Natural Language Processing (NLP)

- **Regex** *(Pattern matching in text)*
- **Transformers** *(e.g., BERT, GPT)*
    - **Large Language Models (LLMs)** *(e.g., GPT-4, Llama, Claude)*
- **GEN AI** is a part of NLP

What is NLP?:

NLP is a field in computer science and Al that gives machines an ability to understand human language better and to assist in language related tasks

Chat GPT is a tool NLP application

Auto completion in gmail.

Language Translation

Text extraction (regex)

NLP: Information Extraction, Text Classification, NER

Both : Text generation, summarization, chatbots, Language Translation

GEN AI: Image, audio, video generation

Three Techniques of doing NLP:

Rule Based:

Extracting the text based on some criteria

ReGex

Machine Learning

Count vectorizer the words and using Statistical Model to solve

Spam classification

Deep Learning:

BERT models

NLP Tasks:

Ticket for a bug reported

A ticket is converted to embedding based on some Navies bayes classifier it divides into High , medium and low priorities and gets assigned to the human assistance.

Extract text from the images using OCR and using Doc2Vec convert to vector and using some classification models classify the problem

Text similarity

Named Entity Recognition

Chat BoT

Information Retrieval → retrieving the required information

Text summarization

Topic Modeling

NLP Pipeline:

Based on the Problem [Ticket Classification]:

Sentence → Sentence Tokenization → making into sentences → word Tokenization → words → Stop word removal → stemming, Lemmatization →

Data Acquisition→Pre processing → Feature Engineering → parsing & syntax Analysis → Model Building → post-processing & evaluation→ Deployment → Monitor & update [can go back to steps if the accuracy is not expected]

Tools Overview:

State of the Art NLP: Hugging Face, LLM Apis, LangChain

Classical NLP, Low-latency Pipelines : SpaCy, Gensim

Teaching Experimentation Research: NLTK

SpaCy Vs NLTK:

Provides most efficient NLP algorithm for a given task. Hence if you care about the end result, go with Spacy

Provides access to many algorithms. If you care about specific algo and customizations go with NLTK (manual mode camera gives you lot of freedom to customize)

```
import spacy

nlp = spacy.load('en_core_web_sm')

doc = nlp('Dr.Strange loves pav bhaji of Mumbai. Hulk loves chatt of delhi')

for sentence in doc.sents:

    print(sentence)

import nltk

from nltk.tokenize import sent_tokenize

sent_tokenize('Dr. Strange loves pav bhaji of Mumbai. Hulk loves chatt of delhi')
```

Spacy

- Spacy is Object Oriented
- Spacy is user friendly
- Provides most efficient NLP algorithm for a given task. Hence if you care about the end result, go with Spacy
- Spacy is new library and has a very active user community

NLTK

- NLTK is mainly a string processing library
- NLTK is also user friendly but probably less user friendly compared to Spacy
- Provides access to many algorithms. If you care about specific algo and customizations go with NLTK
- NLTK is old library. User community active as Spacy

Text Pre-processing:

Sentence Tokenization

Word Tokenization

Stop word removal

Stemming, Lemmatization

Word Tokenization:

Easy to do but will have a huge vocabulary [each word is a token]

OOV: Out-of-Vocabulary

Sub-word Tokenization: Most using

BPE → GPT

Word Piece → BERT

Sentence Piece → T5

Character Tokenization:

Using a single character as a token

- Tokenization is the foundational step in NLP, breaking down text into smaller, processable units.
- Word tokenization splits text into words, making it simple but prone to out-of-vocabulary issues.
- Character tokenization breaks text into individual characters, providing flexibility for unknown words.
- Subword tokenization efficiently handles rare words by splitting them into frequently used sub-parts.
- Subword tokenization is a popular approach used by models such as BERT, GPT etc. due to the advantages that it offers.

Tokenization in Spacy:

```
import spacy
# en is english model blank language object
nlp =spacy.blank('en') # text --> tokenizer [nlp text] --> Doc
doc = nlp('Dr. Strange love pav bhaji of mumbai as it costs only 2$ per plate.')
for token in doc:
print(token)
```

Rules:

" Let's go to N.Y.!"

First Split by prefixes

", Lets got to N.Y.!"

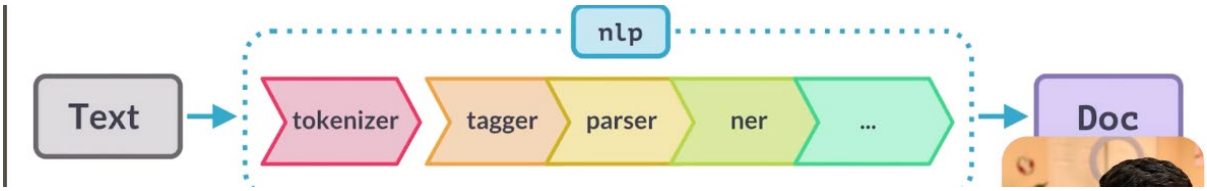
Exceptions

", Let, 's, got to N.Y.!"

Suffix

Exception

["][Let] ['s] [go] [to] [N.Y.] [!]["]



Language Processing Pipeline in Spacy:

```
Inside the blank pipeline in the blank model, we can add some components
nlp= spacy.load('en_core_web_sm')
nlp.pipe_names
doc = nlp('Captain America ate 100$ of samosa. Then he said I can do this all day.')
for token in doc:
  print(f'{token.text:<12} | {token.pos_:<12}  |  {token.lemma_:<12}')
from spacy import displacy
displacy.render(doc,style='ent')
```

Stemming & Lemmatization:

Talking → talk, eating → eat

Reducing the word to its base word (stemming)

Use fixed rules such as remove able, ing etc. to derive a base word

Ate → eat

Here you need knowledge of a language(a.k.a linguistic knowledge) to derive a base word) (Lemmatization)

Spacy doesnot support stemming

NLTK support both

```
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

words = ['eating','eats','eat','ate','adjustable','rafting','ability','meeting']

for word in words:

        print(f"{word:<10} | {stemmer.stem(word)}")

nlp = spacy.load('en_core_web_sm')

doc = nlp('eating eats eate ate adjustable rafting ability meeting better')

for token in doc:

        print(f'{token.text:<12} |  {token.lemma_:<12}  |  {token.lemma:<12}')

nlp = spacy.load('en_core_web_sm')

doc = nlp('mando talked for 3 hours although talking is not his thing he became talktive')

for token in doc:

        print(f'{token.text:<12} |  {token.lemma_:<12}')

ar = nlp.get_pipe('attribute_ruler')

ar.add([[{"TEXT":"Bro"}], [{"TEXT":"Brah"}]], {"LEMMA": "Brother"})

doc = nlp("Bro, you wanna go? Brah, don't say no! I am exhausted.")

for token in doc:

        print(f'{token.text:<12} |  {token.lemma_:<12}')
```

- Stemming reduces words to their root form by stripping suffixes, often leading to incomplete or less meaningful roots (e.g., "running" -> "run").
- Lemmatization reduces words to their base or dictionary form (lemma) by considering the word's meaning and context (e.g., "running" -> "run").
- Lemmatization is more accurate than stemming but computationally heavier, making it preferable for tasks requiring linguistic precision.

- Stemming is faster and simpler, useful for quick text normalization when perfect accuracy isn't critical.
- Both techniques improve text preprocessing by reducing word variations, aiding in tasks like search, classification, and sentiment analysis.

Part of Speech (POS) Tagging:

Noun → Person, place, thing

Verb → Action

Pronoun → Substitute of Noun

Adjective → Describe the Noun. Adds meaning to it.

Adverb → Describe verb, adjective or adverb what kind of activity (measurement) how

Interjections → Emotions [hurray alas

Conjunction → combining word [and or but]

Preposition → Links noun to another says nouns position [what the position is noun] [after, again, in]

```
nlp = spacy.load('en_core_web_sm')

doc = nlp('Elon flew to mars yesterday.He carried biryani masala with him')

for token in doc:

        print(f'{token.text:<12} |  {token.pos_:<12} |
{spacy.explain(token.pos_):<12} | {token.tag_:<12} | {spacy.explain(token.tag_)}')
```

- Part of Speech (POS) tagging assigns grammatical labels (e.g., noun, verb, adjective) to each word in a sentence.
- It enhances text analysis by providing syntactic structure and understanding word relationships.
- POS tagging helps in downstream NLP tasks like named entity recognition, parsing, and machine translation.
- Accurate POS tagging improves context understanding, disambiguates word meanings, and boosts NLP model performance.
- In Spacy, token.pos_can be used to retrieve the POS tag.

Stop Words:

Words like **to, for , a, over, from, the, had,...**

When should I not remove stop words

This is a good movie → **remove stop words** → Good movie

This is a not good movie → **remove stop words** → good movie

Chat bot, Q&A system, Language Translation, Any case where valuable information is lost

Named Entity Recognition (NER):

Use case # 1:

Regular Expressions (Regex):

Text Representation[Featuring Engineering]:

Introduction to Text Representation:

Feature Engineering is a process of extracting features from raw data.

Representing Text as a vector is also known as vector space model

Often in NLP, feeding a good text representation to an ordinary algorithm will get you much farther compared to applying a top-notch algorithm to an ordinary text representation

One Hot Encoding

Bag of Words

TFIDF

Word Embedding

Contextual Embedding

Label and One Hot Encoding:

One Hot Encoding: Put 1 on the one word and 0 on the remaining words [curse of dimensionality]

Similar words do not have similar encoding

Vector Size increase

Out of Vocabulary(OOV) problem

No fixed length representation

Label Encoding: Labeling the words based on order [Good Average Bad] → [1, 2, 3]

Can't apply this for the unordered values

🔍 **What is Bag of Words?**

The Bag of Words model represents text data (like sentences or documents) as a collection of words, disregarding grammar and word order but keeping multiplicity (i.e., how many times each word appears).

🧠 **How It Works**

1. **Vocabulary Creation:**
    - Collect all unique words from the entire text corpus.

- This set of words becomes your vocabulary.

2. **Vector Representation:**
    - Each document is represented as a vector of word counts.
    - The length of the vector equals the size of the vocabulary.
    - Each position in the vector corresponds to a word in the vocabulary and contains the frequency of that word in the document.

📦 Example

Suppose you have two sentences:

- Sentence 1: "I love NLP"
- Sentence 2: "NLP is fun"

Vocabulary: ["I", "love", "NLP", "is", "fun"]

Vectors:

- Sentence 1 → [1, 1, 1, 0, 0]
- Sentence 2 → [0, 0, 1, 1, 1]

✅ Advantages

- Simple and easy to implement.
- Works well for many basic NLP tasks like spam detection or sentiment analysis.

❌ Limitations

- Ignores word order and context.
- Can lead to high-dimensional and sparse vectors.
- Doesn't capture semantics (e.g., synonyms are treated as different words).

The Bag of N-grams model is an extension of the Bag of Words (BoW) model in NLP. It captures not just individual words, but also sequences of words, which helps preserve some context and word order.

🔍 What is an N-gram?

An N-gram is a contiguous sequence of N items (usually words) from a given text.

- Unigram: Single words (N=1)
- Bigram: Pairs of consecutive words (N=2)
- Trigram: Triplets of consecutive words (N=3)
- And so on...

🧠 How Bag of N-grams Works

1. Tokenize the text into N-grams.

2. Build a vocabulary of all unique N-grams in the corpus.
3. Represent each document as a vector of N-gram frequencies.

📦 Example

Text: "I love natural language processing"

- Unigrams: ["I", "love", "natural", "language", "processing"]
- Bigrams: ["I love", "love natural", "natural language", "language processing"]
- Trigrams: ["I love natural", "love natural language", "natural language processing"]

✅ Advantages

- Captures local context and word order better than BoW.
- Useful for tasks like text classification, language modeling, and machine **translation.**

❌ Limitations

- Still doesn't capture long-range dependencies.
- Vocabulary size grows rapidly with larger N.
- Can lead to sparse and high-dimensional feature spaces.
- Out of Vocabulary problem

TF-IDF stands for **Term Frequency–Inverse Document Frequency**. It's a statistical measure used in **Natural Language Processing (NLP)** to evaluate how important a word is to a document in a collection or corpus.

🔍 What Does TF-IDF Do?

It balances two things:

1. **Term Frequency (TF)** – How often a word appears in a document.
2. **Inverse Document Frequency (IDF)** – How unique or rare the word is across all documents.

🧠 Formula

For a word *t* in document *d*:

- $\boldsymbol{TF}$:

  $$TF(t,d) = \frac{Number\ of\ times\ t\ appears\ in\ d}{Total\ number\ of\ terms\ in\ d}$$

- $\boldsymbol{IDF}$:

  $$IDF(t) = log(\frac{N}{1 + DF(t)})$$

  Where:

  - $N = total\ number\ of\ documents$

- $DF(t) = number\ of\ documents\ containing\ the\ term\ t$

- ***TF-IDF:***

$$TF - IDF(t,d) = TF(t,d) \times IDF(t)$$

📦 **Example**

Suppose you have 3 documents:

1. "I love NLP"
2. "NLP is fun"
3. "I love fun"

The word **"NLP"** appears in 2 out of 3 documents, so its IDF will be lower than a word like **"love"**, which appears in only 2 documents.

TF-IDF helps **downweight common words** (like "is", "the") and **highlight unique, informative words**.

✅ **Advantages**

- Highlights **important words** in a document.
- Simple and effective for **text classification**, **search engines**, and **keyword extraction**.

❌ **Limitations**

- Doesn't capture **word order** or **context**.
- Still results in **sparse vectors**.
- Not ideal for **deep semantic understanding**.