming is generally not required when using subword tokenization (BPE, WordPiece, SentencePiece).**

Stopword Removal: Eliminating Unnecessary Words

Stopwords are **common words** that do not add much meaning to a sentence (e.g., *is, the, and, a, this, that*).

📌 **Example:**
Input:
"The cat is sitting on the mat."
Without Stopwords:
➡ "cat sitting mat"

Why Remove Stopwords?

✅ Makes text **more meaningful**.
✅ Reduces **computational cost** for NLP models.
✅ Focuses on **important words** in a sentences

Examples of NLP in Action

1. Text Processing

Example: Tokenization (Breaking text into words)

```
from nltk.tokenize import word_tokenize

text = "NLP is a fascinating field!"

print(word_tokenize(text))
```

Output: ['NLP', 'is', 'a', 'fascinating', 'field', '!']

---

2. Speech Recognition

Example: Converting speech to text
(Used in Siri, Google Assistant, Alexa)

```
import speech_recognition as sr
recognizer = sr.Recognizer()
with sr.Microphone() as source:
    print("Say something...")
    audio = recognizer.listen(source)
    text = recognizer.recognize_google(audio)
    print("You said:", text)
```

Output: "You said: What is NLP?" *(Based on spoken input)*

---

3. Named Entity Recognition (NER)

Example: Extracting named entities (People, Places, Organizations, etc.)

```
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Elon Musk founded SpaceX in 2002.")
for ent in doc.ents:
    print(ent.text, "->", ent.label_)
```

Output:

```
Elon Musk -> PERSON
SpaceX -> ORG
2002 -> DATE
```

---

4. Sentiment Analysis

Example: Detecting positive or negative sentiment

```
from textblob import TextBlob
text = "I love learning NLP, it's amazing!"
sentiment = TextBlob(text).sentiment.polarity
print("Sentiment Score:", sentiment)
```

Output: Sentiment Score: 0.75 *(Positive sentiment)*

---

5. Machine Translation

Example: Translating English to French

```
from transformers import MarianMTModel, MarianTokenizer
model_name = "Helsinki-NLP/opus-mt-en-fr"
tokenizer = MarianTokenizer.from_pretrained(model_name)
model = MarianMTModel.from_pretrained(model_name)
text = "Hello, how are you?"
tokens = tokenizer(text, return_tensors="pt")
translation = model.generate(**tokens)
print(tokenizer.decode(translation[0], skip_special_tokens=True))
```

Output: "Bonjour, comment ça va ?"

---

6. Text Summarization

Example: Extracting a summary from a long paragraph

```
from transformers import pipeline

summarizer = pipeline("summarization")

text = "Natural Language Processing is a fascinating field of AI. It allows computers to understand and generate human language. Applications of NLP include chatbots, translation, speech recognition, and more."

summary = summarizer(text, max_length=30, min_length=10, do_sample=False)

print(summary[0]['summary_text'])
```

Output: "NLP enables computers to understand and generate human language."