# NLP

NLP, or Natural Language Processing, is a field of artificial intelligence that focuses on enabling computers to understand, interpret, and generate human language, allowing for interaction and tasks like translation and text analysis.

## Applications

- Language Translation: Translating text or speech from one language to another.
- Sentiment Analysis: Determining the emotional tone or opinion expressed in a piece of text.
- Question Answering: Allowing computers to understand and answer questions posed in natural language.
- Speech Recognition: Converting spoken words into text.
- Chatbots and Virtual Assistants: Enabling computers to engage in natural language conversations.
- Text Summarization: Condensing long texts into shorter, more manageable summaries.
- Named Entity Recognition: Identifying and classifying entities like people, organizations, and locations in text.
- Autocompletion
- Spam Filtration

# Regex for NLP

Regular expressions (regex) are powerful tools in Natural Language Processing (NLP) for tasks like text cleaning, pattern matching, data extraction, and validation, enabling efficient manipulation and analysis of text data.

- A regular expression (regex) is a sequence of characters that defines a search pattern.
- They are used to find, extract, and manipulate text based on specific patterns.
- Think of them as a language for specifying text search strings.

They are used for Text Preprocessing, Pattern Matching, Data Extraction, Data Validation, and Information Retrieval.

## Examples of Regex Usage in NLP:

- **Extracting email addresses:** [\w\.-]+@[\w\.-]+\.\w+
- **Finding phone numbers:** \d{3}-\d{3}-\d{4} or \(\d{3}\) \d{3}-\d{4}
- **Removing punctuation:** [^a-zA-Z0-9\s]
- **Tokenizing text:** Splitting text into words or sentences based on spaces or punctuation.
- **Identifying specific words:** \bcat\b (finds the word "cat" as a whole word, not part of another word like "cats")

## Key Concepts in Regex:

- **Metacharacters:**

Special characters with specific meanings, like **.** (any character), ***** (zero or more occurrences), **+** (one or more occurrences), **?** (zero or one occurrence), **[]** (character class), **()** (grouping).

- **Character Classes:**

Sets of characters, like **[a-z]** (lowercase letters), **[0-9]** (digits), **\s** (whitespace).

- **Quantifiers:**

Symbols that specify how many times a character or group should appear, like *, +, ?, {n}, {n,}, {n,m}.

- **Anchors:**

Characters that specify the position of a match, like ^ (beginning of string), $ (end of string), \b (word boundary).

# NLP Core Concepts

### 1. Tokenization

Tokenization is the process of breaking text into smaller units (tokens), which can be **words, subwords, or sentences**.

#### Types of Tokenization

1. **Word Tokenization**
    - Splitting a sentence into words.
    - Example:
    - Text: "I love NLP!"
    - Word Tokens: ['I', 'love', 'NLP', '!']
2. **Subword Tokenization**
    - Splitting words into meaningful subwords (used in BERT, GPT).
    - **Example:**
    - Word: "unhappiness"
    - Subword Tokens: ['un', 'happiness']
3. **Sentence Tokenization**
    - Splitting text into sentences.
    - **Example:**
    - Text: "I love NLP. It's amazing!"
    - Sentence Tokens: ["I love NLP.", "It's amazing!"]

📌 **Libraries for Tokenization**:

- nltk.word_tokenize(text), nltk.sent_tokenize(text)
- spacy tokenizer
- Hugging Face tokenizers for subword models

---

### 2. Stopwords Removal

Stopwords are common words (e.g., *the, is, in, at, which, and*) that don't add much meaning and can be removed to improve efficiency.

### Example

```
from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

text = "This is a simple NLP example"

tokens = word_tokenize(text)

filtered_tokens = [word for word in tokens if word.lower() not in stopwords.words('english')]

print(filtered_tokens)  # ['simple', 'NLP', 'example']
```

📌 **Stopwords Lists**:

- nltk.corpus.stopwords.words('english')
- spacy has built-in stopwords

---

### 3. Stemming vs Lemmatization

Both techniques **reduce words to their base/root form**, but they work differently.

### Stemming

- Removes suffixes but may not return valid words.
- Uses **heuristic rules**.
- Example (Porter Stemmer):
- `from nltk.stem import PorterStemmer`
- `stemmer = PorterStemmer()`
- `print(stemmer.stem("running"))  # Output: run`
- `print(stemmer.stem("flies"))    # Output: flies`

### Lemmatization

- Uses a **dictionary lookup** to return valid words.
- More accurate than stemming.

- Example (WordNet Lemmatizer):
- from nltk.stem import WordNetLemmatizer
- lemmatizer = WordNetLemmatizer()
- print(lemmatizer.lemmatize("running", pos="v"))  # Output: run
- print(lemmatizer.lemmatize("flies", pos="n"))    # Output: fly

📌 When to Use?

- Stemming: If speed matters and accuracy isn't critical.
- Lemmatization: If you need correct words (better for NLP tasks like search engines).

---

### 4. POS Tagging (Part of Speech Tagging)

Assigns grammatical labels (noun, verb, adjective, etc.) to words in a sentence.

Example using spaCy

```
import spacy

nlp = spacy.load("en_core_web_sm")

text = "The quick brown fox jumps over the lazy dog."

doc = nlp(text)

for token in doc:
    print(token.text, "->", token.pos_)
```

Output:

The -> DET

quick -> ADJ

brown -> ADJ

fox -> NOUN

jumps -> VERB

over -> ADP

the -> DET

lazy -> ADJ

dog -> NOUN

📌 **Why is POS Tagging Useful?**

- Improves Named Entity Recognition (NER)
- Helps in **lemmatization** (knowing "running" is a verb helps get "run")
- Used in **syntactic parsing**

---

### 5. Named Entity Recognition (NER)

NER identifies and categorizes entities (names, places, dates, organizations, etc.).

**Example using spaCy**

```
import spacy

nlp = spacy.load("en_core_web_sm")

text = "Elon Musk founded SpaceX in 2002."

doc = nlp(text)

for ent in doc.ents:

    print(ent.text, "->", ent.label_)
```

**Output:**

Elon Musk -> PERSON

SpaceX -> ORG

2002 -> DATE

📌 **NER Applications**:

- Extracting **company names** from resumes
- **Medical NLP** (finding diseases, drugs in medical text)
- Chatbot intelligence (identifying **locations, names** in user input)

---

### 6. Dependency Parsing

- Analyzes the **grammatical structure** of a sentence.
- Determines how words relate to each other.

**Example using spaCy**

```
import spacy

nlp = spacy.load("en_core_web_sm")

text = "The cat sat on the mat."

doc = nlp(text)

for token in doc:

    print(token.text, "->", token.dep_, "->", token.head.text)
```

**Output:**

The -> det -> cat

cat -> nsubj -> sat

sat -> ROOT -> sat

on -> prep -> sat

the -> det -> mat

mat -> pobj -> on

📌 **Key Terms**

- **ROOT**: The main verb of the sentence
- **nsubj**: Nominal subject (who/what performs the action)
- **det**: Determiner (e.g., *the, a*)
- **pobj**: Object of the preposition

✅ **Applications**

- Question answering systems
- Machine translation
- Grammar checking tools

---

## 7. TF-IDF & Bag of Words (BoW)

### Bag of Words (BoW)

- Converts text into a **vector of word counts**.
- Ignores order but captures frequency.

**Example**

| Sentence | NLP | is | fun |
|---|---|---|---|
| "NLP is fun" | 1 | 1 | 1 |
| "NLP is great" | 1 | 1 | 0 |

### 📌 Limitations

- Doesn't consider **word importance**.
- Large vocab size → **high-dimensional vectors**.

---

### TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF improves BoW by giving **more weight to rare words** and **less weight to common words**.

- **TF (Term Frequency)** = (Word count in doc) / (Total words in doc)
- **IDF (Inverse Document Frequency)** = log (Total docs / Docs containing word)

Example:

- "NLP" appears in 5 out of 10 documents → Low IDF score.
- "Transformer" appears in 1 out of 10 documents → High IDF score.

**Example using Sklearn**

```
from sklearn.feature_extraction.text import TfidfVectorizer

docs = ["NLP is fun", "NLP is great"]

vectorizer = TfidfVectorizer()

tfidf_matrix = vectorizer.fit_transform(docs)
```

```
print(vectorizer.get_feature_names_out())  # ['fun', 'great', 'is', 'nlp']
print(tfidf_matrix.toarray())  # TF-IDF values
```

✅ TF-IDF vs BoW?

- BoW is simpler and used in basic NLP tasks.
- TF-IDF is better when word importance matters (e.g., search engines).

## 1. Transformer Models: Overview

Transformers revolutionized NLP by replacing RNNs and LSTMs. They use the **self-attention mechanism** and **positional encoding** to process text in parallel, leading to models like **BERT, GPT, and T5**.

### 🔹 Key Components of a Transformer

1. **Self-Attention Mechanism** – Enables the model to focus on important words.
2. **Multi-Head Attention** – Enhances attention by using multiple perspectives.
3. **Positional Encoding** – Adds order information to words.
4. **Feedforward Layers** – Used after self-attention for transformation.
5. **Layer Normalization & Dropout** – Improves training stability.

---

## 2. Self-Attention Mechanism

Self-attention allows a model to weigh the importance of each word relative to others in a sentence. This is key for understanding context.

### How Does It Work?

Given a sentence:
💬 *"The cat sat on the mat."*

- Traditional models might struggle to capture dependencies like **"The cat"**.
- **Self-attention** enables the model to learn which words are important for each token.

### Mathematical Formulation

For each input word (token), we compute:

1. **Query (QQ), Key (KK), and Value (VV) Matrices**
    - These are learned transformations of input embeddings.
    - If input dimension = dmodeld_{model}, then:
        - Q=XWQQ = XW_Q
        - K=XWKK = XW_K

        - V=XWVV = XW_V
          (where WQ,WK,WVW_Q, W_K, W_V are weight matrices)

2. Attention Score Calculation (Scaled Dot-Product Attention)
    - Compute scores using dot product of Query and Key:
      Attention(Q,K,V)=softmax(QKTdk)V\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V
    - dkd_k is the dimension of keys (used for scaling).
    - Softmax ensures attention weights sum to 1.

Example

If we process the sentence *"The cat sat on the mat."*

- The word **"cat"** attends to **"sat"** with high weight.
- **"The"** attends more to **"cat"**, but not much to **"on the mat"**.

✅ Benefits of Self-Attention

- Captures **long-range dependencies** in sentences.
- Works in **parallel**, unlike RNNs (which process sequentially).
- Used in **BERT (bi-directional context)** and **GPT (auto-regressive context)**.

---

### 3. Retrieval-Augmented Generation (RAG)

RAG enhances text generation by **retrieving relevant information** before generating a response.

Why RAG?

- LLMs (e.g., GPT) have **fixed knowledge** (limited to training data).
- RAG **fetches external knowledge** dynamically, improving accuracy.
- Useful in **chatbots, search engines, question answering, summarization**.

How Does RAG Work?

RAG has **two main components**:

1. Retriever

- Uses a **vector database** (like FAISS, Pinecone, ChromaDB)
- Searches for relevant documents based on user input
- Example: A search engine finds the top 5 Wikipedia pages for a query.

2. **Generator**
   - Uses an **LLM (GPT, T5, etc.)**
   - Combines retrieved knowledge with the prompt
   - Generates a final answer

**Example Workflow**

1. User asks: *"Who is the CEO of OpenAI?"*
2. **Retriever** finds recent articles mentioning OpenAI's CEO.
3. **Generator** synthesizes a response using retrieved data.

**Mathematical View of RAG**

$P(y|x,z) = P(y|x,R(x))P(y \mid x, z) = P(y \mid x, R(x))$

Where:

- $x$ = user query
- $R(x)$ = retrieved documents
- $y$ = final generated response

📌 **Vector Search in RAG**

- Uses **word embeddings** to compare text similarity.
- Converts text to vectors using **BERT embeddings**.
- Stores & retrieves from **vector databases (FAISS, Pinecone, Weaviate)**.

---

**4. RAG Implementation Example**

```
from transformers import RagTokenizer, RagRetriever, RagSequenceForGeneration


# Load model
```

```
tokenizer = RagTokenizer.from_pretrained("facebook/rag-sequence-nq")

retriever = RagRetriever.from_pretrained("facebook/rag-sequence-nq")

model = RagSequenceForGeneration.from_pretrained("facebook/rag-sequence-nq", retriever=retriever)


# Input query

query = "What is the capital of France?"

inputs = tokenizer(query, return_tensors="pt")


# Generate response

generated = model.generate(**inputs)

response = tokenizer.batch_decode(generated, skip_special_tokens=True)

print(response) # Output: "The capital of France is Paris."
```

---

Summary

| Concept | Explanation |
|---|---|
| Self-Attention | Helps a model focus on important words in a sequence. |
| Multi-Head Attention | Uses multiple attention heads for better representation. |
| RAG (Retrieval-Augmented Generation) | Combines search (retrieval) with LLM generation. |
| Vector Databases | Stores and retrieves embeddings for similarity search. |