JMAN GROUP

Data Science

Natural Language Processing NLP

Table of Contents

1 Introduction to NLP

1.1 What is NLP?

Natural Language Processing, or NLP, is the sub-field of AI that is focused on enabling computers to understand, interpret, and respond to human language in a way that is both meaningful and useful. It bridges the gap between human communication and machine understanding.

1.2 Why is NLP important?

- Large volumes of textual data:

  Natural language processing helps computers communicate with humans in their own language and scales other language-related tasks. For example, NLP makes it possible for computers to read text, hear speech, interpret it, measure sentiment and determine which parts are important. Today's machines can analyse more language-based data than humans, without fatigue and in a consistent, unbiased way. Considering the staggering amount of unstructured data that's generated every day, from medical records to social media, automation will be critical to fully analyse text and speech data efficiently.

- Structuring a highly unstructured data source

  Human language is astoundingly complex and diverse. We express ourselves in infinite ways, both verbally and in writing. Not only are there hundreds of languages and dialects, but within each language is a unique set of grammar and syntax rules, terms and slang. When we write, we often misspell or abbreviate words or omit punctuation. When we speak, we have regional accents, and we mumble, stutter and borrow terms from other languages. While supervised and unsupervised learning, and specifically deep learning, are now widely used for modelling human language, there's also a need for syntactic and semantic understanding and domain expertise that are not necessarily present in these machine learning approaches. NLP is important because it helps resolve ambiguity in language and adds useful numeric structure to the data for many downstream applications, such as speech recognition or text analytics.

- Insights from Text:

  Extracting meaningful insights from text data supports decision-making in various domains like healthcare, finance, and customer service.

1.3 Application of NLP?

- Text Classification
  - Spam Detection: Classify emails or messages as spam or not spam.
  - Sentiment Analysis: Identify the sentiment of product reviews, social media posts, or customer feedback (positive, negative, neutral).
  - Topic Classification: Categorize news articles or documents into predefined topics (e.g., politics, sports, technology).
  - Intent Detection: Determine the intent behind user queries in chatbots (e.g., inquiry, complaint, request).
- Text Generation
  - Chatbots/Virtual Assistants: Generate human-like responses for customer support or personal assistants.
  - Content Creation: Create summaries, product descriptions, or creative writing.
  - Code Generation: Generate programming code based on natural language input.

- Information Retrieval
  - Document Search: Build systems that retrieve the most relevant documents or information for a query.
  - FAQ Systems: Extract relevant answers to user questions from a database of FAQs.
  - Legal Document Retrieval: Identify and retrieve clauses or relevant information from contracts.
- Information Extraction
  - Named Entity Recognition (NER): Identify and extract entities like names, locations, dates, or organizations from text.
  - Key phrase Extraction: Extract main topics or keywords from documents.
  - Relationship Extraction: Identify relationships between entities (e.g., "John is the CEO of Company X").
- Machine Translation
  - Language Translation: Translate text from one language to another (e.g., English to French).
  - Code Translation: Convert code between programming languages (e.g., Python to Java).
- Summarization
  - Text Summarization: Create concise summaries of articles, reports, or long documents.
  - Financial Reporting: Summarize key financial metrics or highlights from detailed reports.
- Question Answering
  - Open-Domain QA: Answer any general question using large datasets like Wikipedia.
  - Domain-Specific QA: Provide answers in specific domains (e.g., medical, legal, or financial).
- Sentiment and Bias Analysis
  - Customer Feedback Analysis: Analyse customer sentiments to identify satisfaction trends.
  - Media Bias Analysis: Detect and remove biased language in news articles or reports.
  - Financial Materiality Analysis: Identify biased or material language in financial statements.
- Text-to-Speech and Speech-to-Text
  - Speech Recognition: Convert spoken words into text (e.g., transcribing meetings).
  - Voice Assistants: Generate spoken language responses from text.
- Document Processing
  - OCR (Optical Character Recognition): Extract text from scanned documents or images.
  - Invoice Processing: Extract and process fields like invoice numbers, dates, and amounts from invoices.
- Behavioural and Predictive Analytics
  - Churn Prediction: Analyse customer reviews or chat logs to predict the likelihood of churn.
  - Behaviour Analysis: Infer user behaviour patterns from their text interactions.

1.4 Key Challenges in NLP?

- Ambiguity: Words can have multiple meanings depending on context.
  Example: "I saw a bat" (bat = animal or a baseball bat).
- Cultural Nuances: Slang, idioms, and regional language variations.
- Language Diversity: Supporting multiple languages with different syntax and grammar rules.
- Data Quality: Dealing with noisy, unstructured, and incomplete text data.

1.5 How NLP Works?

Natural Language Processing (NLP) involves two main components:

- Linguistic Rules: Language rules for grammar, syntax, and semantics.

- Machine Learning (ML): Training models to recognize patterns and make predictions based on text data.

NLP works by combining various computational techniques to analyse, understand and generate human language in a way that machines can process. Key steps in processing natural language include:

1.5.1 Text Preprocessing

Text preprocessing transforms raw text into a clean, structured format for analysis.

- Tokenization splits text into smaller units like words or sentences.
- Lowercasing ensures consistency by converting all text to lowercase.
- Stop Word Removal filters out common, non-informative words (e.g., "is," "the").
- Stemming and Lemmatization reduce words to their base forms (e.g., "running" → "run").
- Text Cleaning removes punctuation, special characters, and numbers that clutter the analysis.

This process ensures standardized and noise-free input for models.

1.5.2 Feature Extraction

Feature extraction converts text into numerical representations for machine analysis:

- Bag of Words (BoW) and TF-IDF measure word occurrence and importance.
- Word Embeddings (e.g., Word2Vec, GloVe) represent words as dense vectors, capturing semantic relationships.
- Contextual Embeddings (e.g., BERT) provide nuanced representations by considering word context within sentences.

These techniques create structured data, enabling models to understand textual information.

1.5.3 Text Analysis

Text analysis interprets the processed data to extract insights:

- Part-of-Speech (POS) Tagging identifies grammatical roles.
- Named Entity Recognition (NER) highlights entities like names, places, or dates.
- Dependency Parsing examines grammatical relationships between words.
- Sentiment Analysis assesses emotional tone (positive, negative, neutral).
- Topic Modeling uncovers hidden themes across documents.
- Natural Language Understanding (NLU) focuses on comprehending meaning and intent.

These techniques transform unstructured text into actionable insights.

1.5.4 Model Training

Processed and extracted features train machine learning models. During training:

- Models learn patterns and relationships.
- Parameters are adjusted to minimize errors and improve performance.
- Validation and fine-tuning enhance accuracy and applicability to real-world tasks.

Trained models predict outcomes, classify text, or generate insights for unseen data.

1.5.5 Tools and Resources

Software tools streamline the NLP pipeline:

- NLTK: Preprocessing and text analysis.
- TensorFlow: Building and training machine learning models.

2 Basics

2.1 Tokenization

2.1.1 What is Tokenization?

Tokenization is the process of breaking down a text into smaller units called "tokens." Tokens can be words, phrases, or characters, depending on the level of analysis. It is a fundamental step in natural language processing, as it prepares text data for further analysis or model training.

2.1.2 Why is Tokenization important?

- Text Simplification: Converts complex text into manageable units for analysis.
- Input for NLP Models: Tokenized text serves as input for algorithms and models.
- Preserving Meaning: Helps retain the structure and meaning of the text.

2.1.3 Types of Tokenization:

- **Word Tokenization:** Splits text into individual words.
  - Example:
    Input: “I love NLP!”
    Output: ["I", "love", "NLP", "!"]
- **Sentence Tokenization:** Splits text into sentences.
  - Example:
    Input: "NLP is exciting. Tokenization is the first step!”
    Output: ["NLP is exciting.", "Tokenization is the first step!"]
- **Sub word Tokenization:** Breaks words into smaller meaningful units, used in modern NLP models.
  - Example:
    Input: "unbreakable”
    Output: ["un", "break", "able”]
- **Character Tokenization:** Splits text into individual characters.
  - Example:
    Input: “NLP”
    Output: ["N", "L", "P"]

2.1.4 Methods of Tokenization:

- **Rule-Based Tokenization:** Uses predefined rules like whitespace, punctuation, or regex patterns to split text.

```
text = "Tokenization is important!"
tokens = text.split()
print(tokens)
```

```
['Tokenization', 'is', 'important!']
```

- **Library-Based Tokenization:** Utilizes NLP libraries for more sophisticated tokenization.

```
#Example (Using nltk in Python):

from nltk.tokenize import word_tokenize
text = "Tokenization is important!"
tokens = word_tokenize(text)
print(tokens)
```

```
['Tokenization', 'is', 'important', '!']
```

- **Advanced Tokenization:** Employs algorithms like Byte Pair Encoding (BPE) or WordPiece for subword tokenization, commonly used in transformer-based models like BERT.

2.2 Stop Words Removal in NLP:

2.2.1 What are Stop Words?

Stop words are common words in a language that are often removed from text data during preprocessing because they do not add significant meaning to the analysis. Examples of stop words include "is," "the," "in," "and," etc.

2.2.2 Why Remove Stop Words?

- **Reduce Noise**: Stop words occur frequently and can overwhelm the analysis without adding value.
- **Improve Efficiency**: Eliminating stop words reduces the size of the text data, speeding up computations.
- **Focus on Meaningful Words**: By removing stop words, the model can focus on words that carry more semantic weight.

2.2.3 Common Stop Words

Here are some examples of stop words in English:

- Articles: a, an, the
- Prepositions: in, on, at
- Conjunctions: and, or, but
- Pronouns: he, she, it, they

2.2.4 Implementation of Stop Words Removal

- **Using a Predefined Stop Words List**

  Most NLP libraries provide a predefined list of stop words.

  Example (Python using nltk):

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

text = "This is an example sentence demonstrating stop word removal."
stop_words = set(stopwords.words('english'))
words = word_tokenize(text)
filtered_words = [word for word in words if word.lower() not in stop_words]
print(filtered_words)
```

```
['example', 'sentence', 'demonstrating', 'stop', 'word', 'removal', '.']
```

- **Custom Stop Words List**

  You can create your own list of stop words tailored to your specific use case.

```
custom_stop_words = {"example", "stop"}
filtered_words = [word for word in words if word.lower() not in custom_stop_words]
print(filtered_words)
```

```
['This', 'is', 'an', 'sentence', 'demonstrating', 'word', 'removal', '.']
```

2.2.5 Challenges in Removing Stop Words

- **Context Dependency:** Some stop words might be important in specific contexts (e.g., "not" in sentiment analysis).
- **Language Variability:** Stop words vary across languages and dialects, requiring multilingual support.
- **Domain-Specific Words:** Generic stop words list may not work well for specialized domains (e.g., medical or legal texts).

2.2.6 When to Retain Stop Words

- Retain stop words if their presence is meaningful for the task, such as:
  - Sentiment analysis (e.g., "not happy" has a different sentiment than "happy").
  - Language modelling, where all words contribute to predictions.

2.3 Stemming and Lemmatization in NLP:

Both stemming and lemmatization are text preprocessing techniques used to reduce words to their base or root forms. They are critical for standardizing words and improving the efficiency of NLP models.

2.3.1 Stemming

Stemming is the process of reducing a word to its root form by chopping off prefixes or suffixes. It uses heuristic rules rather than understanding the context or grammatical correctness.

Example:

Words: "playing," "played," "plays"

Stem: “play"

- **Popular Stemmers:**
  - Porter Stemmer
  - Lancaster Stemmer
  - Snowball Stemmer
- **Advantages:**
  - Simple and fast.
  - Reduces dimensionality of text data.
- **Disadvantages:**
  - May result in non-meaningful stems (e.g., "studies" → "studi").
  - Ignores the meaning or context of words.

```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
words = ["playing", "played", "plays"]
stems = [stemmer.stem(word) for word in words]
print(stems)
```

```
['play', 'play', 'play']
```

2.3.2 Lemmatization

Lemmatization is the process of reducing a word to its base or dictionary form (lemma) while ensuring the result is a valid word. It considers the word's context and part of speech (POS).
Example:

Words: "running," "ran"
Lemma: "run"

- Advantages:
  - Produces meaningful and grammatically correct words.
  - Better for context-sensitive NLP tasks.
- Disadvantages:
  - Slower than stemming.
  - Requires more computational resources.

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
words = ["running", "ran", "runs"]
lemmas = [lemmatizer.lemmatize(word, pos="v") for word in words]
print(lemmas)
```

```
['run', 'run', 'run']
```

3 Text Representation:

3.1 Bag of Words (BoW)

3.1.1 What is Bag of Words (BoW)?

Bag of words (BoW; also stylized as bag-of-words) is a feature extraction technique that models text data for processing in information retrieval and machine learning algorithms. More specifically, BoW models are an unstructured assortment of all the known words in a text document defined solely according to frequency while ignoring word order and context. Bag of words is one of several steps in many text mining pipelines.

Most natural language processing (NLP) packages come loaded with functions to create bag of words models, such as scikit-learn's CountVectorizer function

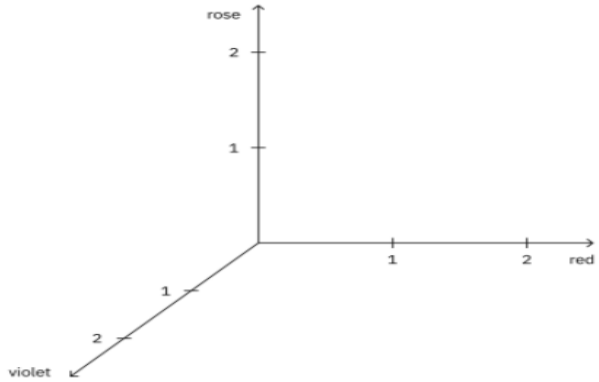
3.1.2 How bag of words models works

Bag of words (BoW) is a foundational text processing technique. It represents text as a vector in a multi-dimensional space, where each dimension corresponds to a unique word in the text set. The number of dimensions equals the number of unique words. Each text document is represented as a point in this space, and the position along each dimension is determined by the word's frequency in that document. BoW simplifies text into numerical data but requires understanding vector spaces to fully grasp its workings. Despite its simplicity, it plays a critical role in text analysis.

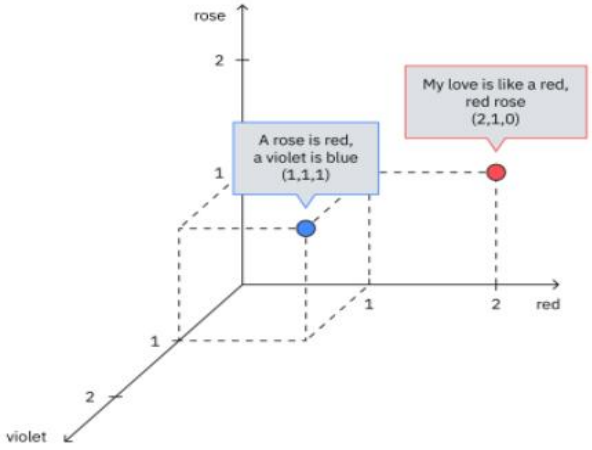
For example, assume we have a text set in which the contents of two separate documents are respectively:

- Document 1: A rose is red, a violet is blue
- Document 2: My love is like a red, red rose

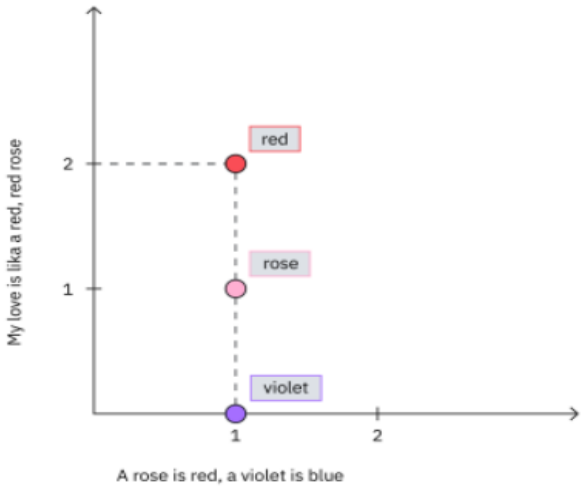
Because it is difficult to imagine anything beyond a three-dimensional space, we will limit ourselves to just that. A vector space for a corpus containing these two documents would have separate dimensions for red, rose, and violet. A three-dimensional vector space for these words may look like:



Since red, rose, and violet all occur once in Document 1, the vector for that document in this space will be (1,1,1). In Document 2, red appears twice, rose once, and violet not at all. Thus, the vector point for Document 2 is (2,1,0). Both of these document-points will be mapped in the three-dimensional vector space as:



Note that this figure visualizes text documents as data vectors in a three-dimensional feature space. But bag of words can also represent words as feature vectors in a data space. A feature vector signifies the value (occurrence) of a given feature (word) in a specific data point (document). So the feature vectors for red, rose, and violet in Documents 1 and 2 would look like:



Note that the order of words in the original documents is irrelevant. For a bag of words model, all that matters is each word's number of occurrences across the text set.

3.1.3 Why use bag of words models

Because bag of words models only quantify the frequency of words in a given document, bag of words is often described as a simple modelling technique. But bag of words assists in many NLP tasks, most notably document classification. Indeed, literature often discusses bag of words alongside statistical classifiers like Naïve Bayes.

Text classification tasks interpret those words with high frequency in a document as representing the document's main ideas. This is not an unreasonable assumption. For example, if some of the most frequent words in a document are president, voters, and election, there is a high probability the document is a political text, specifically discussing a presidential election. Text classification with bag of words then extrapolates that documents with similar content are similar in type.

3.1.4 Steps in BoW

- Text Preprocessing:
  - Tokenize the text.
  - Convert to lowercase.
  - Remove punctuation and stop words (optional).
- Build Vocabulary: Identify all unique words in the dataset.
- Vectorize: Represent each text as a vector where each dimension corresponds to a word in the vocabulary.

```
from sklearn.feature_extraction.text import CountVectorizer

documents = ["I love NLP.", "NLP is amazing."]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(documents)

print("These are the 2 sentences:", documents)
print("Vocabulary:", vectorizer.get_feature_names_out())
print("BoW Representation:\n", X.toarray())
```

```
These are the 2 sentences: ['I love NLP.', 'NLP is amazing.']
Vocabulary: ['amazing' 'is' 'love' 'nlp']
BoW Representation:
 [[0 0 1 1]
 [1 1 0 1]]
```

3.1.5 Advantages of BoW

- Simple and intuitive.
- Effective for text classification tasks like spam detection or sentiment analysis.
- Easy to implement and works well for small datasets.

3.1.6 Limitations of BoW

- Word Correlation: Bag of Words (BoW) assumes words are independent, ignoring correlations. For instance, "election" is more likely to co-occur with "president" than "poet," but BoW treats them as unrelated. This can lead to multicollinearity in models, though Naïve Bayes often handles this well.
- Compound Words: BoW fails to recognize compound phrases (e.g., "Mr. Darcy") as semantic units, treating them as separate words and losing contextual meaning.

- Polysemous Words: Words with multiple meanings (e.g., "bat" as an animal or sports equipment) are collapsed into a single feature, ignoring context and semantic distinctions.
- Sparsity: BoW creates high-dimensional vectors where most values are zero, leading to sparse matrices and risks of overfitting.

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

TF-IDF (Term Frequency-Inverse Document Frequency) is a metric that measures the importance of a word in a document relative to a collection of documents. It is widely used in information retrieval, text classification, search engine and machine learning to:

- Evaluate document relevance to a search query.
- Quantify word or phrase importance in a document.
- Highlight useful terms while down weighting common ones.

3.2.1 How does the TF-IDF model works

TF-IDF consists of two main components:

- Calculation:
  TF-IDF is the product of:
  - Term Frequency (TF): Frequency of the word in the document.

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

  - Inverse Document Frequency (IDF): Weighting that reduces the importance of common terms across the document collection.

$$\text{IDF}(t) = \log\left(\frac{\text{Total number of documents in corpus}}{\text{Number of documents containing the term } t}\right)$$

  - TF-IDF Score:

$$TF - IDF = TF \times IDF$$

  - Words that appear frequently in one document but are rare in the overall corpus get higher scores.
  - Common words like "the," "is," and "and" receive lower scores since they appear in many documents.

3.2.2 Why Use the TF-IDF Model?

- Unlike Bag of Words (BoW), TF-IDF assigns importance to words rather than just counting occurrences.
- It helps filter out commonly occurring words (stop words) while emphasizing distinguishing words.
- Used in document ranking, keyword extraction, text classification, and search engines

3.2.3 Steps in TF-IDF Calculation

- Tokenization: Split text into words.
- Compute Term Frequency (TF) for each word in each document.
- Compute Inverse Document Frequency (IDF) for each word across all documents.
- Calculate TF-IDF Score for each word by multiplying TF and IDF.
- Create a TF-IDF Matrix where each document is represented as a vector.

3.2.4 Advantages of TF-IDF

- Assigns importance to words, reducing the impact of frequently occurring but non-informative words.
- Captures important words for document classification, search engines, and recommendation systems.
- Simple and effective for many NLP tasks.
- Does not require large datasets to be effective.

3.2.5 Limitations of TF-IDF

- Ignores the position and sequence of words (like BoW).
- Does not capture word meanings or synonyms (e.g., "car" and "automobile" are treated differently).
- Struggles with polysemy (same word, different meanings).
- Computationally expensive for large datasets.