Natural Language Processing  
Week # 10

The Islamia University of Bahawalpur  
Department of Software Engineering

Bahawalnagar Campus

Natural Language Processing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and human language. It involves the development of algorithms and models that enable machines to understand, interpret, and generate human-like text. NLP plays a crucial role in various applications, such as language translation, sentiment analysis, and chatbots, revolutionizing the way we interact with technology by bridging the gap between human communication and machine understanding.

Sunawar Khan

Table of Contents

[Introduction 1](#_Toc166664258)

[Integration of Computer Science: 1](#_Toc166664259)

[Incorporation of Artificial Intelligence: 2](#_Toc166664260)

[Utilization of Human Language: 2](#_Toc166664261)

[History 3](#_Toc166664262)

[1940-1960: - Focused on Machine Translation (MT) 3](#_Toc166664263)

[1948: - Pioneering the Path of NLP at Birkbeck College, London 3](#_Toc166664264)

[1950s: - Chomsky's Theory of Generative Language - Bridging Linguistics and CS 3](#_Toc166664265)

[1957: - Chomsky idea of Generative Grammar 4](#_Toc166664266)

[1960: - Evolution of Linguistic Frameworks – Case Grammar 4](#_Toc166664267)

[1960: - SHRDLU 4](#_Toc166664268)

[LUNAR: Advancing Natural Language Database Interfaces 4](#_Toc166664269)

[Evolution of NLP: Pre- and Post-1980s 5](#_Toc166664270)

[The Synergy of Modern NLP Applications in Artificial Intelligence 5](#_Toc166664271)

[5 Stages to Process NLP 6](#_Toc166664272)

[Lexical Analysis: 7](#_Toc166664273)

[Syntactic Analysis: 7](#_Toc166664274)

[Semantic Analysis: 7](#_Toc166664275)

[Discourse Integration: 7](#_Toc166664276)

[Pragmatic Analysis: 7](#_Toc166664277)

[Components of NLP 7](#_Toc166664278)

[Natural Language Understanding (NLU) 7](#_Toc166664279)

[Mapping Input into Useful Representations: 8](#_Toc166664280)

[Analyzing Different Aspects of Language: 8](#_Toc166664281)

[Components of NLU 8](#_Toc166664282)

[ Intent Recognition: 8](#_Toc166664283)

[Ambiguities in NLU 8](#_Toc166664284)

[Natural Language Generation (NLG): 9](#_Toc166664285)

[Text Planning: 9](#_Toc166664286)

[Sentence Planning: 10](#_Toc166664287)

[Text Realization: 10](#_Toc166664288)

[Components of NLG 10](#_Toc166664289)

[ Text Summarization: 10](#_Toc166664290)

[ Dialogue Generation: 10](#_Toc166664291)

[ Text-to-Speech (TTS): 10](#_Toc166664292)

[ Content Creation: 10](#_Toc166664293)

[NLP API’s 10](#_Toc166664294)

[Google Cloud Natural Language API: 10](#_Toc166664295)

[IBM Watson Natural Language Understanding: 11](#_Toc166664296)

[Microsoft Azure Text Analytics: 11](#_Toc166664297)

[Amazon Comprehend: 11](#_Toc166664298)

[Aylien Text Analysis API: 11](#_Toc166664299)

[MeaningCloud: 11](#_Toc166664300)

[MonkeyLearn: 11](#_Toc166664301)

[NLP Cloud: 11](#_Toc166664302)

[NLP Libraries in Python 12](#_Toc166664303)

[NLTK (Natural Language Toolkit): 12](#_Toc166664304)

[spaCy: 12](#_Toc166664305)

[Gensim: 12](#_Toc166664306)

[TextBlob: 12](#_Toc166664307)

[TensorFlow NLP: 12](#_Toc166664308)

[Applications of Natural Language Processing 13](#_Toc166664309)

[Question Answering 13](#_Toc166664310)

[Key components of QA systems 13](#_Toc166664311)

[Information Retrieval: 13](#_Toc166664312)

[Answer Extraction: 14](#_Toc166664313)

[Answer Generation: 14](#_Toc166664314)

[Spam Detection 14](#_Toc166664315)

[Key Components of Spam Detection 15](#_Toc166664316)

[Feature Extraction: 15](#_Toc166664317)

[Training Data Preparation: 15](#_Toc166664318)

[Machine Learning Models: 15](#_Toc166664319)

[Model Evaluation: 15](#_Toc166664320)

[Thresholding and Filtering 15](#_Toc166664321)

[Feedback Loop 15](#_Toc166664322)

[Sentiment Analysis 15](#_Toc166664323)

[Spelling Correction 17](#_Toc166664324)

[Speech Recognition 17](#_Toc166664325)

[Chatbots 18](#_Toc166664326)

[NLP Pipeline 19](#_Toc166664327)

[Data Collection: 19](#_Toc166664328)

[Preprocessing 19](#_Toc166664329)

[Tokenization: 19](#_Toc166664330)

[Lowercasing: 19](#_Toc166664331)

[Stopword Removal: 19](#_Toc166664332)

[*From Sample Sentence* 20](#_Toc166664333)

[Stemming/Lemmatization: 20](#_Toc166664334)

[*Stemming Example* 20](#_Toc166664335)

[*Lemmatization Example* 20](#_Toc166664336)

[Normalization: 20](#_Toc166664337)

[Noise Removal: 21](#_Toc166664338)

[Feature Engineering: 21](#_Toc166664339)

[Bag of Words (BoW): 21](#_Toc166664340)

[TF-IDF (Term Frequency-Inverse Document Frequency): 22](#_Toc166664341)

[*Sample Code* 24](#_Toc166664342)

[Word Embeddings: 25](#_Toc166664343)

[Setting up the NLP Environment 27](#_Toc166664344)

[Installing Essential Python Libraries 27](#_Toc166664345)

[Downloading Language Models and Datasets 27](#_Toc166664346)

[Accessing Pre-trained Models for Various NLP Tasks: 27](#_Toc166664347)

[**Text** **Preprocessing** 28](#_Toc166664348)

[Cleaning and Tokenizing Text Data 28](#_Toc166664349)

[Output 29](#_Toc166664350)

[Removing Stop Words and Punctuation 29](#_Toc166664351)

[Output 30](#_Toc166664352)

[Stemming and Lemmatization 30](#_Toc166664353)

[Output 31](#_Toc166664354)

[Text Analysis Techniques 31](#_Toc166664355)

[Frequency Analysis and Visualization 31](#_Toc166664356)

[Part-of-Speech Tagging 33](#_Toc166664357)

[Output 33](#_Toc166664358)

[Counting POS Tags–Chunking 33](#_Toc166664359)

[Example 34](#_Toc166664360)

[Output 34](#_Toc166664361)

[Named Entity Recognition (NER) 34](#_Toc166664362)

[Output 35](#_Toc166664363)

[Sentiment Analysis Unveiled 35](#_Toc166664364)

[Understanding Sentiment Analysis 35](#_Toc166664365)

[Significance in Social Media and Customer Feedback: 35](#_Toc166664366)

[Challenges and Nuances in Sentiment Analysis: 36](#_Toc166664367)

[Implementing Sentiment Analysis using TextBlob 36](#_Toc166664368)

[Output 36](#_Toc166664369)

[Language Translation Unleashed 37](#_Toc166664370)

[Introduction to Language Translation 37](#_Toc166664371)

[Power of Language Translation in Global Communication: 37](#_Toc166664372)

[Real-world Examples of Translation Technologies: 37](#_Toc166664373)

[Implementing a Basic Language Translation Model using Transformers 37](#_Toc166664374)

[Output 38](#_Toc166664375)

[Text Classification Mastery 38](#_Toc166664376)

[Overview of Text Classification 38](#_Toc166664377)

[Defining Text Classification and Its Applications: 38](#_Toc166664378)

[Business Use Cases: Spam Detection, Sentiment Classification, and More: 38](#_Toc166664379)

[Building a Simple Text Classifier using Machine Learning 38](#_Toc166664380)

[Practical Implementation with Naive Bayes and SVM: 38](#_Toc166664381)

[Output 40](#_Toc166664382)

[Summary 40](#_Toc166664383)

Natural Language Processing

# Introduction

Natural Language Processing (NLP) is a multidisciplinary field at the intersection of linguistics, computer science, artificial intelligence, and cognitive psychology. It involves the study and development of algorithms and techniques that enable computers to understand, interpret, and generate human language in a manner that is both meaningful and contextually appropriate.

Natural Language Processing (NLP) is a discipline within the intersection of Computer Science, Human language, and Artificial Intelligence (AI). It encompasses technologies that enable machines to comprehend, analyze, manipulate, and interpret human languages. By leveraging NLP, developers can organize knowledge to facilitate various tasks including translation, automatic summarization, Named Entity Recognition (NER), speech recognition, relationship extraction, and topic segmentation.

At its core, NLP enables intelligent systems to interact with humans using natural languages such as English. This technology becomes indispensable when we seek to imbue machines, like robots or clinical expert systems, with the ability to understand and execute tasks based on human instructions or when we require decision-making capabilities from dialogue-based systems.

The domain of NLP revolves around empowering computers to perform practical tasks with the natural languages humans employ. Inputs to an NLP system can take the form of spoken words or written text, while outputs may include synthesized speech, written responses, or actions performed by the system based on the processed input.

The field of NLP involves making computers to perform useful tasks with the natural languages  
humans use. The input and output of an NLP system can be:

* Speech
* Written Text

A company, let's call it "ZaryabCo," operates an online platform offering various tech products and services. To enhance customer satisfaction and streamline support processes, ZaryabCo decides to implement a virtual assistant for customer support.

## Integration of Computer Science:

Computer science plays a crucial role in the development of the virtual assistant. Software engineers utilize algorithms and data structures to design the underlying architecture of the system. They build the infrastructure for processing user queries, storing knowledge bases, and managing the interaction flow.

## Incorporation of Artificial Intelligence:

Artificial intelligence (AI) technologies, particularly NLP, are integrated into the virtual assistant to enable natural language understanding and generation. Machine learning models, trained on large datasets of customer interactions, are deployed to recognize intents and extract relevant information from user queries. These models use techniques like text classification, named entity recognition, and sentiment analysis to understand the context and sentiment of user messages.

## Utilization of Human Language:

The virtual assistant interacts with users using natural language, allowing customers to communicate with the system as they would with a human support agent. Users can ask questions, seek assistance with product issues, or request information about services using their own words and phrases. The virtual assistant analyzes these queries, interprets the user's intent, and provides appropriate responses or actions.

So, Artificial Intelligence (AI), Computer Science, and Human Language intersect and work together in the field of Natural Language Processing (NLP) to enable machines to understand, process, and generate human language in a meaningful way.

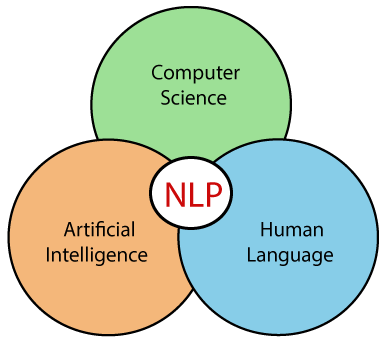


Figure 1:- Computer Science and Artificial Intelligence Relates to Natural Language Processing

# History

## 1940-1960: - Focused on Machine Translation (MT)

During the 1940s to the 1960s, the nascent field of Natural Language Processing (NLP) was primarily focused on the ambitious task of Machine Translation (MT). This period marked the early exploration of using computers to automate the translation of human languages, spurred in part by the urgency of World War II and the need for cryptographic analysis. One notable milestone during this era was the Georgetown-IBM Experiment in 1954, which showcased early attempts at electronic translation from Russian to English. However, early machine translation systems relied heavily on rule-based approaches, which struggled to handle the complexity and ambiguity inherent in natural language. Despite these challenges, this period laid the groundwork for computational linguistics as a discipline, with researchers beginning to delve into the underlying structure and grammar of human languages. While progress in machine translation was slow and initial results often fell short of expectations, the foundational work conducted during this time set the stage for future advancements in NLP, paving the way for the sophisticated language technologies that would emerge in subsequent decades.

## 1948: - Pioneering the Path of NLP at Birkbeck College, London

In 1948, a significant milestone in the history of Natural Language Processing (NLP) occurred with the introduction of the first recognizable NLP application at Birkbeck College in London. While specific details about this application are limited, its introduction marked an important early instance of leveraging computational techniques to process and analyze human language. This event underscores the growing interest and experimentation with NLP during this period, laying the groundwork for future advancements in the field. Despite the rudimentary nature of early NLP applications, this pioneering effort at Birkbeck College foreshadowed the transformative impact that NLP would have on various domains, including linguistics, artificial intelligence, and information technology, in the decades to come.

## 1950s: - Chomsky's Theory of Generative Language - Bridging Linguistics and CS

During the 1950s, a significant divergence emerged between the fields of linguistics and computer science regarding the nature of language. This period saw conflicting views on how best to approach the computational analysis of human language. Meanwhile, in 1957, Noam Chomsky, a prominent linguist, published his seminal work "Syntactic Structures." In this influential book, Chomsky introduced the concept of generative grammar and argued that language is inherently generative in nature. Chomsky's theory posited that human language is not merely a collection of static rules but rather a system capable of generating an infinite variety of grammatically correct sentences. This notion challenged prevailing behaviorist views of language acquisition and laid the foundation for the development of formal language theory and computational linguistics. Chomsky's insights bridged the gap between linguistics and computer science, providing a theoretical framework that would inform the design of early natural language processing systems and shape the direction of research in the field for decades to come.

1957: - Chomsky idea of Generative Grammar

In 1957, Noam Chomsky made a groundbreaking contribution to linguistics with the introduction of Generative Grammar. This concept represented a departure from previous approaches to understanding language structure and usage. Chomsky proposed that language could be described using a set of rules that generate syntactic structures, rather than simply observing patterns of usage. Generative Grammar provided a systematic framework for analyzing the underlying structure of sentences, emphasizing the role of rules and principles in generating grammatically correct sentences. By focusing on the rules governing the formation of sentences, Chomsky's theory offered deeper insights into the innate structure of human language. This approach paved the way for advancements in computational linguistics and natural language processing, as it provided a formal foundation for building language models and processing algorithms. Chomsky's work laid the groundwork for further developments in linguistic theory and had a profound impact on the study of language across various disciplines.

## 1960: - Evolution of Linguistic Frameworks – Case Grammar

During the period spanning from the 1960s to the 1980s, significant advancements were made in linguistic frameworks, with Case Grammar emerging as a notable development. Linguist Charles J. Fillmore introduced Case Grammar in 1968, revolutionizing the understanding of the relationship between nouns and verbs in languages like English. This approach employed prepositions to express the connections between different elements of a sentence. In Case Grammar, case roles were defined to delineate the roles of entities such as agents, themes, and instruments in relation to specific verbs. For instance, in the sentence "Neha broke the mirror with the hammer," Case Grammar identifies Neha as the agent, the mirror as the theme, and the hammer as the instrument. This period witnessed a flourishing of linguistic theories and frameworks, laying the groundwork for further exploration and refinement in the study of language structure and usage.

## 1960: - SHRDLU

In the late 1960s and early 1970s, computer scientist Terry Winograd developed SHRDLU, a groundbreaking program that revolutionized the field of Natural Language Understanding (NLU). SHRDLU enabled users to communicate with computers in natural language and interactively manipulate objects in a virtual world. Users could issue commands such as "pick up the green ball" and ask questions like "What is inside the black box," with SHRDLU executing the requested actions and providing accurate responses. This pioneering program showcased the integration of syntax, semantics, and world knowledge to create a system capable of understanding and reasoning about natural language commands. SHRDLU's significance lies in its demonstration of how linguistic principles and computational techniques can be combined to build intelligent systems with the ability to comprehend human language and interact with the world.

## LUNAR: Advancing Natural Language Database Interfaces

LUNAR stands as a classic exemplar of Natural Language Database Interface systems, embodying the synergy of Augmented Transition Networks (ATNs) and Woods' Procedural Semantics. Developed as a pioneering system, LUNAR showcased remarkable proficiency in translating complex natural language expressions into database queries. With an impressive success rate, handling 78% of requests without errors, LUNAR demonstrated significant strides in bridging the gap between human language and database interaction. By integrating ATNs for language parsing and procedural semantics for query generation, LUNAR underscored the potential of computational linguistics in facilitating seamless communication between users and databases. Its legacy persists as a testament to the transformative impact of linguistic theories and computational techniques in the development of intelligent systems for natural language processing.

## Evolution of NLP: Pre- and Post-1980s

Before 1980, Natural Language Processing (NLP) systems relied heavily on intricate sets of hand-written rules to analyze and process language. However, following this period, a significant shift occurred as NLP began incorporating machine learning algorithms for language processing. By leveraging the power of machine learning, NLP systems gained the capability to learn from data and adapt to various linguistic patterns and complexities.

The early 1990s marked a period of accelerated growth for NLP, particularly in achieving higher levels of accuracy, notably in English grammar processing. This advancement was facilitated by the introduction of electronic texts, providing valuable resources for training and evaluating natural language programs. Additionally, the proliferation of computers equipped with faster CPUs and expanded memory capacities further fueled the progress of NLP technologies.

However, perhaps the most pivotal factor driving the advancement of natural language processing was the emergence and widespread adoption of the Internet. The vast amounts of text data available on the web provided unparalleled opportunities for training and testing NLP algorithms. This wealth of online information enabled NLP systems to become more robust, accurate, and capable of handling real-world language tasks with increasing proficiency.

## The Synergy of Modern NLP Applications in Artificial Intelligence

Modern Natural Language Processing (NLP) encompasses a diverse array of applications, including speech recognition, machine translation, and machine text reading. When integrated, these applications enable artificial intelligence systems to acquire knowledge of the world and interact with users in natural language. Amazon Alexa serves as a prime example of this integration, providing users with a seamless conversational experience.

By leveraging advanced speech recognition technology, Alexa can accurately transcribe spoken words into text, enabling users to communicate with the system using natural language commands and queries. Machine translation capabilities further enhance Alexa's functionality, allowing it to understand and respond to requests in multiple languages.

Moreover, Alexa's machine text reading capabilities enable it to access and comprehend vast amounts of textual information, ranging from news articles to user manuals. This enables Alexa to provide informative responses to user inquiries and assist with a wide range of tasks.

By combining these NLP applications, Amazon Alexa embodies the power of artificial intelligence to understand and interact with the world in a human-like manner. Whether it's answering questions, providing recommendations, or controlling smart home devices, Alexa's integration of modern NLP technologies exemplifies the transformative potential of AI-driven natural language processing in enhancing user experiences and facilitating seamless human-machine interaction.

# 5 Stages to Process NLP

The five steps you've mentioned correspond to different stages in the process of Natural Language Understanding (NLU), which is aimed at enabling computers to comprehend and interpret human language. Let's break down each step:

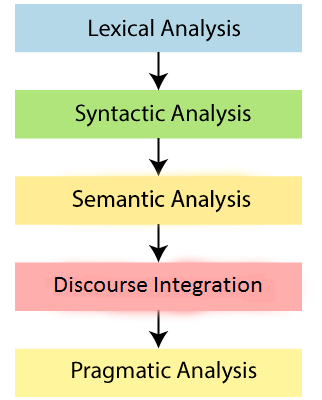


Figure 2:- 5 Stages to Process NLP

Lexical Analysis: This initial step involves breaking down the input text into individual tokens or words, known as lexemes. Lexical analysis identifies the basic units of meaning in the text and may involve tasks such as tokenization, stemming, and lemmatization to normalize and standardize the words. It involves analysis of words in the sentence for grammar and arranging words in a manner that shows the relationship among the words. The sentence such as “The school goes to boy” is rejected by English syntactic analyzer.

Syntactic Analysis: Also known as parsing, syntactic analysis involves analyzing the grammatical structure of sentences to determine how words relate to each other. This step uses formal rules of syntax to parse the text and construct parse trees or syntactic structures that represent the relationships between words and phrases in the sentence.

Semantic Analysis: Semantic analysis focuses on understanding the meaning of words, phrases, and sentences within their context. This step involves mapping the syntactic structures generated in the previous step to their corresponding semantic representations, which capture the intended meaning of the text. Semantic analysis may include tasks such as semantic role labeling, named entity recognition, and word sense disambiguation. The semantic analyzer disregards sentence such as “hot ice-cream”.

Discourse Integration: Discourse integration deals with understanding the larger context and structure of a conversation or text, including relationships between sentences, coherence, and discourse coherence. This step involves integrating the meanings of individual sentences into a cohesive interpretation of the entire discourse, taking into account discourse markers, discourse structure, and discourse relations.

Pragmatic Analysis: Pragmatic analysis considers the context in which language is used, including speaker intentions, implicatures, and conversational implicatures. This final step involves interpreting the meaning of language in light of contextual factors such as social norms, cultural conventions, and the communicative goals of the speaker or writer.

Together, these five steps encompass the process of NLU, enabling computers to understand, interpret, and respond to human language input in a manner that is contextually appropriate and semantically meaningful. Each step plays a crucial role in bridging the gap between human language and machine understanding, facilitating a wide range of applications in areas such as natural language processing, information retrieval, dialogue systems, and machine translation.

# Components of NLP

## Natural Language Understanding (NLU)

NLU involves the process of extracting meaning and intent from human language input, whether it's text, speech, or any other form of communication.

Natural Language Understanding (NLU) is a crucial component of Natural Language Processing (NLP) that enables machines to comprehend and analyze human language input. NLU helps machines extract metadata from content, such as concepts, entities, keywords, emotions, relations, and semantic roles, to gain a deeper understanding of the meaning and context of the text.

In business applications, NLU plays a vital role in understanding customers' problems expressed in both spoken and written language. By accurately interpreting customer queries, feedback, or requests, businesses can effectively address customer needs and enhance their overall experience.

**The process of NLU involves several key tasks:**

Mapping Input into Useful Representations: NLU begins by mapping the given input in natural language, whether it's spoken or written, into structured representations that machines can understand and process. This often involves tasks like tokenization, part-of-speech tagging, and syntactic parsing to break down the input into meaningful units and analyze its grammatical structure.

Analyzing Different Aspects of Language: Once the input is mapped into useful representations, NLU analyzes various aspects of the language to extract relevant metadata. This includes identifying concepts and entities mentioned in the text, detecting keywords or key phrases that convey important information, analyzing the emotional tone or sentiment expressed in the text, identifying relationships between entities, and assigning semantic roles to different elements of the text (e.g., agent, patient, recipient).

By performing these tasks, NLU enables machines to understand the nuances and subtleties of human language, allowing them to interpret user input accurately and extract valuable insights from text data. In business applications, NLU empowers organizations to automate tasks like customer support, sentiment analysis, market research, and information retrieval, ultimately improving efficiency, productivity, and customer satisfaction.

### Components of NLU

* Speech Recognition: Converting spoken language into text.
* Text Tokenization: Breaking down text into smaller units, such as words or phrases.
* Part-of-Speech Tagging: Assigning grammatical labels (e.g., noun, verb, adjective) to tokens.
* Named Entity Recognition (NER): Identifying and categorizing named entities such as people, organizations, and locations.
* Semantic Role Labeling (SRL): Identifying the roles of words or phrases in a sentence, such as agent, patient, or recipient.
* Intent Recognition: Understanding the purpose or goal behind a user's input, often in the context of conversational agents or chatbots.

### Ambiguities in NLU

Ambiguities in Natural Language Understanding (NLU) arise from the inherent complexity, variability, and ambiguity of human language. Some common types of ambiguities in NLU include:

Lexical Ambiguity: Lexical ambiguity occurs when a word has multiple meanings or interpretations depending on context. For example, the word "bank" can refer to a financial institution or the edge of a river.

Syntactic Ambiguity: Syntactic ambiguity arises when a sentence can be parsed in multiple ways, leading to different interpretations of its meaning. For instance, in the sentence "I saw the man with the telescope," it is unclear whether the speaker used the telescope or the man being observed had one.

Semantic Ambiguity: Semantic ambiguity occurs when a word or phrase has multiple interpretations or senses that are contextually relevant. For example, the word "bat" can refer to a flying mammal or a piece of sports equipment used in baseball.

Referential Ambiguity: Referential ambiguity arises when it is unclear which entity or concept a pronoun or noun phrase refers to. For instance, in the sentence "She gave the book to her," it is unclear who "her" refers to without additional context.

Anaphoric Ambiguity: Anaphoric ambiguity occurs when a pronoun or noun phrase refers to multiple antecedents within a discourse, making it challenging to determine the intended referent. For example, in the sentence "John told Mary that he was leaving," it is unclear whether "he" refers to John or Mary.

Contextual Ambiguity: Contextual ambiguity arises when the meaning of a word or phrase depends on the broader context in which it is used. For example, the word "hot" can refer to temperature, attractiveness, or popularity depending on the context.

Addressing ambiguities in NLU requires sophisticated algorithms and techniques that consider contextual clues, syntactic structures, semantic relationships, and world knowledge to accurately interpret and understand natural language input. Despite the challenges posed by ambiguities, advances in NLU technologies continue to improve the accuracy and effectiveness of natural language understanding systems.

## Natural Language Generation (NLG):

NLG involves the process of generating human-like language output based on underlying data, knowledge, or instructions.

Natural Language Generation (NLG) is the process of generating coherent and meaningful phrases and sentences in natural language from some internal representation or structured data. NLG involves several key steps:

Text Planning: In this phase, NLG systems retrieve relevant content or information from a knowledge base or data source. This could involve selecting data points, facts, or concepts that are relevant to the message being conveyed.

Sentence Planning: Once the relevant content is identified, NLG systems move on to sentence planning. This step involves choosing the appropriate words, forming meaningful phrases, and structuring the tone of the sentence to effectively communicate the intended message. Sentence planning also considers factors such as grammar, syntax, and style to ensure that the generated text is coherent and natural-sounding.

Text Realization: The final step in NLG is text realization, where the sentence plan is mapped into a complete sentence structure. This involves arranging the chosen words and phrases into grammatically correct sentences, considering factors such as word order, tense, and agreement.

## Components of NLG

* Text Summarization: Condensing large amounts of text into shorter, more concise summaries while preserving key information.
* Language Translation: Converting text from one language to another while preserving meaning and context.
* Dialogue Generation: Generating natural language responses in conversational systems or chatbots based on user input and context.
* Text-to-Speech (TTS): Converting written text into spoken language, often with synthesized voices.
* Content Creation: Automatically generating content such as product descriptions, news articles, or reports based on input data or templates.

In essence, NLU focuses on understanding and interpreting human language input, while NLG focuses on generating human-like language output. Together, these components enable computers to interact with users in a more natural and meaningful way, powering a wide range of applications in fields such as virtual assistants, customer service, information retrieval, and content generation.

It's worth noting that while NLG is a complex task, Natural Language Understanding (NLU) is generally considered to be more challenging. NLU involves the process of extracting meaning and understanding from natural language input, which can be inherently ambiguous and context-dependent. NLG, on the other hand, involves generating language output based on a structured representation, which may be more constrained and predictable.

# NLP API’s

There are several Natural Language Processing (NLP) APIs available that provide pre-built tools and services for developers to integrate NLP functionality into their applications. These APIs offer a range of features, including text analysis, sentiment analysis, entity recognition, language translation, and more. Here are some popular NLP APIs:

Google Cloud Natural Language API: Google's API offers a suite of NLP capabilities, including entity recognition, sentiment analysis, and syntax analysis. It enables developers to extract valuable insights from text data, such as identifying entities mentioned in the text, analyzing the emotional tone conveyed, and parsing the grammatical structure of sentences.

IBM Watson Natural Language Understanding: IBM's Watson NLU API provides advanced NLP capabilities, such as entity extraction, sentiment analysis, emotion analysis, and concept tagging. It empowers developers to derive deeper insights from text data by uncovering entities, sentiments, and themes present in the content, enabling applications ranging from customer feedback analysis to content recommendation systems.

Microsoft Azure Text Analytics: Microsoft's Text Analytics API offers sentiment analysis, key phrase extraction, language detection, and entity recognition. It provides developers with powerful tools for understanding and analyzing text data, allowing them to gain insights into customer sentiment, identify important keywords, and extract valuable information from unstructured text sources.

Amazon Comprehend: Amazon's Comprehend API offers entity recognition, sentiment analysis, key phrase extraction, and language detection. It equips developers with the tools needed to analyze and understand text at scale, enabling applications such as customer feedback analysis, content categorization, and trend detection from large volumes of text data.

Aylien Text Analysis API: Aylien's API provides NLP features including entity recognition, sentiment analysis, language detection, and summarization. It offers developers a comprehensive suite of tools for extracting insights from text data, enabling applications such as content recommendation, trend analysis, and summarization of news articles or documents.

MeaningCloud: MeaningCloud offers text analytics functionalities such as sentiment analysis, entity extraction, and text classification. With MeaningCloud, developers can gain valuable insights from text data by analyzing the sentiment conveyed, identifying important entities mentioned, and categorizing text into relevant topics or categories.

MonkeyLearn: MonkeyLearn provides text analysis tools for tasks like sentiment analysis, topic classification, and named entity recognition. By leveraging MonkeyLearn's APIs, developers can analyze text data to understand the sentiment expressed, classify documents into different topics or categories, and identify and extract named entities such as people, organizations, and locations.

NLP Cloud: NLP Cloud offers a suite of NLP APIs including sentiment analysis, text classification, and named entity recognition. Developers can use NLP Cloud to analyze and process text data, extracting insights such as sentiment polarity, categorizing text into predefined classes or categories, and identifying named entities mentioned in the text.

# NLP Libraries in Python

## NLTK (Natural Language Toolkit):

Explanation: NLTK is a comprehensive library for NLP tasks such as tokenization, stemming, part-of-speech tagging, and parsing. It provides a wide range of tools and resources for text processing and analysis, making it a popular choice for both beginners and experts in NLP.

**Installation**: pip install nltk

**Import**: import nltkImport: import nltk

## spaCy:

Explanation: spaCy is a modern NLP library featuring tokenization, named entity recognition, dependency parsing, and text classification. It is designed for efficient processing of large volumes of text data and is known for its speed, accuracy, and ease of use.

**Installation**: pip install spacy

**Import**: import spacy

## Gensim:

Explanation: Gensim is a library for topic modeling, document similarity analysis, and word embedding techniques like Word2Vec and Doc2Vec. It provides tools for exploring and analyzing large collections of text data, making it suitable for tasks such as document clustering, information retrieval, and semantic analysis.

**Installation**: pip install gensim

**Import**: import gensim

## TextBlob:

Explanation: TextBlob is a simple and intuitive library built on NLTK and Pattern for tasks like sentiment analysis, part-of-speech tagging, and noun phrase extraction. It provides a high-level interface for common NLP tasks, making it easy to use for beginners while still offering advanced features for more experienced users.

**Installation**: pip install textblob

**Import**: from textblob import TextBlob

## TensorFlow NLP:

Explanation: TensorFlow NLP is part of the TensorFlow ecosystem, providing tools and models for various NLP tasks such as sequence labeling, machine translation, and text generation. It leverages the power of TensorFlow for efficient processing and training of NLP models, enabling developers to build and deploy state-of-the-art NLP applications.

**Installation**: pip install tensorflow-text

**Import**: import tensorflow\_text

# Applications of Natural Language Processing

Natural Language Processing (NLP) finds application across a wide range of fields, revolutionizing how we interact with technology, analyze data, and communicate with each other. Here are some key applications of NLP:

## Question Answering

Question Answering (QA) is a branch of Natural Language Processing (NLP) that focuses on developing systems capable of automatically generating accurate and relevant answers to questions posed by humans in natural language. QA systems aim to mimic human-like comprehension and reasoning abilities, enabling them to understand the meaning and context of questions and provide appropriate responses.

These systems typically employ a combination of NLP techniques, machine learning algorithms, and knowledge representation methods to analyze questions, search for relevant information, and generate accurate answers. QA systems may operate on various types of text data sources, including structured databases, unstructured text documents, and web pages.

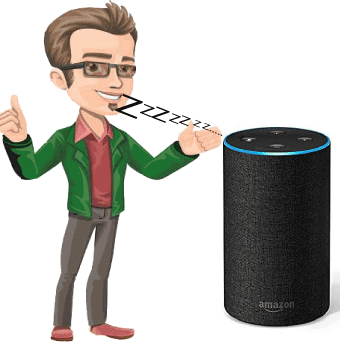


Figure 3:- Question Answer System

## Key components of QA systems

Question Understanding: This involves parsing and analyzing the structure of questions to determine their intent, focus, and key entities or concepts. NLP techniques such as syntactic parsing, semantic analysis, and named entity recognition may be used to understand the meaning and context of questions.

Information Retrieval: QA systems search for relevant information from a knowledge base or corpus of text data that may contain the answer to the question. This process may involve keyword matching, document retrieval, and ranking of candidate answers based on their relevance to the question.

Answer Extraction: Once relevant information is retrieved; QA systems extract the answer from the retrieved documents or data sources. This may involve identifying and extracting relevant passages, sentences, or entities that directly address the question.

Answer Generation: In some cases, QA systems may need to generate a concise and coherent answer based on the extracted information. This may involve summarization, paraphrasing, or synthesis of information to produce a human-readable response. QA systems can be applied to a wide range of tasks and domains, including

Factoid QA, Reading Comprehension, Customer Support, Information Retrieval, Open-Domain QA etc.

## Spam Detection

Spam detection is a vital application of Natural Language Processing (NLP) and machine learning techniques aimed at identifying and filtering out unwanted or unsolicited emails from reaching a user's inbox. This process is essential for maintaining email security, reducing inbox clutter, and protecting users from phishing attempts, malware, and fraudulent activities.

In the context of email communication, Natural Language Processing (NLP) plays a pivotal role in distinguishing between spam and legitimate (non-spam) messages. NLP techniques are employed to analyze the text content, sender information, subject lines, and metadata of incoming emails. By extracting features such as keywords, language patterns, and sender reputation, NLP enables machine learning algorithms to classify emails as either spam or not spam with high accuracy. Spam emails often exhibit characteristic traits such as deceptive subject lines, suspicious sender addresses, and repetitive content, which NLP algorithms can identify and flag for filtering. In contrast, legitimate emails typically contain coherent language, relevant content, and sender information that aligns with the recipient's preferences. Through the application of NLP in spam detection, email providers can effectively safeguard users from unwanted and potentially harmful spam messages, ensuring a safer and more efficient email experience.

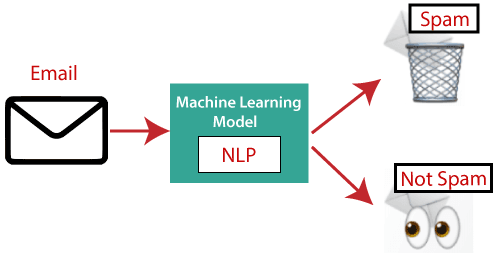


Figure 4:- Mail Spam Detection

## Key Components of Spam Detection

Feature Extraction: NLP techniques are employed to extract relevant features from email messages, such as text content, sender information, subject lines, and metadata. These features serve as inputs to machine learning algorithms for classification.

Training Data Preparation: Spam detection models are trained on labeled datasets containing examples of both spam and legitimate emails. These datasets are used to teach the model to distinguish between spam and non-spam messages based on their features.

Machine Learning Models: Various machine learning algorithms, such as logistic regression, support vector machines (SVM), and ensemble methods like random forests and gradient boosting, are used to build predictive models for spam detection. These models learn patterns and relationships from the training data to classify new email messages as either spam or non-spam.

Model Evaluation: Spam detection models are evaluated using metrics such as accuracy, precision, recall, and F1-score to assess their performance in correctly identifying spam emails while minimizing false positives (legitimate emails incorrectly classified as spam) and false negatives (spam emails incorrectly classified as legitimate).

Thresholding and Filtering: A threshold is set to determine the probability or confidence level above which an email is classified as spam. Emails exceeding this threshold are filtered out and directed to the spam folder or flagged for further review by the user.

Feedback Loop: Spam detection systems often incorporate a feedback loop mechanism where user feedback on classified emails is used to improve the performance of the model over time. This involves retraining the model with updated data to adapt to evolving spam patterns and user preferences.

It ensures spam detection plays a crucial role in maintaining email security and user experience by automatically identifying and filtering out unwanted emails, thereby ensuring that users receive only legitimate and relevant messages in their inbox.

## Sentiment Analysis

Sentiment Analysis, often referred to as opinion mining, is a powerful application of Natural Language Processing (NLP) and statistical techniques aimed at analyzing the attitude, behavior, and emotional state conveyed by text data, particularly on the web. By leveraging NLP algorithms, Sentiment Analysis assigns sentiment values to text, categorizing it as positive, negative, or neutral based on the underlying sentiment expressed. Additionally, it goes beyond polarity detection to identify the mood or emotion conveyed by the text, such as happiness, sadness, anger, and more. This analytical approach enables businesses and organizations to gain valuable insights into public opinion, customer feedback, and social media sentiment, empowering them to make informed decisions, tailor marketing strategies, and enhance customer experiences.

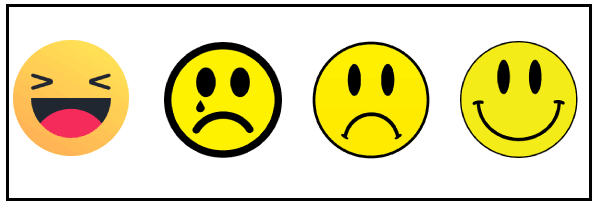


Figure 5:- Sentiment Analysis

Machine Translation

Machine Translation (MT) is a transformative application of Natural Language Processing (NLP) that enables the automatic translation of text or speech from one natural language to another. By leveraging advanced algorithms and linguistic models, MT systems analyze the structure and semantics of input text in the source language and generate equivalent text in the target language. This process involves various stages, including text preprocessing, alignment of parallel corpora, statistical modeling, and neural network-based approaches. Prominent examples of MT systems include Google Translator, which offers translation services for a wide range of languages and facilitates cross-lingual communication across the globe. MT has revolutionized how we access and interact with content in different languages, facilitating cross-cultural communication, international collaboration, and multilingual information exchange.



Figure 6:- Google Translation

## Spelling Correction

Microsoft Corporation offers word processing software such as Microsoft Word and presentation software like PowerPoint, which include built-in features for spelling correction. These applications utilize advanced Natural Language Processing (NLP) algorithms and dictionaries to automatically identify and correct spelling errors in real-time as users type or edit documents. The spelling correction functionality highlights misspelled words with squiggly red underlines and suggests corrections or alternative spellings to users, helping them to produce error-free documents with accuracy and efficiency. Additionally, these software tools often provide options for customizing spell-check settings, adding new words to the dictionary, and adjusting language preferences to accommodate different linguistic variations and user preferences. Overall, the integration of spelling correction capabilities within Microsoft's word processor and presentation software enhances the quality and professionalism of documents and presentations created by users, contributing to a seamless and productive user experience.

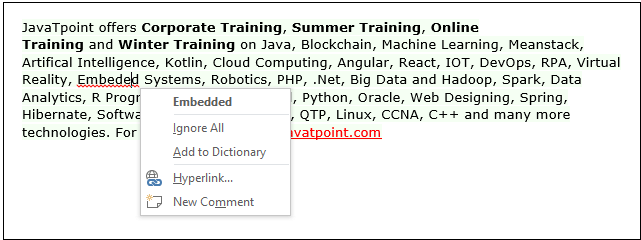


Figure 7:- Spelling Correction

## Speech Recognition

Speech recognition is a pivotal technology utilized for converting spoken words into text, facilitating a multitude of applications across various domains. Commonly employed in mobile devices, speech recognition enables hands-free interaction and voice commands, enhancing user convenience and accessibility. In home automation systems, speech recognition enables users to control smart devices and appliances using voice commands, fostering seamless integration and automation within the home environment. Furthermore, speech recognition is instrumental in video transcription and captioning, aiding in content accessibility and comprehension for individuals with hearing impairments. In professional settings, speech recognition enables users to dictate text directly into applications like Microsoft Word, streamlining the process of document creation and transcription. Moreover, speech recognition finds application in voice biometrics for identity verification and authentication, as well as in voice user interfaces (VUIs) for interactive communication with devices and systems. Overall, the versatility and utility of speech recognition technology contribute to its widespread adoption in diverse applications, revolutionizing how we interact with technology and enabling more natural and intuitive user experiences.

## Chatbots

Implementing chatbots is indeed a significant application of Natural Language Processing (NLP) that has gained widespread adoption among companies seeking to enhance customer service and engagement. Chatbots leverage NLP techniques to understand and respond to user queries and requests in natural language, providing conversational interfaces for interacting with users in real-time. These chatbots are integrated into various platforms, including websites, messaging apps, and social media channels, enabling companies to offer round-the-clock customer support and assistance.

By implementing chatbots, companies can automate routine inquiries, handle frequently asked questions, and provide personalized recommendations or assistance to users. Additionally, chatbots can efficiently triage customer inquiries, escalating more complex issues to human agents when necessary, thereby improving operational efficiency and reducing response times. Furthermore, chatbots can gather valuable insights into customer preferences, behavior, and sentiment through NLP-driven sentiment analysis and conversation analytics, enabling companies to tailor their products, services, and marketing strategies accordingly.

Overall, chatbots represent a powerful application of NLP in customer service and support, offering companies a scalable and cost-effective solution for engaging with customers, resolving inquiries, and delivering personalized experiences. As NLP technology continues to advance, chatbots are poised to become even more sophisticated, intuitive, and capable of meeting the evolving needs and expectations of users in various industries.

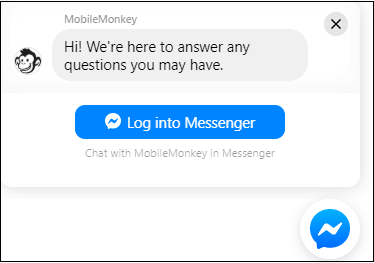


Figure 8:- Chatbot

# NLP Pipeline

Data Collection: Gather the text data you want to analyze. This could be from various sources such as websites, databases, APIs, or even text files.

Preprocessing:

Tokenization: Break the text into individual words or tokens. This step might also involve handling punctuation, numbers, and special characters appropriately.

from nltk.tokenize import word\_tokenize  
text = "This is a sample sentence, showing off the tokenization process."  
tokens = word\_tokenize(text)  
print(tokens)

**Output**

Lowercasing: Convert all text to lowercase to ensure uniformity.

text = "This is a Sample Sentence."  
lowercase\_text = text.lower()  
print(lowercase\_text)

**Output**



Stopword Removal: Remove common words like "and", "the", "is", etc., which don't carry much meaning.

#### Import Stop Words

import nltk  
from nltk.corpus import stopwords  
# Download NLTK stopwords if not already downloaded  
nltk.download('stopwords')  
# Get English stopwords  
stop\_words = set(stopwords.words('english'))  
# Print the stopwords  
print(stop\_words)

**Output**



#### From Sample Sentence

from nltk.corpus import stopwords  
text = "This is a sample sentence with stopwords, such as 'the', 'is', 'and'."  
stop\_words = set(stopwords.words('english'))  
tokens = word\_tokenize(text)  
filtered\_tokens = [word for word in tokens if word.lower() not in stop\_words]  
print(filtered\_tokens)

**Output**



Stemming/Lemmatization: Reduce words to their base or root form. For example, "running" becomes "run", "cars" becomes "car".

#### Stemming Example

from nltk.stem import PorterStemmer  
# Create a PorterStemmer object  
stemmer = PorterStemmer()  
  
# List of words to stem  
words = ["running", "runs", "ran", "runner", "easily", "fairly"]  
# Stem each word in the list  
stemmed\_words = [stemmer.stem(word) for word in words]  
  
# Print the stemmed words  
print(stemmed\_words)

**Output**



#### Lemmatization Example

from nltk.stem import WordNetLemmatizer  
# Create a WordNetLemmatizer object  
lemmatizer = WordNetLemmatizer()  
# List of words to lemmatize  
words = ["running", "runs", "ran", "runner", "easily", "fairly"]  
# Lemmatize each word in the list  
lemmatized\_words = [lemmatizer.lemmatize(word) for word in words]  
# Print the lemmatized words  
print(lemmatized\_words)

**Output**



Normalization: Handle spelling variations, abbreviations, or acronyms to ensure consistency.

Noise Removal: Eliminate irrelevant characters, HTML tags, or any other unwanted elements.

import re  
  
text = "This is a <b>sample</b> sentence with <a href='https://example.com'>HTML</a> tags."  
  
clean\_text = re.sub('<[^<]+?>', '', text)  
print(clean\_text)

Output



## Feature Engineering:

Bag of Words (BoW): Convert text into numerical vectors by counting the frequency of words in the document.

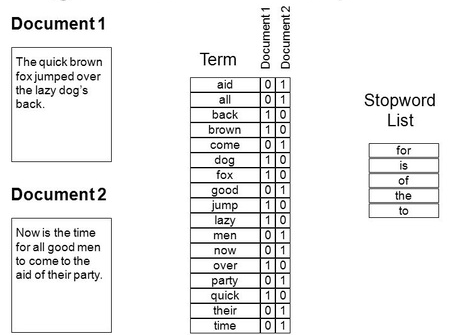
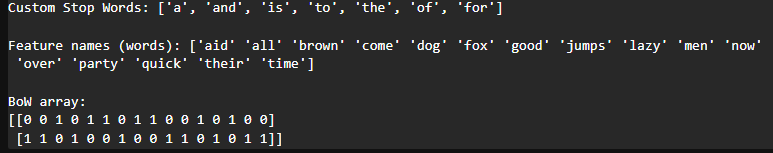


Figure 9:- Bag of Words

#### Sample code

from sklearn.feature\_extraction.text import CountVectorizer  
# Sample corpus  
corpus = [  
 "The Quick Brown Fox Jumps Over the Lazy Dog.",  
 "Now is the time for all good men to come to the aid of their party."  
]  
# Custom stop words  
custom\_stop\_words = ["a", "and", "is", "to", "the", "of", "for"]  
# Create a CountVectorizer object with custom stop words  
vectorizer = CountVectorizer(stop\_words=custom\_stop\_words)  
# Fit the vectorizer to the corpus and transform the corpus into a BoW representation  
bow\_matrix = vectorizer.fit\_transform(corpus)  
# Get the feature names (words)  
feature\_names = vectorizer.get\_feature\_names\_out()  
# Print the stop words list  
print("Custom Stop Words:", custom\_stop\_words)  
# Convert the BoW matrix to an array for easier visualization  
bow\_array = bow\_matrix.toarray()  
# Print the feature names and the BoW array  
print("\nFeature names (words):", feature\_names)  
print("\nBoW array:")  
print(bow\_array)

**Output**



TF-IDF (Term Frequency-Inverse Document Frequency): Similar to BoW, but it also considers the importance of words by weighing down frequent terms and weighing up rare terms.

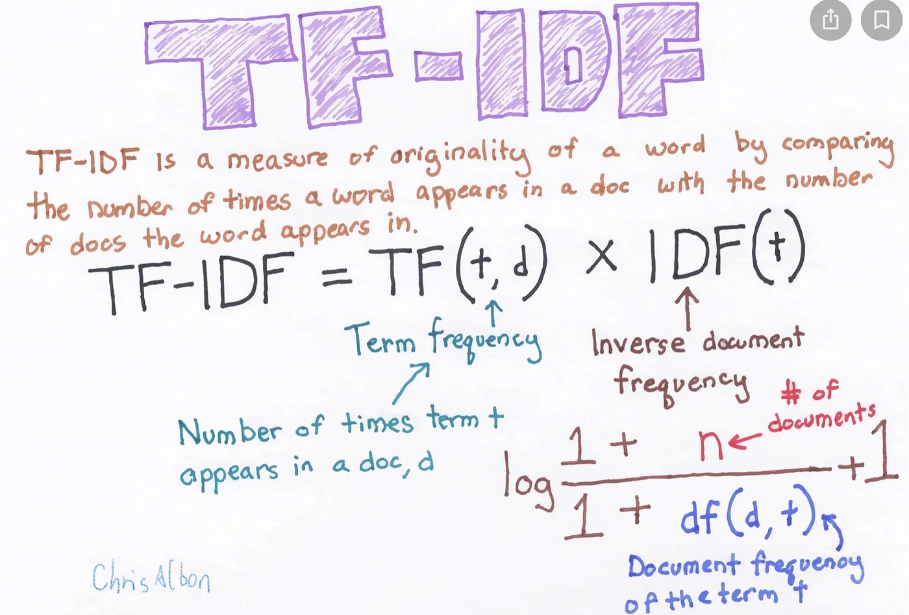


Figure :- TF-IDF

Term Frequency: In document d, the frequency represents the number of instances of a given word t. Therefore, we can see that it becomes more relevant when a word appears in the text, which is rational. Since the ordering of terms is not significant, we can use a vector to describe the text in the bag of term models. For each specific term in the paper, there is an entry with the value being the term frequency.

The weight of a term that occurs in a document is simply proportional to the term frequency.

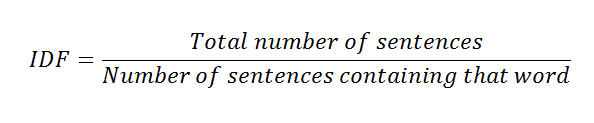


tf(t,d) = count of t in d / number of words in d

Document Frequency: This tests the meaning of the text, which is very similar to TF, in the whole corpus collection. The only difference is that in document d, TF is the frequency counter for a term t, while df is the number of occurrences in the document set N of the term t. In other words, the number of papers in which the word is present is DF.

df(t) = occurrence of t in documents

Inverse Document Frequency: Mainly, it tests how relevant the word is. The key aim of the search is to locate the appropriate records that fit the demand. Since tf considers all terms equally significant, it is therefore not only possible to use the term frequencies to measure the weight of the term in the paper. First, find the document frequency of a term t by counting the number of documents containing the term:



Calculating

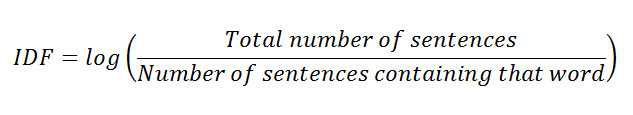
df(t) = N(t)

where

df(t) = Document frequency of a term t

N(t) = Number of documents containing the term t

Term frequency is the number of instances of a term in a single document only; although the frequency of the document is the number of separate documents in which the term appears, it depends on the entire corpus. Now let’s look at the definition of the frequency of the inverse paper. The IDF of the word is the number of documents in the corpus separated by the frequency of the text.



Calculating

idf(t) = N/ df(t) = N/N(t)

idf(t) = log(N/ df(t))

#### Sample Code

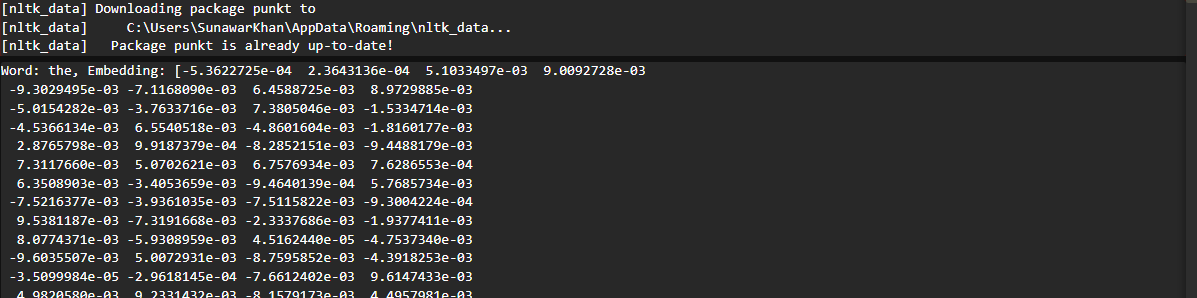
from sklearn.feature\_extraction.text import TfidfVectorizer  
  
# Sample corpus  
corpus = [  
 "The quick brown fox jumps over the lazy dog.",  
 "Now is the time for all good men to come to the aid of their party."  
]  
  
# Create a TfidfVectorizer object  
tfidf\_vectorizer = TfidfVectorizer()  
  
# Fit the vectorizer to the corpus and transform the corpus into a TF-IDF representation  
tfidf\_matrix = tfidf\_vectorizer.fit\_transform(corpus)  
  
# Get the feature names (words)  
feature\_names = tfidf\_vectorizer.get\_feature\_names\_out()  
  
# Convert the TF-IDF matrix to an array for easier visualization  
tfidf\_array = tfidf\_matrix.toarray()  
  
# Print the feature names and the TF-IDF array  
print("Feature names (words):", feature\_names)  
print("\nTF-IDF array:")  
print(tfidf\_array)

Word Embeddings: Represent words in a continuous vector space where the relationships between words are preserved. Techniques like Word2Vec, GloVe, or FastText are commonly used.

#### Word2Vec

from gensim.models import Word2Vec  
from nltk.tokenize import word\_tokenize  
import nltk  
nltk.download('punkt')  
  
# Sample corpus  
corpus = [  
 "The quick brown fox jumps over the lazy dog.",  
 "Now is the time for all good men to come to the aid of their party."  
]  
  
# Tokenize the corpus  
tokenized\_corpus = [word\_tokenize(document.lower()) for document in corpus]  
  
# Train the Word2Vec model  
model = Word2Vec(sentences=tokenized\_corpus, vector\_size=100, window=5, min\_count=1, workers=4)  
  
# Get the word embeddings for specific words  
word\_embeddings = {  
 word: model.wv[word]  
 for word in model.wv.key\_to\_index  
}  
  
# Print the word embeddings for specific words  
for word, embedding in word\_embeddings.items():  
 print(f"Word: {word}, Embedding: {embedding}")

**Output**

****

Python Implementation

# Setting up the NLP Environment

In this chapter, we guide you through the essential steps to set up a robust Natural Language Processing (NLP) environment. From installing crucial Python libraries to downloading language models and datasets, this chapter provides a hands-on approach to prepare your system for NLP development.

## Installing Essential Python Libraries

Step-by-Step Guide for Installing NLTK, spaCy, and TextBlob:

Natural Language Processing in Python relies on powerful libraries that simplify complex tasks. We begin by walking you through the installation process for three fundamental NLP libraries:

*# Python code snippet for installing NLTK, spaCy, and TextBlob*  
*# Ensure you have Python and pip installed before running these commands*  
  
*# Install NLTK*  
pip install nltk  
  
*# Install spaCy*  
pip install spacy  
  
*# Download spaCy English language model*  
python -m spacy download en  
  
*# Install TextBlob*  
pip install textblob

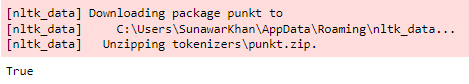
These commands use the Python package manager, pip, to install the NLTK, spaCy, and TextBlob libraries. Additionally, spaCy requires downloading a language model, in this case, the English language model. This sets the foundation for powerful NLP capabilities in your Python environment.

## Downloading Language Models and Datasets

### Accessing Pre-trained Models for Various NLP Tasks:

Many NLP tasks benefit from pre-trained models, which have been trained on vast amounts of data and can be fine-tuned for specific applications. In this section, we guide you on obtaining and using these models:

*# Python code snippet for downloading pre-trained models*  
import nltk  
from nltk import download  
  
*# Download NLTK data (for tasks such as tokenization)*  
nltk.download('punkt')



*# spaCy English model is already downloaded in the previous section*  
*# Use it in your spaCy-based tasks*  
import spacy

*# Download the English language model*

spacy.cli.download("en\_core\_web\_sm")



*# Load the model after downloading*  
nlp = spacy.load("en\_core\_web\_sm")

*# Downloading additional TextBlob corpora*  
from textblob import download\_corpora  
download\_corpora()

These snippets showcase how to download essential data and language models for NLTK, spaCy, and TextBlob. Proper configuration ensures that your NLP applications have access to the necessary tools and resources.

According to our discussion, your environment will be equipped with the foundational tools and datasets needed for effective Natural Language Processing. The code provided serve as a practical guide, enabling you to follow along and replicate the setup on your own machine. This chapter lays the groundwork for the hands-on exploration of NLP concepts and applications.

# **Text** **Preprocessing**

Text preprocessing is a critical step in any Natural Language Processing (NLP) pipeline. In this chapter, we delve into the intricacies of cleaning and preparing text data for analysis, covering techniques such as tokenization, stop word removal, punctuation removal, stemming, and lemmatization.

## Cleaning and Tokenizing Text Data

Text data often contains noise in the form of HTML tags, special characters, and other artifacts that can hinder analysis. Here's a detailed exploration of cleaning and tokenization:

*# Python code snippet for cleaning and tokenizing text data*  
import re  
from nltk.tokenize import word\_tokenize  
  
def clean\_and\_tokenize(text):  
 *# Remove HTML tags and special characters*  
 clean\_text = re.sub(r'<.\*?>|\&\w+;', '', text)  
   
 *# Tokenize the cleaned text*  
 tokens = word\_tokenize(clean\_text)  
   
 return tokens  
  
*# Sample text*  
text = "<p>This is a sample text for text preprocessing.</p>"  
  
*# Clean and tokenize the text*  
tokens = clean\_and\_tokenize(text)  
print("Original Text:", text)  
print("Cleaned and Tokenized:", tokens)

### Output



This code snippet defines a function clean\_and\_tokenize that removes HTML tags and special characters using regular expressions and then tokenizes the cleaned text using the NLTK library.

## Removing Stop Words and Punctuation

Stop words (common words like "the," "and," "is") and punctuation marks often do not contribute much to the semantic meaning of text and can be removed to focus on more meaningful words:

*#import libraries*  
import nltk  
nltk.download('stopwords')  
  
*# Python code snippet for removing stop words and punctuation*  
from nltk.corpus import stopwords  
from string import punctuation  
  
def remove\_stopwords\_and\_punctuation(tokens):  
    *# Remove stop words and punctuation*  
    stop\_words = set(stopwords.words('english'))  
    filtered\_tokens = [word for word in tokens if word.lower() not in stop\_words and word.isalpha()]  
      
    return filtered\_tokens  
*# Sample tokens from the previous snippet*  
tokens = ['This', 'is', 'a', 'sample', 'text', 'for', 'text', 'preprocessing']  
  
*# Remove stop words and punctuation*  
filtered\_tokens = remove\_stopwords\_and\_punctuation(tokens)  
print("Original Tokens:", tokens)  
print("Filtered Tokens:", filtered\_tokens)

### Output



This code demonstrates how to use NLTK's stop words list and Python's string module to filter out stop words and punctuation from the tokenized text.

## Stemming and Lemmatization

Stemming and lemmatization aim to reduce words to their base or root form, simplifying the analysis of text. NLTK provides tools for both stemming and lemmatization:

*#import libraries*  
import nltk  
nltk.download('stopwords')

nltk.download('omw-1.4')

nltk.download('wordnet')

*# Python code snippet for stemming and lemmatization*  
from nltk.stem import PorterStemmer, WordNetLemmatizer  
  
def apply\_stemming\_and\_lemmatization(tokens):  
 *# Stemming*  
 ps = PorterStemmer()  
 stemmed\_tokens = [ps.stem(word) for word in tokens]  
   
 *# Lemmatization*  
 lemmatizer = WordNetLemmatizer()  
 lemmatized\_tokens = [lemmatizer.lemmatize(word) for word in tokens]  
   
 return stemmed\_tokens, lemmatized\_tokens  
  
*# Sample tokens from the previous snippet*  
tokens = ['This', 'is', 'a', 'sample', 'text', 'for', 'text', 'preprocessing']  
  
*# Apply stemming and lemmatization*  
stemmed\_tokens, lemmatized\_tokens = apply\_stemming\_and\_lemmatization(tokens)  
print("Original Tokens:", tokens)  
print("Stemmed Tokens:", stemmed\_tokens)  
print("Lemmatized Tokens:", lemmatized\_tokens)

Output



This code showcases how to use NLTK's PorterStemmer for stemming and WordNetLemmatizer for lemmatization.

By mastering these text preprocessing techniques, you ensure that your NLP models operate on clean and meaningful text data, setting the stage for more accurate and insightful analyses in subsequent chapters.

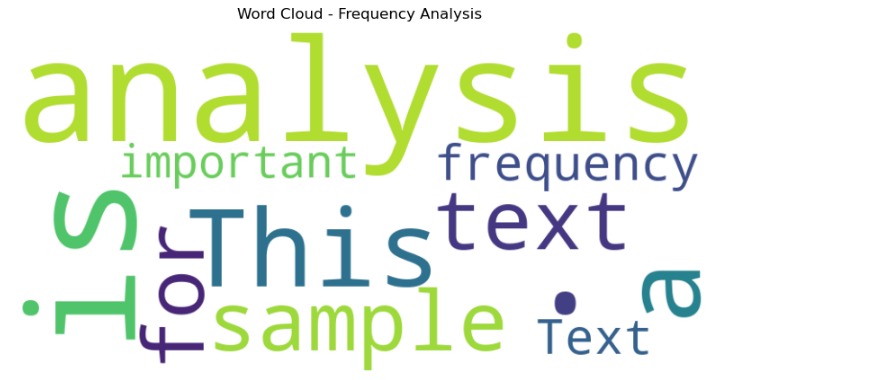
# Text Analysis Techniques

Text analysis is a fundamental aspect of Natural Language Processing (NLP), providing insights into the structure, content, and patterns within textual data. This chapter explores various text analysis techniques, including frequency analysis and visualization, part-of-speech tagging, and Named Entity Recognition (NER).

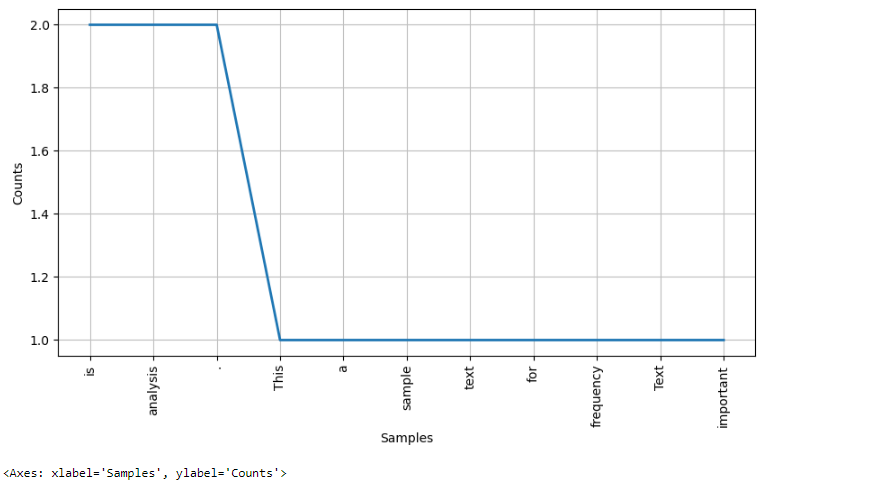
## Frequency Analysis and Visualization

Analyzing the frequency of words in a text corpus helps uncover patterns and key terms. Visualization, such as word clouds and bar charts, provides a clear representation of the most common words. Let's explore this through Python code:

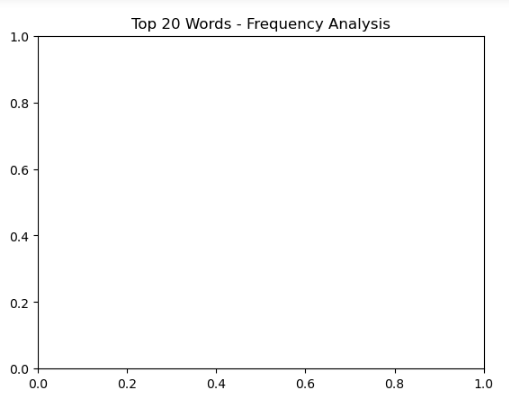
*# Python code snippet for frequency analysis and visualization*  
import matplotlib.pyplot as plt  
from wordcloud import WordCloud  
from nltk import FreqDist  
from nltk.tokenize import word\_tokenize  
  
*# Sample text*  
text = "This is a sample text for frequency analysis. Text analysis is important."  
  
*# Tokenize the text*  
tokens = word\_tokenize(text)  
  
*# Calculate word frequencies*  
freq\_dist = FreqDist(tokens)  
  
*# Generate a word cloud*  
wordcloud = WordCloud(width=800, height=400, background\_color='white').generate\_from\_frequencies(freq\_dist)  
  
*# Plot the word cloud*  
plt.figure(figsize=(10, 5))  
plt.imshow(wordcloud, interpolation='bilinear')  
plt.axis('off')  
plt.title('Word Cloud - Frequency Analysis')  
plt.show()



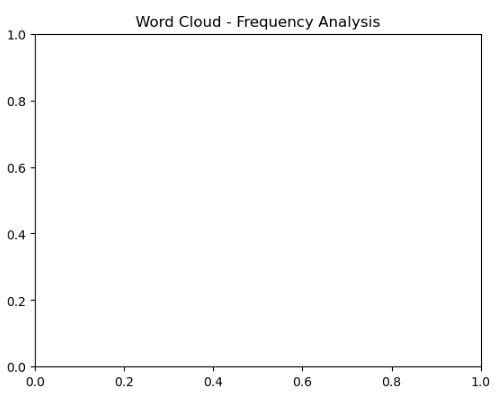
*# Plot a bar chart of word frequencies*  
plt.figure(figsize=(10, 5))  
freq\_dist.plot(20, cumulative=False)



*# Plot a bar chart of word frequencies*  
plt.figure(figsize=(10, 5))  
freq\_dist.plot(20, cumulative=False)  
plt.title('Top 20 Words - Frequency Analysis')  
plt.show()



plt.title('Word Cloud - Frequency Analysis')  
plt.show()



This code snippet demonstrates the use of NLTK for tokenization and frequency distribution, as well as Matplotlib and WordCloud for visualizing word frequencies. The resulting word cloud and bar chart provide a quick overview of the most prevalent words in the given text.

## Part-of-Speech Tagging

Part-of-speech (POS) tagging involves assigning grammatical categories (such as nouns, verbs, adjectives) to words in a sentence. NLTK provides tools for part-of-speech tagging:

import nltk  
nltk.download('averaged\_perceptron\_tagger')

*# Python code snippet for part-of-speech tagging*  
from nltk import pos\_tag  
from nltk.tokenize import word\_tokenize  
  
*# Sample text*  
text = "This is a sample sentence for part-of-speech tagging."  
  
*# Tokenize the text*  
tokens = word\_tokenize(text)  
  
*# Perform part-of-speech tagging*  
pos\_tags = pos\_tag(tokens)  
print("Part-of-Speech Tags:", pos\_tags)

### Output



## Counting POS Tags–Chunking

The identification of parts of speech (POS) and short phrases can be done with the help of chunking. It is one of the important processes in natural language processing. As we are aware about the process of tokenization for the creation of tokens, chunking actually is to do the labeling of those tokens. In other words, we can say that we can get the structure of the sentence with the help of chunking process.

### Example

In the following example, we will implement Noun-Phrase chunking, a category of chunking which will find the noun phrase chunks in the sentence, by using NLTK Python module.

Consider the following steps to implement noun-phrase chunking −

#### Step 1: Chunk grammar definition

In this step, we need to define the grammar for chunking. It would consist of the rules, which we need to follow.

#### Step 2: Chunk parser creation

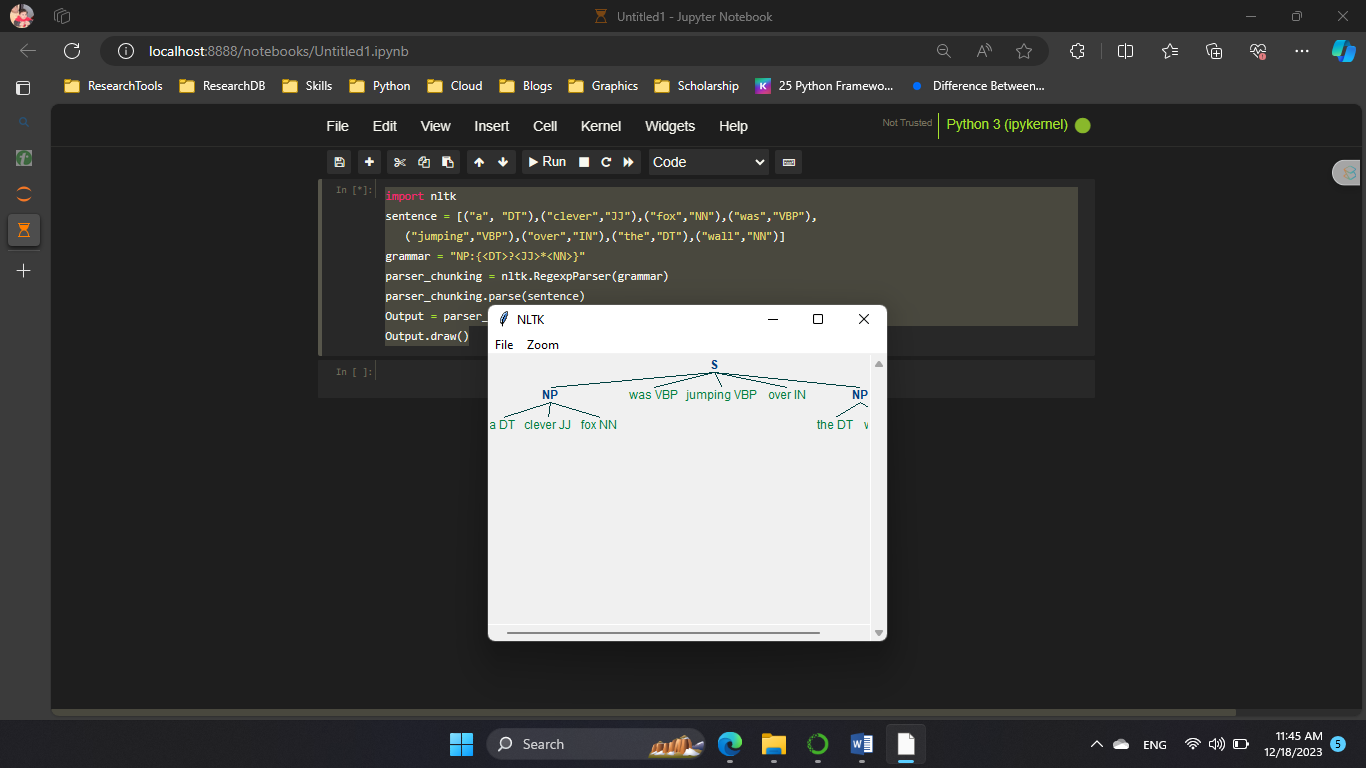
Next, we need to create a chunk parser. It would parse the grammar and give the output.

#### Step 3: The Output

In this step, we will get the output in a tree format.

import nltk  
sentence = [("a", "DT"),("clever","JJ"),("fox","NN"),("was","VBP"),  
 ("jumping","VBP"),("over","IN"),("the","DT"),("wall","NN")]  
grammar = "NP:{<DT>?<JJ>\*<NN>}"  
parser\_chunking = nltk.RegexpParser(grammar)  
parser\_chunking.parse(sentence)  
Output = parser\_chunking.parse(sentence)  
Output.draw()

### Output



## Named Entity Recognition (NER)

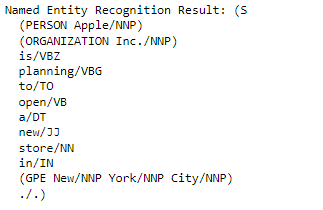
Named Entity Recognition is the process of identifying and classifying entities (such as names of people, organizations, locations) within text. NLTK provides a basic NER classifier:

import nltk

nltk.download('maxent\_ne\_chunker')

nltk.download('words')  
*# Python code snippet for Named Entity Recognition (NER)*  
from nltk import ne\_chunk  
from nltk.tokenize import word\_tokenize  
  
*# Sample text*  
text = "Apple Inc. is planning to open a new store in New York City."  
  
*# Tokenize the text*  
tokens = word\_tokenize(text)  
  
*# Perform Named Entity Recognition*  
ner\_result = ne\_chunk(pos\_tag(tokens))  
print("Named Entity Recognition Result:", ner\_result)

### Output



This code uses NLTK's ne\_chunk function to perform Named Entity Recognition on a given text. The output is a tree structure with identified entities marked.

This chapter equips you with essential techniques for analyzing text data, from exploring word frequencies to identifying grammatical components and named entities. The provided Python code snippets serve as practical examples to implement these text analysis techniques in your NLP projects.

# Sentiment Analysis Unveiled

Sentiment analysis, also known as opinion mining, is a crucial aspect of Natural Language Processing (NLP) that involves determining the sentiment or emotional tone expressed in a piece of text. This chapter delves into the understanding of sentiment analysis, its applications, and a hands-on implementation using the TextBlob library in Python.

## Understanding Sentiment Analysis

### Significance in Social Media and Customer Feedback:

Sentiment analysis is widely employed in social media monitoring to gauge public opinions about products, services, or events. Additionally, it plays a pivotal role in analyzing customer feedback, helping businesses understand the sentiments expressed in reviews, comments, and surveys.

### Challenges and Nuances in Sentiment Analysis:

While sentiment analysis provides valuable insights, it is not without challenges. Ambiguity, sarcasm, and cultural nuances can make it challenging to accurately determine sentiment. This section explores the complexities associated with sentiment analysis and strategies to address them.

### Implementing Sentiment Analysis using TextBlob

TextBlob is a powerful Python library that simplifies text processing tasks, including sentiment analysis. Let's walk through a hands-on implementation:

*# Python code snippet for sentiment analysis using TextBlob*  
from textblob import TextBlob  
  
*# Sample text*  
text = "I love natural language processing!"  
  
*# Create a TextBlob object*  
blob = TextBlob(text)  
  
*# Analyze sentiment*  
polarity = blob.sentiment.polarity  
subjectivity = blob.sentiment.subjectivity  
  
*# Print sentiment analysis results*  
print(f"Text: {text}")  
print(f"Sentiment Polarity: {polarity}")  
print(f"Sentiment Subjectivity: {subjectivity}")

### Output



In this code snippet, the TextBlob library is used to create a TextBlob object from the sample text. The sentiment property of the TextBlob object provides both polarity (ranging from -1 to 1, where -1 is negative, 1 is positive) and subjectivity (ranging from 0 to 1, where 0 is objective, 1 is subjective) scores.

This hands-on implementation allows you to quickly assess the sentiment expressed in a given text and provides a foundation for incorporating sentiment analysis into your NLP projects.

This chapter not only provides a conceptual understanding of sentiment analysis but also guides you through a practical implementation using the TextBlob library. Sentiment analysis is a valuable tool for businesses, researchers, and developers seeking to understand the emotions conveyed in textual data.

# Language Translation Unleashed

Language translation is a pivotal application of Natural Language Processing (NLP), enabling communication and understanding across diverse linguistic contexts. This chapter introduces language translation, discusses its significance in global communication, and guides you through the implementation of a basic language translation model using transformers, a powerful architecture in the field of NLP.

## Introduction to Language Translation

### Power of Language Translation in Global Communication:

Language barriers can impede effective communication in our interconnected world. Language translation addresses this challenge by facilitating the exchange of information, ideas, and culture across different languages. From business interactions to international collaborations, language translation plays a vital role in fostering global communication.

### Real-world Examples of Translation Technologies:

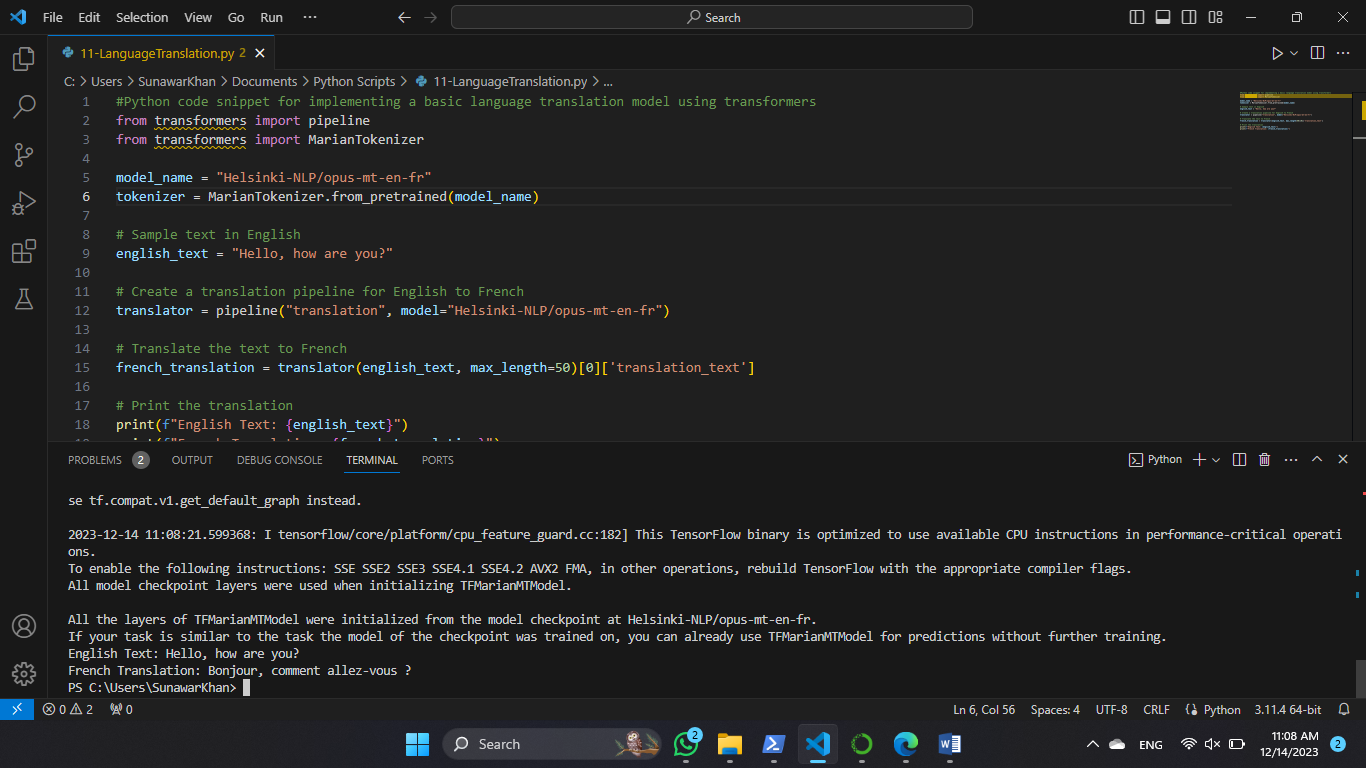
This section showcases real-world applications of language translation technologies. From online language translation services to multilingual chatbots, these examples illustrate the practical impact of NLP in breaking down language barriers.

### Implementing a Basic Language Translation Model using Transformers

Transformers have revolutionized the field of NLP, and models like GPT (Generative Pre-trained Transformer) have demonstrated exceptional language understanding capabilities. Let's explore a basic implementation of language translation using the transformers library:

*# Python code snippet for implementing a basic language translation model using transformers*  
from transformers import pipeline  
  
*# Sample text in English*  
english\_text = "Hello, how are you?"  
  
*# Create a translation pipeline for English to French*  
translator = pipeline("translation", model="Helsinki-NLP/opus-mt-en-fr")  
  
*# Translate the text to French*  
french\_translation = translator(english\_text, max\_length=50)[0]['translation\_text']  
  
*# Print the translation*  
print(f"English Text: {english\_text}")  
print(f"French Translation: {french\_translation}")

### Output



In this code snippet, the transformers library is used to create a translation pipeline for English to French. The chosen model is "Helsinki-NLP/opus-mt-en-fr," which is pre-trained on parallel English-French data. The result is a French translation of the provided English text.

This hands-on example demonstrates how easily transformers can be leveraged for language translation tasks. The versatility of transformer models allows for translations between various language pairs by selecting the appropriate pre-trained model.

By the end of this chapter, you will have gained insights into the transformative power of language translation technologies and the capabilities of transformer models in breaking down language barriers. The provided code snippet serves as a starting point for incorporating language translation into your NLP applications.

# Text Classification Mastery

Text classification is a fundamental task in Natural Language Processing (NLP) that involves assigning predefined categories or labels to text documents based on their content. This chapter provides a comprehensive overview of text classification, covering its significance, applications, and a detailed guide on building a simple text classifier using machine learning techniques, specifically Naive Bayes and Support Vector Machines (SVM).

## Overview of Text Classification

### Defining Text Classification and Its Applications:

Text classification, also known as text categorization, is the process of assigning predefined labels or categories to text documents based on their content. This technique finds applications in various domains, such as spam detection in emails, sentiment analysis in customer reviews, and topic categorization in news articles. Understanding text classification is crucial for solving real-world problems where organizing and categorizing large volumes of text data is essential.

### Business Use Cases: Spam Detection, Sentiment Classification, and More:

This section explores specific business use cases to illustrate the practical applications of text classification. Examples include spam detection to filter out unwanted emails, sentiment classification to gauge customer opinions, and topic categorization for organizing news articles. These applications highlight the versatility of text classification in addressing diverse challenges.

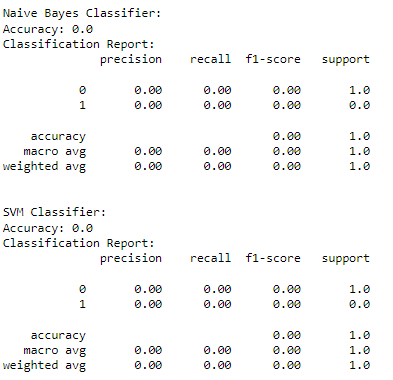
## Building a Simple Text Classifier using Machine Learning

### Practical Implementation with Naive Bayes and SVM:

Text classification often involves machine learning algorithms that learn patterns from labeled training data and then apply these patterns to classify unseen text. Two popular algorithms for text classification are Naive Bayes and Support Vector Machines (SVM). Let's dive into a hands-on implementation using the scikit-learn library:

*# Python code snippet for building a simple text classifier using Naive Bayes and SVM*  
from sklearn.model\_selection import train\_test\_split  
from sklearn.feature\_extraction.text import CountVectorizer  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn.svm import SVC  
from sklearn.metrics import accuracy\_score, classification\_report  
  
*# Sample data: Positive and negative movie reviews*  
texts = ["This movie is fantastic!", "I hated the acting in this film.", "An absolute masterpiece.", "The plot was confusing."]  
labels = [1, 0, 1, 0] *# 1: Positive, 0: Negative*  
  
*# Split the data into training and testing sets*  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(texts, labels, test\_size=0.25, random\_state=42)  
  
*# Create a bag-of-words representation of the text data*  
vectorizer = CountVectorizer()  
X\_train\_vectorized = vectorizer.fit\_transform(X\_train)  
X\_test\_vectorized = vectorizer.transform(X\_test)  
  
*# Train a Naive Bayes classifier*  
nb\_classifier = MultinomialNB()  
nb\_classifier.fit(X\_train\_vectorized, y\_train)  
  
*# Predict using Naive Bayes*  
nb\_predictions = nb\_classifier.predict(X\_test\_vectorized)  
  
*# Train an SVM classifier*  
svm\_classifier = SVC()  
svm\_classifier.fit(X\_train\_vectorized, y\_train)  
  
*# Predict using SVM*  
svm\_predictions = svm\_classifier.predict(X\_test\_vectorized)  
  
*# Evaluate the classifiers*  
print("Naive Bayes Classifier:")  
print(f"Accuracy: {accuracy\_score(y\_test, nb\_predictions)}")  
print("Classification Report:")  
print(classification\_report(y\_test, nb\_predictions))  
  
print("\nSVM Classifier:")  
print(f"Accuracy: {accuracy\_score(y\_test, svm\_predictions)}")  
print("Classification Report:")  
print(classification\_report(y\_test, svm\_predictions))

### Output



In this code snippet, the scikit-learn library is used to preprocess the text data, create a bag-of-words representation, and train classifiers based on Naive Bayes and SVM algorithms. The classifiers are then evaluated on a test set, and performance metrics such as accuracy and classification report are displayed.

This hands-on implementation provides a practical understanding of building a text classifier using machine learning techniques. The code can be adapted to different text classification tasks by replacing the sample data with your own labeled dataset.

By the end of this chapter, you will have a solid foundation in text classification, empowering you to apply these techniques to real-world scenarios where automated categorization of textual information is essential.

Lab Work

**Behind the Scene:** To study Preprocessing of text (Tokenization, Filtration, Script Validation, Stop Word Removal, Stemming)

## Theory:

To preprocess your text simply means to bring your text into a form that is predictable and analyzablefor your task. A task here is a combination of approach and domain.

Machine Learning needs data in the numeric form. We basically used encoding technique (BagOfWord, Bi-gram,n-gram, TF-IDF, Word2Vec) to encode text into numeric vector. But before encoding we first need to clean the text data and this process to prepare (or clean) text databefore encoding is called text preprocessing*,*this is the very first step to solve the NLP problems.

## Tokenization

Tokenization is about splitting strings of text into smaller pieces, or “tokens”. Paragraphs can be tokenized into sentences and sentences can be tokenized into words.

## Filtration

Similarly, if we are doing simple word counts, or trying to visualize our text with a word cloud, stopwords are some of the most frequently occurring words but don’t really tell us anything. We’re often better off tossing the stopwords out of the text. By checking the Filter Stopwords option in the Text Pre-processing tool, you can automatically filter these words out.

## Script Validation

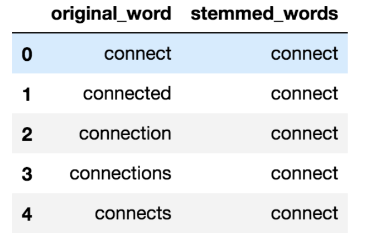
The script must be validated properly.

## Stemming

Stemming is the process of reducing inflection in words (e.g. troubled, troubles) to their root form (e.g. trouble). The “root” in this case may not be a real root word, but just a canonical form of the original word.

Stemming uses a crude heuristic process that chops off the ends of words in the hope of correctly transforming words into its root form. So, the words “trouble”, “troubled” and “troubles” might actually be converted to troublinstead of trouble because the ends were just chopped off (ughh, how crude!).

There are different algorithms for stemming. The most common algorithm, which is also known to be empirically effective for English, is [Porters Algorithm](https://tartarus.org/martin/PorterStemmer/). Here is an example of stemming in action with Porter Stemmer:



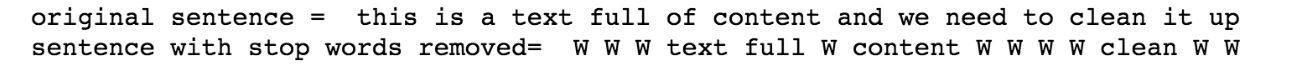
## Stopword Removal

Stop words are a set of commonly used words in a language. Examples of stop words in English are “a”, “the”, “is”, “are” and etc. The intuition behind using stop words is that, by removing low information words from text, we can focus on the important words instead.

For example, in the context of a search system, if your search query is*“what is text preprocessing?”*, you want the search system to focus on surfacing documents that talk about text preprocessing over documents that talk about what is. This can be done by preventing all words from your stop word list from being analyzed. Stop words are commonly applied in search systems, text classification applications, topic modeling, topic extraction and others.

In my experience, stop word removal, while effective in search and topic extraction systems, showed to be non-critical in classification systems. However, it does help reduce the number of features in consideration which helps keep your models decently sized.

Here is an example of stop word removal in action. All stop words are replaced with a dummy character, **W**:

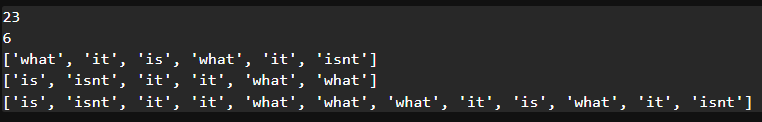


## Code:

### String handling:

**print**(len("what it is what it isnt"))   
s=["what","it","is" ,"what","it","isnt"]   
**print**(len(s))   
x=sorted(s)   
**print**(s)   
**print**(x)   
d=x+s  
**print**(d)

#### Output:



### File handling (tokenization and filtering):

**for** line **in** open("file.txt"):   
**for** word **in** line.split():   
**if** word.endswith('ing'):   
print(word)   
print(len(word))

#### Output:

eating

6

dancing

7

jumping

7

## File.txt

I like eating in restraunt, I like dancing too.

My daughter like bungee jumping.

## Conclusion:

In the above experiment we have studied regarding preprocessing of text in detail like filtration, stop word removal, tokenization, stemming, script validation and have tried to implement the code for it and have successfully executed it.

# Experiment No. 2

**Aim:** To Study Morphological Analysis

## Theory:

## Morphological Analysis:

While performing the morphological analysis, each particular word is analyzed. Non-word tokens such as punctuation are removed from the words. Hence the remaining words are assigned categories. For instance, Ram’s iPhone cannot convert the video from .mkv to .mp4. In Morphological analysis, word by word the sentence is analyzed.So here, Ram is a proper noun, Ram’s is assigned as possessive suffix and .mkv and .mp4 is assigned as a file extension.  
As shown above, the sentence is analyzed word by word. Each word is assigned a syntactic category. The file extensions are also identified present in the sentence which is behaving as an adjective in the above example. In the above example, the possessive suffix is also identified. This is a very important step as the judgment of prefixes and suffixes will depend on a syntactic category for the word. For example, swims and swims are different. One makes it plural, while the other makes it a third-person singular verb. If the prefix or suffix is incorrectly interpreted then the meaning and understanding of the sentence are completely changed. The interpretation assigns a category to the word. Hence, discard the uncertainty from the word.

## Regular Expression:

Regular expressions also called regex. It is a very powerful programming tool that is used for a variety of purposes such as feature extraction from text, string replacement and other string manipulations. A regular expression is a set of characters, or a pattern, which is used to find sub strings in a given string. for ex. extracting all hashtags from a tweet, getting email id or phone numbers etc.,from a large unstructured text content.

In short, if there’s a pattern in any string, you can easily extract, substitute and do variety of other string manipulation operations using regular expressions. Regular expressions are a language in itself since they have their own compilers and almost all popular programming languages support working with regexes.

## Stop Word Removal:

The words which are generally filtered out before processing a natural language are called **stop words**. These are actually the most common words in any language (like articles, prepositions,

pronouns, conjunctions, etc) and does not add much information to the text. Examples of a few stop words in English are “the”, “a”, “an”, “so”, “what”.

Stop words are available in abundance in any human language. By removing these words, we remove the low-level information from our text in order to give more focus to the important information. In order words, we can say that the removal of such words does not show any negative consequences on the model we train for our task.

Removal of stop words definitely reduces the dataset size and thus reduces the training time due to the fewer number of tokens involved in the training.



## Synonym:

The word synonym defines the relationship between different words that have a similar meaning. A simple way to decide whether two words are synonymous is to check for substitutability. Two Words are synonyms in a context if they can be substituted for each for each other without changing the meaning of the sentence.

## Stemming:

Stemming is the process of reducing a word to its word stem that affixes to suffixes and prefixes or to the roots of words known as a lemma. Stemming is important in natural language understanding ([NLU](https://searchenterpriseai.techtarget.com/definition/natural-language-understanding-NLU)) and natural language processing ([NLP](https://searchbusinessanalytics.techtarget.com/definition/natural-language-processing-NLP)).

## Code:

### Regular Expression:

**import** re  
input="The 5 biggest animals are 1. Elephant,2 Rhino and 3 dinasaur"   
input=input.lower()   
print(input)    
result= re.sub(r'\d+','',input)  
print(result)

#### Output:

the 5 biggest animals are 1. elephant,2 rhino and 3 dinasaur

the biggest animals are . elephant, rhino and dinasaur

### Stop word removal:

def punctuations(raw\_review):   
  text = raw\_review   
  text = text.replace("n't", ' not')   
  text = text.replace("'s", ' is')   
  text = text.replace("'re", ' are')   
  text = text.replace("'ve", ' have')   
  text = text.replace("'m", ' am')    
  text = text.replace("'d", ' would')  
  text = text.replace("'ll", ' will')   
  text = text.replace("in", 'ing')   
  **import** re   
  letters\_only = re.sub("[^a-zA-Z]"," ",text)  
  **return**(''.**join**(letters\_only))  
t="Hows's my team doin, you're supposed to be  not loosin"   
p=punctuations(t)  
print(p)

#### Output

Hows is my team doing you are supposed to be not loosing

### Synonym:

#### **import nltk**

#### **nltk.download('wordnet')**

#### **from nltk.corpus import wordnet**

#### **synonyms = []**

#### **for syn in wordnet.synsets('Machine'):**

#### **print(syn.definition())**

#### **for lemma in syn.lemmas():**

#### **synonyms.append(lemma.name())**

#### **print(synonyms)**

#### Output:

['machine', 'machine', 'machine', 'machine', 'simple\_machine', 'machine', 'political\_machine', 'car', 'auto', 'automobile', 'machine', 'motorcar', 'machine', 'machine']

### Stemming:

from nltk.stem import PorterStemmer   
stemmer = PorterStemmer()   
**print**(stemmer.stem('eating'))   
**print**(stemmer.stem('ate'))

#### Output:

eat

ate

## Conclusion:

Thus, in the above experiment we have studied regarding morphological analysis in detail with stemming, synonym, stop word removal, regular expression and tried to implement the code and got proper output.

# Experiment No. 3

**Aim:** To study N-gram model

## Theory:

Given a sequence of N-1 words, an N-gram model predicts the most probable word that might follow this sequence. It's a probabilistic model that's trained on a corpus of text. Such a model is useful in many NLP applications including speech recognition, machine translation and predictive text input.

An N-gram model is built by counting how often word sequences occur in corpus text and then estimating the probabilities. Since a simple N-gram model has limitations, improvements are often made via smoothing, interpolation and backoff.

An N-gram model is one type of a Language Model (LM), which is about finding the probability distribution over word sequences.

Consider two sentences: "There was heavy rain" vs. "There was heavy flood". From experience, we know that the former sentence sounds better. An N-gram model will tell us that "heavy rain" occurs much more often than "heavy flood" in the training corpus. Thus, the first sentence is more probable and will be selected by the model.

A model that simply relies on how often a word occurs without looking at previous words is called **unigram**. If a model considers only the previous word to predict the current word, then it's called **bigram**. If two previous words are considered, then it's a **trigram** model.

An n-gram model for the above example would calculate the following probability:

P('There was heavy rain') = P('There', 'was', 'heavy', 'rain') = P('There')P('was'|'There')P('heavy'|'There was')P('rain'|'There was heavy')

Since it's impractical to calculate these conditional probabilities, using *Markov assumption*, we approximate this to a bigram model:

**P**('There was heavy rain') ~ **P**('There')**P**('was'|'There')**P**('heavy'|'was')**P**('rain'|'heavy')

In speech recognition, input may be noisy and this can lead to wrong speech-to-text conversions. N-gram models can correct this based on their knowledge of the probabilities. Likewise, N-gram models are used in machine translation to produce more natural sentences in the target language.

When correcting for spelling errors, sometimes dictionary lookups will not help. For example, in the phrase "in about fifteen mineuts" the word 'minuets' is a valid dictionary word but it's incorrect in this context. N-gram models can correct such errors.

N-gram models are usually at word level. It's also been used at character level to do stemming, that is, separate the root word from the suffix. By looking at N-gram statistics, we could also classify languages or differentiate between US and UK spellings. For example, 'sz' is common in Czech; 'gb' and 'kp' are common in Igbo.

In general, many NLP applications benefit from N-gram models including part-of-speech tagging, natural language generation, word similarity, sentiment extraction and predictive text input.

## Code:

**import** re   
from nltk.util **import** ngrams  
s = "Machine learning is an important part of AI " "and AI is going to become inmporant for daily functionong "   
tokens = [token **for** token **in** s.split(" ")]  
output = list(ngrams(tokens, 2))   
print(output)

### Output:

[('Machine', 'learning'), ('learning', 'is'), ('is', 'an'), ('an', 'important'), ('important', 'part'), ('part', 'of'), ('of', 'AI'), ('AI', 'and'), ('and', 'AI'), ('AI', 'is'), ('is', 'going'), ('going', 'to'), ('to', 'become'), ('become', 'inmporant'), ('inmporant', 'for'), ('for', 'daily'), ('daily', 'functionong'), ('functionong', '')]

## Conclusion:

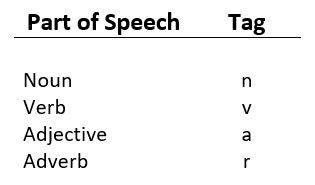
Thus, in the above experiment we have studied regarding N-Gram Model in detail with the help of theory and then tried to implement the code and successfully executed it.

# Experiment No.4

**Aim:** To study POS tagging

## Theory:

It is a process of converting a sentence to forms – list of words, list of tuples (where each tuple is having a form (word, tag)). The tag in case of is a part-of-speech tag, and signifies whether the word is a noun, adjective, verb, and so on.



Default taggingis a basic step for the part-of-speech tagging. It is performed using the DefaultTagger class. The DefaultTagger class takes ‘tag’ as a single argument. NN is the tag for a singular noun. DefaultTagger is most useful when it gets to work with most common part-of-speech tag. that’s why a noun tag is recommended.



Tagging is a kind of classification that may be defined as the automatic assignment of description to the tokens. Here the descriptor is called tag, which may represent one of the part-of-speech, semantic information and so on.

Now, if we talk about Part-of-Speech (PoS) tagging, then it may be defined as the process of assigning one of the parts of speech to the given word. It is generally called POS tagging. In simple words, we can say that POS tagging is a task of labelling each word in a sentence with its appropriate part of speech. We already know that parts of speech include nouns, verb, adverbs, adjectives, pronouns, conjunction and their sub-categories.

Most of the POS tagging falls under Rule Base POS tagging, Stochastic POS tagging and Transformation based tagging.

## Rule-based POS Tagging

One of the oldest techniques of tagging is rule-based POS tagging. Rule-based taggers use dictionary or lexicon for getting possible tags for tagging each word. If the word has more than one possible tag, then rule-based taggers use hand-written rules to identify the correct tag. Disambiguation can also be performed in rule-based tagging by analyzing the linguistic features of a word along with its preceding as well as following words. For example, suppose if the preceding word of a word is article, then word must be a noun.

## Stochastic POS Tagging

Another technique of tagging is Stochastic POS Tagging. Now, the question that arises here is which model can be stochastic. The model that includes frequency or probability (statistics) can be called stochastic. Any number of different approaches to the problem of part-of-speech tagging can be referred to as stochastic tagger.

The simplest stochastic tagger applies the following approaches for POS tagging –

## Word Frequency Approach

In this approach, the stochastic taggers disambiguate the words based on the probability that a word occurs with a particular tag. We can also say that the tag encountered most frequently with the word in the training set is the one assigned to an ambiguous instance of that word. The main issue with this approach is that it may yield inadmissible sequence of tags.

## Tag Sequence Probabilities

It is another approach of stochastic tagging, where the tagger calculates the probability of a given sequence of tags occurring. It is also called n-gram approach. It is called so

because the best tag for a given word is determined by the probability at which it occurs with the n previous tags.

## Transformation-based Tagging

Transformation based tagging is also called Brill tagging. It is an instance of the transformation-based learning (TBL), which is a rule-based algorithm for automatic tagging of POS to the given text. TBL, allows us to have linguistic knowledge in a readable form, transforms one state to another state by using transformation rules.

It draws the inspiration from both the previous explained taggers − rule-based and stochastic. If we see similarity between rule-based and transformation tagger, then like rule-based, it is also based on the rules that specify what tags need to be assigned to what words. On the other hand, if we see similarity between stochastic and transformation tagger then like stochastic, it is machine learning technique in which rules are automatically induced from data.

## HMM for POS Tagging

The POS tagging process is the process of finding the sequence of tags which is most likely to have generated a given word sequence. We can model this POS process by using a Hidden Markov Model (HMM), where tags are the hidden states that produced the observable output, i.e., the words.

## Code:

**import** nltk  
nltk.download('averaged\_perceptron\_tagger')  
nltk.download('punkt')  
**text** = nltk.word\_tokenize("And now for Everything completely Same")   
nltk.pos\_tag(**text**)

### Output:

[('And', 'CC'),

('now', 'RB'),

('for', 'IN'),

('Everything', 'VBG'),

('completely', 'RB'),

('Same', 'JJ')]

## Conclusion:

Thus, we have studied POS Tagging in the above experiment also learned regarding different types of POS Tagging and tried to implement the code for POS Tagging and successfully executed it.

# Experiment No. 5

**Aim:** To study Chunking

## Theory:

Chunk extraction or partial parsing is a process of meaningful extracting short phrases from the sentence (tagged with Part-of-Speech).Chunks are made up of words and the kinds of words are defined using the part-of-speech tags. One can even define a pattern or words that can’t be a part of chuck and such words are known as **chinks**. A ChunkRule class specifies what words or patterns to include and exclude in a chunk.

Defining Chunk patterns:

Chuck patterns are normal regular expressions which are modified and designed to match the part-of-speech tag designed to match sequences of part-of-speech tags. Angle brackets are used to specify an indiviual tag for example – to match a noun tag. One can define multiple tags in the same way.

Chunking is a process of extracting phrases from unstructured text. Instead of just simple tokens which may not represent the actual meaning of the text, its advisable to use phrases such as “**South Africa**” as a single word instead of ‘**South**’ and ‘**Africa**’ separate words.

Chunking in [NLP](https://www.nlpworld.co.uk/nlp-glossary/n/nlp/) is Changing a perception by moving a “chunk”, or a group of bits of information, in the direction of a Deductive or Inductive conclusion through the use of language.

Chunking up or down allows the speaker to use certain language patterns, to utilize the natural internal process through language, to reach for higher meanings or search for more specific bits/portions of missing information.

When we “Chunk Up” the language gets more abstract and there are more chances for agreement, and when we “Chunk Down” we tend to be looking for the specific details that may have been missing in the chunk up.

As an example, if you ask the question “for what purpose cars?” you may get the answer “transport”, which is a higher chunk and more toward abstract.

If you asked “what specifically about a car”? you will start to get smaller pieces of information about a car.

Lateral thinking will be the process of chunking up and then looking for other examples: For example, “for what intentions cars?”, “transportation”, “what are other examples of transportation?” “Buses!”

## Code:

### Noun Phrase chunking:

import nltk   
sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"), ("dog", "NN"), ("barked", "VBD"), ("at", "IN"), ("the", "DT"), ("cat", "NN")]   
grammar = "NP: {<DT>?<JJ>\*<NN>}"   
cp = nltk.RegexpParser(grammar)   
result = cp.parse(sentence)   
print(result)   
result.draw()

#### Output:

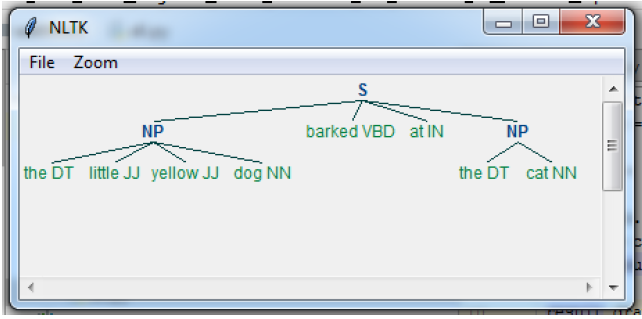
(S

(NP the/DT little/JJ yellow/JJ dog/NN)

barked/VBD

at/IN

(NP the/DT cat/NN))



## Conclusion:

Thus, in the above experiment we have studies regarding chunking and tried to implement the code for same and successfully executed it.

# Experiment No. 6

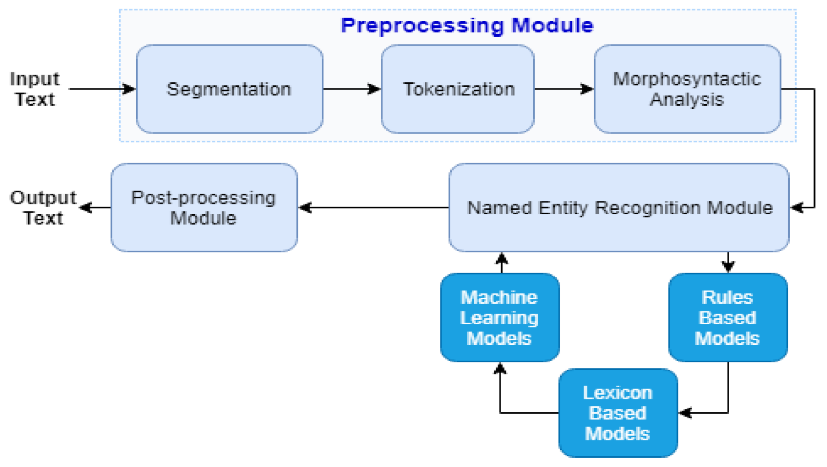
**Aim:** To study Named Entity Recognition

## Theory:

Named Entity Recognition (NER) is a standard NLP problem which involves spotting named entities (people, places, organizations etc.) from a chunk of text, and classifying them into a predefined set of categories. Some of the practical applications of NER include:

* Scanning news articles for the people, organizations and locations reported.
* Providing concise features for search optimization: instead of searching the entire content, one may simply search for the major entities involved.
* Quickly retrieving geographical locations talked about in Twitter posts.

In any text document, there are particular terms that represent specific entities that are more informative and have a unique context. These entities are known as named entities, which more specifically refer to terms that represent real-world objects like people, places, organizations, and so on, which are often denoted by proper names. A naive approach could be to find these by looking at the noun phrases in text documents. Named entity recognition (NER), also known as entity chunking/extraction, is a popular technique used in information extraction to identify and segment the named entities and classify or categorize them under various predefined classes.



## How NER works

At the heart of any NER model is a two step process:

* Detect a named entity
* Categorize the entity

Step one involves detecting a word or string of words that form an entity. Each word represents a token: “The Great Lakes” is a string of three tokens that represents one entity. [Inside-outside-beginning tagging](https://en.wikipedia.org/wiki/Inside%E2%80%93outside%E2%80%93beginning_(tagging)) is a common way of indicating where entities begin and end. We’ll explore this further in a future blog post.

The second step requires the creation of entity categories.

## How is NER used?

NER is suited to any situation in which a high-level overview of a large quantity of text is helpful. With NER, you can, at a glance, understand the subject or theme of a body of text and quickly group texts based on their relevancy or similarity.

Some notable NER use cases include:

## Human resources

Speed up the hiring process by summarizing applicants’ CVs; improve internal workflows by categorizing employee complaints and questions

## Customer support

Improve response times by categorizing user requests, complaints and questions and filtering by priority keywords

## Code:

### Named Entity Recognition

locs = [('Omnicom', 'IN', 'New York'),   
('DDB Needham', 'IN', 'New York'),   
('Kaplan Thaler Group', 'IN', 'New York'),   
('BBDO South', 'IN', 'Atlanta'),   
('Georgia-Pacific', 'IN', 'Atlanta')]   
query = [e1 **for** (e1, rel, e2) **in** locs **if** e2=='Atlanta']   
**print**(query)

### Output:

['BBDO South', 'Georgia-Pacific']

## Conclusion:

Thus, in the above experiment we have studied regarding named entity recognition, working of named entity recognition, how named entity recognition can be used and then implemented the code for the same and successfully executed it.

# Experiment No. 7

**Aim:** Virtual Lab on Word Generation

Theory: Given the root and suffix information, a word can be generated. For example,

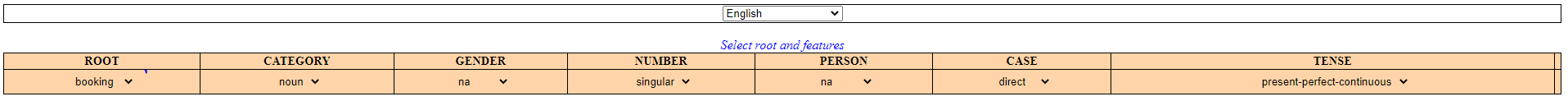
| Language | input:analysis | output:word |
| --- | --- | --- |
| English | rt=boy, cat=n, num=pl | boys |
| English | rt=play, cat=v, num=sg, per=3, tense=pr | plays |

Morphological analysis and generation: Inverse processes.

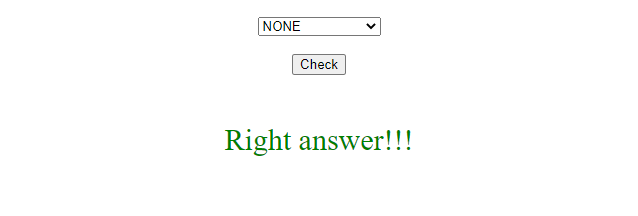
Analysis may involve non-determinism, since more than one analysis is possible.

Generation is a deterministic process. In case a language allows spelling variation, then till that extent, generation would also involve non-determinism.

## Input:

****

### Output:

****

Conclusion:Thus, in the above experiment we have studied regarding Word Generation .

# Experiment No. 8

**Aim:** Miniproject based on NLP applications

Theory:These tools can be very helpful for kids who [struggle with writing](https://www.understood.org/en/learning-thinking-differences/child-learning-disabilities/writing-issues/understanding-your-childs-trouble-with-writing).

To use word prediction, your child needs to use a keyboard to write. This can be an onscreen keyboard on a smartphone or digital tablet. Or it can be a physical keyboard connected to a device or computer.

Those suggestions are shown on the screen, like at the top of an onscreen keyboard. The child clicks or taps on a suggested word, and it’s inserted into the writing.

There are also advanced word prediction tools available. They include:

Tools that read word choices aloud with [text-to-speech](https://www.understood.org/en/school-learning/assistive-technology/assistive-technologies-basics/text-to-speech-technology-what-it-is-and-how-it-works). This is important for kids with reading issues who can’t read what the suggestions are.

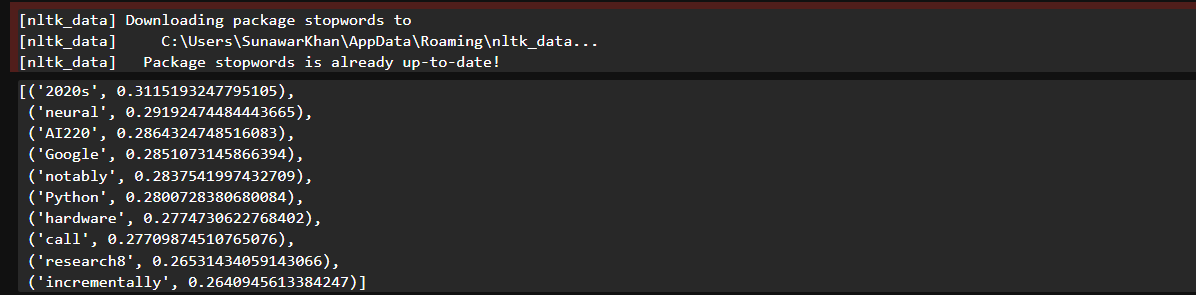
Word prediction tools that make suggestions tailored to specific topics. For instance, the words used in a history paper will differ a lot from those in a science report. To make suggestions more accurate, kids can pick special dictionaries for what they’re writing about.

Tools that display word suggestions in example sentences. This can help kids decide between words that are confusing, like to, too and two.

## Code:

**import** bs4 **as** bs  
**import** urllib.request  
**import** re  
**import** nltk  
**import** string  
**from** nltk.corpus **import** stopwords  
**from** gensim.models **import** Word2Vec  
  
# Download NLTK stopwords  
nltk.download('stopwords')  
  
scrapped\_data = urllib.request.urlopen('https://en.wikipedia.org/wiki/Artificial\_intelligence')  
article = scrapped\_data.read()  
  
parsed\_article = bs.BeautifulSoup(article, 'lxml')  
  
paragraphs = parsed\_article.find\_all('p')  
  
article\_text = ""  
  
**for** p **in** paragraphs:  
 article\_text += p.text  
  
**class** **PreProcessText**(object):  
  
 **def** **\_init\_**(self):  
 **pass**  
  
 **def** **\_\_remove\_punctuation**(self, text):  
 """  
 Takes a String  
 return : Return a String  
 """  
 message = []  
 **for** x **in** text:  
 **if** x **in** string.punctuation:  
 **pass**  
 **else**:  
 message.append(x)  
 message = ''.join(message)  
  
 **return** message  
  
 **def** **\_\_remove\_stopwords**(self, text):  
 """  
 Takes a String  
 return List  
 """  
 words= []  
 **for** x **in** text.split():  
  
 **if** x.lower() **in** stopwords.words('english'):  
 **pass**  
 **else**:  
 words.append(x)  
 **return** words  
  
  
 **def** **token\_words**(self,text=''):  
 """  
 Takes String  
 Return Token also called list of words that is used to  
 Train the Model  
 """  
 message = self.\_\_remove\_punctuation(text)  
 words = self.\_\_remove\_stopwords(message)  
 **return** words  
  
helper = PreProcessText()  
words = helper.token\_words(text=article\_text)  
  
model = Word2Vec(sentences=[words], vector\_size=100, window=5, min\_count=1, workers=4)  
vocabulary = model.wv.key\_to\_index  
sim\_words = model.wv.most\_similar('machine')  
sim\_words

Output:



# Summary

NLP stands at the forefront of artificial intelligence, aiming to imbue computers with the ability to comprehend, interpret, and generate human language in a meaningful and contextually relevant manner. This interdisciplinary field encompasses a diverse range of tasks and techniques tailored towards processing and analyzing natural language data, encompassing both text and speech. Core components of NLP include tokenization, part-of-speech tagging, named entity recognition (NER), sentiment analysis, language modeling, machine translation, text classification, information extraction, question answering, and dialogue systems. With applications spanning customer service, healthcare, finance, education, and social media analysis, NLP empowers industries and domains with its ability to unlock insights from unstructured text data.