Generic NLP system

A generic NLP (Natural Language Processing) system is a broad term that refers to a software or AI system designed to understand, process, and generate human language text in a general or versatile manner. Such systems are not specialized for specific tasks or domains but are instead capable of handling a wide range of natural language understanding and generation tasks. These systems often leverage machine learning techniques and large language models to achieve their functionality.

Here are some key components and capabilities often found in a generic NLP system:

1. **Text Tokenization:** The system can break down input text into smaller units, such as words or subword tokens, to facilitate processing and analysis.
2. **Part-of-Speech Tagging:** It can identify the grammatical parts of speech (e.g., nouns, verbs, adjectives) for each word in a sentence.
3. **Named Entity Recognition (NER):** It can identify and classify named entities in text, such as names of people, places, organizations, dates, and more.
4. **Sentiment Analysis:** The system can determine the sentiment or emotional tone expressed in a piece of text, classifying it as positive, negative, or neutral.
5. **Text Classification:** It can categorize text into predefined classes or topics. For instance, classifying news articles into sports, politics, or entertainment.
6. **Language Translation:** The system can translate text from one language to another, enabling multilingual communication.
7. **Text Summarization:** It can generate concise and coherent summaries of longer texts, extracting the most important information.
8. **Question Answering:** It can answer questions posed in natural language based on a given text or knowledge base.
9. **Language Generation:** It can generate human-like text, including writing essays, stories, or responses to prompts.

10. **Speech Recognition:** Some generic NLP systems may also include speech-to-text capabilities to transcribe spoken language into written text.
11. **Language Model Capabilities:** Many modern generic NLP systems are built upon large pretrained language models, such as GPT-3, GPT-4, or similar architectures. These models can be fine-tuned for specific tasks or used as-is for a wide range of NLP tasks.
12. **Contextual Understanding:** These systems are designed to understand context and generate responses or predictions based on the surrounding text or conversation.
13. **Multimodal NLP:** Some advanced systems can also handle multimodal data, combining text with other forms of information like images or audio.

It's important to note that the capabilities of a generic NLP system can vary widely depending on the specific implementation and the underlying technology. These systems are highly valuable for a variety of applications, including chatbots, virtual assistants, content generation, sentiment analysis, and more, as they provide a versatile foundation for working with human language data.

https://www.tutorialspoint.com/natural_language_processing/natural_language_processing_quick_guide.htm

https://www.geeksforgeeks.org/introduction-to-natural-language-processing/

https://www.geeksforgeeks.org/natural-language-processing-overview/

https://www.javatpoint.com/nlp

Levels/Stages in NLP

https://youtu.be/fHteXkmWN8E?si=U0K5bwS4HwF2kYhp

1. **Morphological Analysis/ Lexical Analysis**

Morphological or Lexical Analysis deals with text at the individual word level. It looks for *morphemes*, the smallest unit of a word. For example, *irrationally* can be broken into *ir* (prefix), *rational* (root) and *-ly* (suffix). Lexical Analysis finds the relation between these morphemes and converts the word into its root

form. A lexical analyzer also assigns the possible Part-Of-Speech (POS) to the word. It takes into consideration the dictionary of the language.

For example, the word “character” can be used as a noun or a verb.

2. **Syntax Analysis**

Syntax Analysis ensures that a given piece of text is correct structure. It tries to parse the sentence to check correct grammar at the sentence level. Given the possible POS generated from the previous step, a syntax analyzer assigns POS tags based on the sentence structure.

For example:

Correct Syntax: *Sun rises in the east.*

Incorrect Syntax: *Rise in sun the east.*

3. **Semantic Analysis**

Consider the sentence: “The apple ate a banana”. Although the sentence is syntactically correct, it doesn’t make sense because apples can’t eat. Semantic analysis looks for meaning in the given sentence. It also deals with combining words into phrases.

For example, “red apple” provides information regarding one object; hence we treat it as a single phrase. Similarly, we can group names referring to the same category, person, object or organisation. “Robert Hill” refers to the same person and not two separate names – “Robert” and “Hill”.

4. **Discourse**

Discourse deals with the effect of a previous sentence on the sentence in consideration. In the text, “Jack is a bright student. He spends most of the time in the library.” Here, discourse assigns “he” to refer to “Jack”.

5. **Pragmatics**

The final stage of NLP, Pragmatics interprets the given text using information from the previous steps. Given a sentence, “Turn off the lights” is an order or request to switch off the lights.

Natural Language Processing is separated into five primary stages or phases, starting with simple word processing and progressing to identifying complicated phrase meanings.

1. Lexical or Morphological Analysis
Lexical or Morphological Analysis is the initial step in NLP. It entails recognizing and analyzing word structures. The collection of words and phrases in a language is referred to as the lexicon. Lexical analysis is the process of breaking down a text file into paragraphs, phrases, and words. The source code is scanned as a stream of characters and converted into intelligible lexemes in this phase. The entire book is divided into paragraphs, phrases, and words.

It refers to the study of text at the level of individual words. It searches for morphemes, which are the smallest units of a word. The lexical analysis identifies the relationship between these morphemes and transforms the word into its root form. The word's probable parts of speech (POS) are also assigned by a lexical analyzer.

2. Syntax Analysis or Parsing
Syntactic or Syntax analysis is a technique for checking grammar, arranging words, and displaying relationships between them. It entails examining the syntax of the words in the phrase and arranging them in a way that demonstrates the relationship between them. Syntax analysis guarantees that the structure of a particular piece of text is proper. It tries to parse the sentence in order to ensure that the grammar is correct at the sentence level. A syntax analyzer assigns POS tags based on the sentence structure given the probable POS created in the preceding stage.

For example, New York goes to John.
This sentence New York goes to John is rejected by the Syntactic Analyzer as it makes no sense.

3. Semantic Analysis
Semantic analysis is the process of looking for meaning in a statement. It concentrates mostly on the literal meaning of words, phrases, and sentences is the main focus. It also deals with putting words together to form sentences. It extracts the text's exact meaning or dictionary definition. The meaning of the text is examined. It is accomplished by mapping the task domain's syntactic structures and objects.

Take the following sentence for example: "The guava ate an apple." The line is syntactically valid, yet it is illogical because guavas cannot eat.

4. Discourse Integration
The term "discourse integration" refers to a feeling of context. The meaning of any sentence is determined by the meaning of the sentence immediately preceding it. In addition, it establishes the meaning of the sentence that follows. The sentences that come before it play a role in discourse integration. That is to

say, that statement or word is dependent on the preceding sentence or words. It's the same with the use of proper nouns and pronouns.

For example, Billy bought it.
The word "it" in the above sentence is dependent on the preceding discourse context. We can see that the "it" does not make sense in this statement. In fact, it refers to anything we don't understand. That is nothing more than the fact that the word "it" is dependent on the preceding sentence, which is not provided. So, once we've learned about "it," we'll be able to simply locate the reference. Discourse is concerned with the impact of a prior sentence on the current sentence.

5. Pragmatic Analysis
The fifth and final phase of NLP is pragmatic analysis. The overall communicative and social content, as well as its impact on interpretation, are the focus of pragmatic analysis. Pragmatic Analysis uses a set of rules that describe cooperative dialogues to help you find the intended result. It covers things like word repetition, who said what to whom, and so on. It comprehends how people communicate with one another, the context in which they converse, and a variety of other factors. It refers to the process of abstracting or extracting the meaning of a situation's use of language. It translates the given text using the knowledge gathered in the preceding stages. "Switch on the TV" when used in a sentence, is an order or request to switch the TV on.

Knowledge in language processing

https://youtu.be/UrNsyqyV7LA?si=KJEXxqJnHVcFuHGP

A natural language understanding system must have knowledge about what the words mean, how words combine to form sentences, how word meanings combine to from sentence meanings and so on.

The different Levels (forms) of knowledge required for natural language understanding are given below.

Phonetic And Phonological Knowledge

Phonetics is the study of language at the level of sounds while phonology is the study of combination of sounds into organized units of speech, the formation of syllables and larger units. Phonetic and phonological knowledge are essential for speech based systems as they deal with how words are related to the sounds that realize them.

Morphological Knowledge

Morphology concerns word formation. It is a study of the patterns of formation of words by the combination of sounds into minimal distinctive units of meaning called mophemes. Morphological knowledge concerns how words are constructed from morphemes.

Syntactic Knowledge

Syntax is the level at which we study how words combine to form phrases, phrases combine to form clauses and clauses join to make sentences. Syntactic analysis concerns sentence formation. It deals with how words can be put together to form correct sentences. It also determines what structural role each word plays in the sentence and what phrases are subparts of what other phrases.

Semantic Knowledge

It concerns meanings of the words and sentences. This is the study of context independent meaning that is the meaning a sentence has, no matter in which context it is used. Defining the meaning of a sentence is very difficult due to the ambiguities involved.

Pragmatic Knowledge

Pragmatics is the extension of the meaning or semantics. Pragmatics deals with the contextual aspects of meaning in particular situations. It concerns how sentences are used in different situations and how use affects the interpretation of the sentence.

Discourse Knowledge

Discourse concerns connected sentences. It is a study of chunks of language which are bigger than a single sentence. Dicourse language concerns inter-sentential links that is how the immediately preceding sentences affect the interpretation of the next sentence. Discourse knowledge is important for interpreting pronouns and temporal aspects of the information conveyed.

World Knowledge

Word knowledge is nothing but everyday knowledge that all speakers share about the world. It includes the general knowledge about the structure of the world and what each language user must know about the other user's beliefs and goals. This essential to make the language understanding much better.

There are several main techniques used in analysing natural language processing. Some of them can be briefly described as follows.

Pattern matching

The idea here is an approach to natural language processing is to interpret input utterances as a whole father than building up their interpretation by combining the structure and meaning of words or other lower level constituents. That means the interpretations are obtained by matching patterns of words against the input utterance. For a deep level of analysis in pattern matching a large number of patterns are required even for a restricted domain. This problem can be ameliorated by hierarchical pattern matching in which the input is gradually canonicalized through pattern matching against subphrases. Another way to reduce the number of patterns is by matching with semantic primitives instead of words.

Syntactically driven Parsing

Syntax means ways that words can fit together to form higher level units such as phrases, clauses and sentences. Therefore syntactically driven parsing means interpretation of larger groups of words are built up out of the interpretation of their syntactic constituent words or phrases. In a way this is the opposite of pattern matching as here the interpretation of the input is done as a whole. Syntactic analyses are obtained by application of a grammar that determines what scentences are legal in the language that is being parsed.

Semantic Grammars

Natural language analysis based on semantic grammar is bit similar to syntactically driven parsing except that in semantic grammar the categories used are defined semantically and syntactically. There here semantic grammar is also involved.

Case frame instantiation

Case frame instantiation is one of the major parsing techniques under active research today. The has some very useful computational properties such as its recursive nature and its ability to combine bottom-up recognition of key constituents with top-down instantiation of less structured constituents.

Ambiguity in NLP

https://youtu.be/-vBxzjx9AYA?si=RbVAcnfMeUMhAOwg

Ambiguity, generally used in natural language processing, can be referred as the ability of being understood in more than one way. In simple terms, we can say that ambiguity is the capability of being understood in more than one way. Natural language is very ambiguous. NLP has the following types of ambiguities –

Lexical Ambiguity

The ambiguity of a single word is called lexical ambiguity. For example, treating the word **silver** as a noun, an adjective, or a verb.

Syntactic Ambiguity

This kind of ambiguity occurs when a sentence is parsed in different ways. For example, the sentence “The man saw the girl with the telescope”. It is ambiguous whether the man saw the girl carrying a telescope or he saw her through his telescope.

Semantic Ambiguity

This kind of ambiguity occurs when the meaning of the words themselves can be misinterpreted. In other words, semantic ambiguity happens when a sentence contains an ambiguous word or phrase. For example, the sentence “The car hit the pole while it was moving” is having semantic ambiguity because the interpretations can be “The car, while moving, hit the pole” and “The car hit the pole while the pole was moving”.

Anaphoric Ambiguity

This kind of ambiguity arises due to the use of anaphora entities in discourse. For example, the horse ran up the hill. It was very steep. It soon got tired. Here, the anaphoric reference of “it” in two situations cause ambiguity.

Pragmatic ambiguity

Such kind of ambiguity refers to the situation where the context of a phrase gives it multiple interpretations. In simple words, we can say that pragmatic ambiguity arises when the statement is not specific. For example, the sentence “I like you too” can have multiple interpretations like I like you (just like you like me), I like you (just like someone else dose).

Applications of NLP

https://youtu.be/2c5yEybIaK4?si=w0HVPaSaFAcGLBXn

Natural Language Processing is a part of artificial intelligence that aims to teach the human language with all its complexities to computers. This is so that machines can understand and interpret the human language to eventually understand human communication in a better way. Natural Language Processing is a cross among many different fields such as **artificial intelligence, computational linguistics, human-computer interaction**, etc. There are many different methods in NLP to understand human language which include statistical and machine learning methods. These involve breaking down human language into its most basic pieces and then understand how these pieces relate to each other and work together to create meanings in sentences.

And why is Natural Language Processing important, you wonder? Well, it allows computers to understand human language and then analyze huge amounts of language-based data in an unbiased way. This is very difficult for humans to accomplish. In addition to that, there are thousands of human languages in hundreds of dialects that are spoken in different ways by different ways. NLP helps resolve the ambiguities in language and creates structured data from a very complex, muddled, and unstructured source.

This is the reason that Natural Language Processing has many diverse applications these days in fields ranging from IT to telecommunications to academics. So, let's see these applications now.

Applications of Natural Language Processing

1. Chatbots

Chatbots are a form of artificial intelligence that are programmed to interact with humans in such a way that they sound like humans themselves. Depending on the complexity of the chatbots, they can either just respond to specific keywords or they can even hold full conversations that make it tough to distinguish them from humans. Chatbots are created using Natural Language Processing and Machine Learning, which means that they understand the complexities of the English language and find the actual meaning of the sentence and they also learn from their conversations with humans and become better with time. Chatbots work in two simple steps. First, they identify the meaning of the question asked and collect all the data from the user that may be required to answer the question. Then they answer the question appropriately.

2. Autocomplete in Search Engines

Have you noticed that search engines tend to guess what you are typing and automatically complete your sentences? For example, On typing "game" in Google, you may get further suggestions for "game of thrones", "game of life" or if you are interested in maths then "game theory". All these suggestions are provided using autocomplete that uses Natural Language Processing to guess what you want to ask. Search engines use their enormous data sets to analyze what their customers are probably typing when they enter particular words and suggest the most common possibilities. They use Natural Language Processing to make sense of these words and how they are interconnected to form different sentences.

3. Voice Assistants

These days voice assistants are all the rage! Whether its Siri, Alexa, or Google Assistant, almost everyone uses one of these to make calls, place reminders, schedule meetings, set alarms, surf the

internet, etc. These voice assistants have made life much easier. But how do they work? They use a complex combination of speech recognition, natural language understanding, and natural language processing to understand what humans are saying and then act on it. The long term goal of voice assistants is to become a bridge between humans and the internet and provide all manner of services based on just voice interaction. However, they are still a little far from that goal seeing as Siri still can't understand what you are saying sometimes!

4. Language Translator

Want to translate a text from English to Hindi but don't know Hindi? Well, Google Translate is the tool for you! While it's not exactly 100% accurate, it is still a great tool to convert text from one language to another. Google Translate and other translation tools as well as use Sequence to sequence modeling that is a technique in Natural Language Processing. It allows the algorithm to convert a sequence of words from one language to another which is translation. Earlier, language translators used Statistical machine translation (SMT) which meant they analyzed millions of documents that were already translated from one language to another (English to Hindi in this case) and then looked for the common patterns and basic vocabulary of the language. However, this method was not that accurate as compared to Sequence to sequence modeling.

5. Sentiment Analysis

Almost all the world is on social media these days! And companies can use sentiment analysis to understand how a particular type of user feels about a particular topic, product, etc. They can use natural language processing, computational linguistics, text analysis, etc. to understand the general sentiment of the users for their products and services and find out if the sentiment is good, bad, or neutral. Companies can use sentiment analysis in a lot of ways such as to find out the emotions of their target audience, to understand product reviews, to gauge their brand sentiment, etc. And not just private companies, even governments use sentiment analysis to find popular opinion and also catch out any threats to the security of the nation.

6. Grammar Checkers

Grammar and spelling is a very important factor while writing professional reports for your superiors even assignments for your lecturers. After all, having major errors may get you fired or failed! That's why grammar and spell checkers are a very important tool for any professional writer. They can not only correct grammar and check spellings but also suggest better synonyms and improve the overall readability of your content. And guess what, they utilize natural language processing to provide the best possible piece of writing! The NLP algorithm is trained on millions of sentences to understand the correct format. That is why it can suggest the correct verb tense, a better synonym, or a clearer sentence structure than what you have written. Some of the most popular grammar checkers that use NLP include Grammarly, WhiteSmoke, ProWritingAid, etc.

7. Email Classification and Filtering

Emails are still the most important method for professional communication. However, all of us still get thousands of promotional Emails that we don't want to read. Thankfully, our emails are automatically divided into 3 sections namely, Primary, Social, and Promotions which means we never have to open the Promotional section! But how does this work? Email services use natural language processing to identify the contents of each Email with text classification so that it can be put in the correct section. This method is not perfect since there are still some Promotional newsletters in Primary, but its better than nothing. In more advanced cases, some companies also use specialty

anti-virus software with natural language processing to scan the Emails and see if there are any patterns and phrases that may indicate a phishing attempt on the employees.

Challenges of NLP

https://monkeylearn.com/blog/natural-language-processing-challenges/

https://content.techgig.com/technology-guide/5-challenges-in-natural-language-processing-to-watch-out-for/articleshow/86867982.cms

Tokenization

https://www.kaggle.com/code/satishgunjal/tokenization-in-nlp

Stemming

https://www.geeksforgeeks.org/introduction-to-stemming/

Segmentation

https://youtu.be/9LXq3oQEEIA?si=nta4tVSk88TEZ9Nc

Lemmatization

https://youtu.be/JpxCt3kvbLk?si=5F8qC6ImEf1NxYe0

Lemmatization is one of the most common text pre-processing techniques used in natural language processing (NLP) and machine learning in general. Lemmatization is not that much different than the stemming of words in NLP. In both stemming and lemmatization, we try to reduce a given word to its root word. The root word is called a stem in the stemming process, and it's called a lemma in the lemmatization process. But there are a few more differences to the two than that. Let's see what those are.

WHAT IS LEMMATIZATION?

Lemmatization is a text pre-processing technique used in natural language processing (NLP) models to break a word down to its root meaning to identify similarities. For example, a lemmatization algorithm would reduce the word *better* to its root word, or lemme, *good*.

How Is Lemmatization Different From Stemming?

In stemming, a part of the word is just chopped off at the tail end to arrive at the stem of the word. There are different algorithms used to find out how many characters have to be chopped off, but the algorithms don't actually know the meaning of the word in the language it belongs to. In lemmatization, the algorithms do have this knowledge. In fact, you can even say that these algorithms refer to a dictionary to understand the meaning of the word before reducing it to its root word, or lemma.

So, a lemmatization algorithm would know that the word *better* is derived from the word *good*, and hence, the lemme is *good*. But a stemming algorithm wouldn't be able to do the same. There could be over-stemming or under-stemming, and the word *better* could be reduced to either *bet*, or *bett*, or just retained as *better*. But there is no way in stemming that can reduce better to its root word *good*. This is the difference between stemming and lemmatization.

Minimum Edit Distance

https://youtu.be/YDguZretJz8?si=smmDHSjgihJPK7IJ

https://youtu.be/GrUx31RqyeY?si=AmnGFeZ0xTjHp3Jy

Collocations

Collocations are phrases or expressions containing multiple words, that are highly likely to co-occur. For example — 'social media', 'school holiday', 'machine learning', 'Universal Studios Singapore', etc.

Why do you need Collocations?

Imagine, having a requirement wherein you want to understand the text reviews left by your customers. You want to understand the behavioural insights like who are your customers, how many of them visit your place, what are they interested in, what do they buy, what activities do they engage with, etc.

For more simplicity, let's consider that you have a restaurant and you have several thousand reviews. Thus, as a restaurant owner you need to understand the behavioural insights of your customers, as discussed above.

Using Named Entity Recognition, I extracted certain interesting entities in the PERSON, EVENT, DATE, PRODUCT categories. Such as, 'Saturday' in DATE. I then wanted to find out what people are writing around 'Saturday' in their reviews!

Thus, I narrowed down on several such broad themes such as 'family', 'couple', 'holiday', 'brunch', etc. Collocations helped me in fetching the two or three words that are highly likely to co-occur around these themes. These two or three words that occur together are also known as BiGram and TriGram.

How is Collocations different than regular BiGrams or TriGrams?

The set of two words that co-occur as BiGrams, and the set of three words that co-occur as TriGrams, may not give us meaningful phrases. For example, the sentence 'He applied machine learning'

contains bigrams: ‘He applied’, ‘applied machine’, ‘machine learning’. ‘He applied’ and ‘applied machine’ do not mean anything, while ‘machine learning’ is a meaningful bigram. Just considering co-occurring words may not be a good idea, since phrases such as ‘of the’ may co-occur frequently, but are actually not meaningful. Thus, the need for collocations from NLTK library. It only gives us the meaningful BiGrams and TriGrams.

How is one Collocation better than the other?

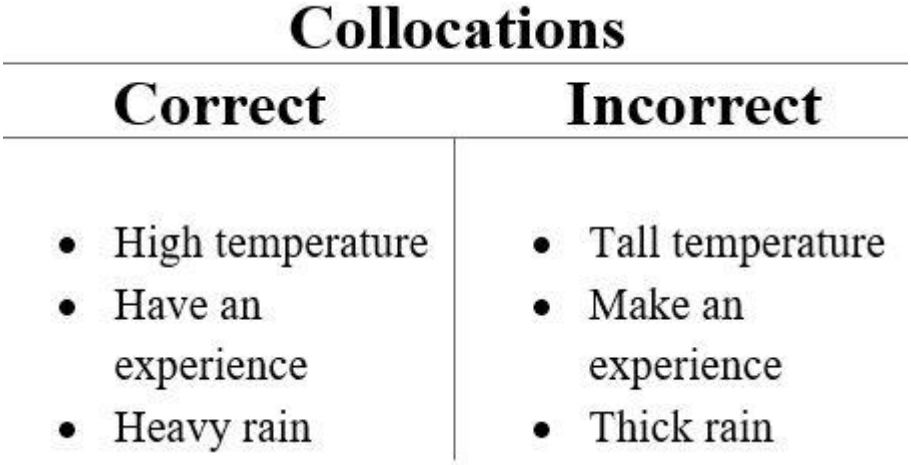
Oh! So you basically want to know how the scoring works? Well, I used Pointwise Mutual Information or PMI score. Discussing what’s PMI and how is it computed is not the scope of this blog, but here are some great articles which you can read to understand more: Article 1 and Article 2. I used the PMI scores to quantify and rank the BiGrams, TriGrams churned out by Collocations library.

How to implement Collocations?

As I mentioned earlier, I wanted to find out what do people write around certain themes such as some particular dates or events or person. So, from my code you will be able to see BiGrams, TriGrams around specific words. That is, I want to know BiGrams, TriGrams that are highly likely to formulate besides a ‘specific word’ of my choice. That specific word is nothing but the theme that we got from Named Entity Recognition.

Collocations are two or more words that tend to appear frequently together, for example – *United States*. There are many other words that can come after United, such as the United Kingdom and United Airlines. As with many aspects of natural language processing, context is very important. And for collocations, context is everything. In the case of collocations, the

context will be a document in the form of a list of words. Discovering collocations in this list of words means to find common phrases that occur frequently throughout the



text.

Finite Automata

https://youtu.be/5ZAQ6gz5sxw?si=EjnlLMRvjP2bFJ_G

The term automata, derived from the Greek word "αὐτόματα" meaning "self-acting", is the plural of automaton which may be defined as an abstract self-propelled computing device that follows a predetermined sequence of operations automatically.

An automaton having a finite number of states is called a Finite Automaton (FA) or Finite State automata (FSA).

Mathematically, an automaton can be represented by a 5-tuple (Q, ∑, δ, q0, F), where –

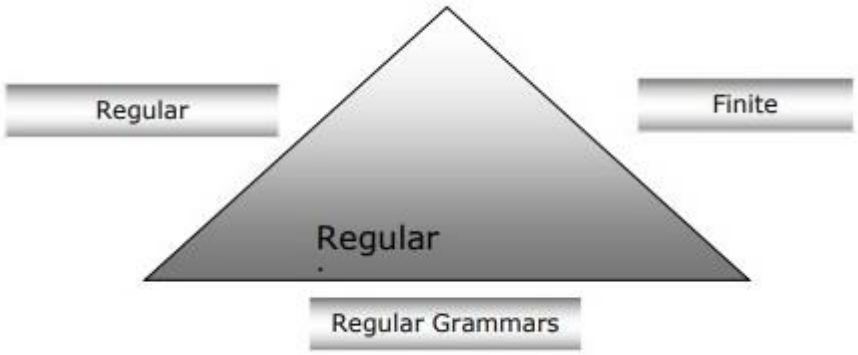
- Q is a finite set of states.
- ∑ is a finite set of symbols, called the alphabet of the automaton.
- δ is the transition function
- q0 is the initial state from where any input is processed (q0 ∈ Q).
- F is a set of final state/states of Q (F ⊆ Q).

Relation between Finite Automata, Regular Grammars and Regular Expressions

Following points will give us a clear view about the relationship between finite automata, regular grammars and regular expressions –

- As we know that finite state automata are the theoretical foundation of computational work and regular expressions is one way of describing them.
- We can say that any regular expression can be implemented as FSA and any FSA can be described with a regular expression.
- On the other hand, regular expression is a way to characterize a kind of language called regular language. Hence, we can say that regular language can be described with the help of both FSA and regular expression.
- Regular grammar, a formal grammar that can be right-regular or left-regular, is another way to characterize regular language.

Following diagram shows that finite automata, regular expressions and regular grammars are the equivalent ways of describing regular languages.



Types of Finite State Automation (FSA)

Finite state automation is of two types. Let us see what the types are.

Deterministic Finite automation (DFA)

It may be defined as the type of finite automation wherein, for every input symbol we can determine the state to which the machine will move. It has a finite number of states that is why the machine is called Deterministic Finite Automaton (DFA).

Mathematically, a DFA can be represented by a 5-tuple (Q, ∑, δ, q0, F), where –

- Q is a finite set of states.
- ∑ is a finite set of symbols, called the alphabet of the automaton.
- δ is the transition function where δ: Q × ∑ → Q .
- q0 is the initial state from where any input is processed (q0 ∈ Q).
- F is a set of final state/states of Q (F ⊆ Q).

Whereas graphically, a DFA can be represented by diagraphs called state diagrams where –

- The states are represented by **vertices**.
- The transitions are shown by labeled **arcs**.
- The initial state is represented by an **empty incoming arc**.
- The final state is represented by **double circle**.

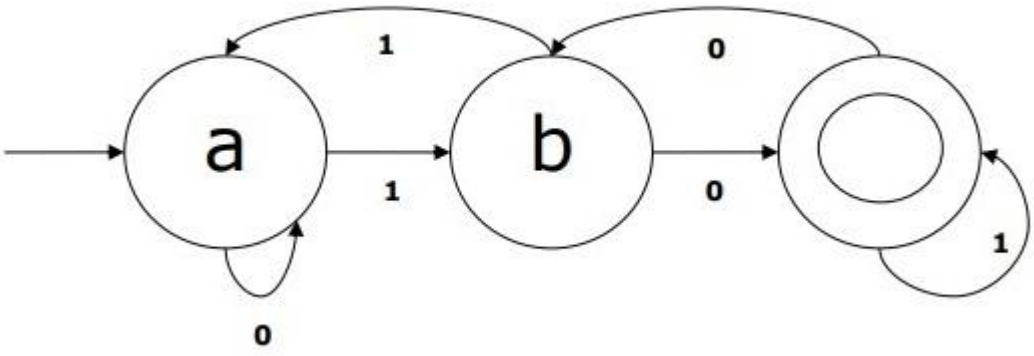
Example of DFA

Suppose a DFA be

- Q = {a, b, c},
- ∑ = {0, 1},
- q_0 = {a},
- F = {c},
- Transition function δ is shown in the table as follows –

Current State	Next State for Input 0	Next State for Input 1
A	a	B
B	b	A
C	c	C

The graphical representation of this DFA would be as follows –



Non-deterministic Finite Automation (NDFA)

It may be defined as the type of finite automation where for every input symbol we cannot determine the state to which the machine will move i.e. the machine can move to any combination of the states. It has a finite number of states that is why the machine is called Non-deterministic Finite Automation (NDFA).

Mathematically, NDFA can be represented by a 5-tuple (Q, ∑, δ, q0, F), where –

- Q is a finite set of states.
- ∑ is a finite set of symbols, called the alphabet of the automaton.
- δ :-is the transition function where δ: Q × ∑ → 2^{Q}.

- q0 :-is the initial state from where any input is processed (q0 ∈ Q).
- F :-is a set of final state/states of Q (F ⊆ Q).

Whereas graphically (same as DFA), a NDFA can be represented by diagraphs called state diagrams where –

- The states are represented by **vertices**.
- The transitions are shown by labeled **arcs**.
- The initial state is represented by an **empty incoming arc**.
- The final state is represented by double **circle**.

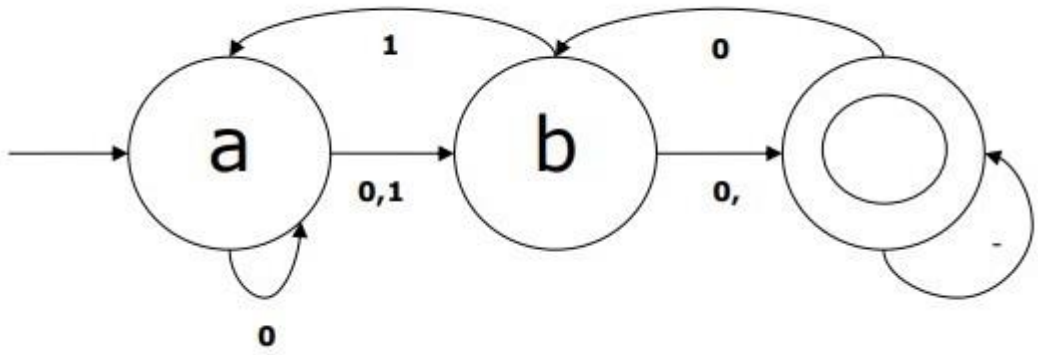
Example of NDFA

Suppose a NDFA be

- Q = {a, b, c},
- ∑ = {0, 1},
- q0 = {a},
- F = {c},
- Transition function δ is shown in the table as follows –

Current State	Next State for Input 0	Next State for Input 1
A	a, b	B
B	C	a, c
C	b, c	C

The graphical representation of this NDFA would be as follows –



Finite state transducer

A Finite State Transducer (FST) is a computational model used in various fields, including Natural Language Processing (NLP), to represent and process finite sets of strings or sequences. In NLP, FSTs are particularly valuable for tasks

involving regular or structured transformations of text, such as tokenization, stemming, morphological analysis, and spelling correction. Here's an overview of Finite State Transducers in the context of NLP:

1. **Definition:** An FST is a mathematical model consisting of a finite set of states, a finite set of input symbols (often characters or tokens), a finite set of output symbols (typically corresponding to transformations or labels), a transition function that defines state transitions based on input symbols, and a set of initial and final states.
2. **Key Components:**
    - **States:** Each state represents a specific point in the computation.
    - **Input Alphabet:** The set of allowable input symbols or characters.
    - **Output Alphabet:** The set of allowable output symbols or labels.
    - **Transition Function:** Defines how the FST transitions between states based on input symbols and produces output symbols.
    - **Initial and Final States:** States where the FST starts and ends.
3. **Applications in NLP:**
    - **Tokenization:** FSTs can be used to tokenize text, i.e., split it into words or subword units. For example, you can create an FST that recognizes spaces or punctuation as token delimiters.
    - **Stemming:** Stemming is the process of reducing words to their base or root form. FSTs can be designed to perform stemming by mapping different word forms to their common root.
    - **Morphological Analysis:** FSTs are used in languages with complex morphology to analyze and generate word forms based on grammatical rules and affixes.
    - **Spelling Correction:** FSTs can correct misspelled words by generating possible corrections based on edit distance or other similarity measures.
    - **Part-of-Speech Tagging:** FSTs can be used to map words to their corresponding part-of-speech tags.
4. **Properties:**

- **Determinism:** An FST is deterministic if, for each state and input symbol, there is at most one transition. Deterministic FSTs are often preferred for their efficiency.
- **Non-determinism:** Non-deterministic FSTs may have multiple possible transitions for a state and input symbol. They can be used for certain tasks but are generally less efficient.
- **Weighted FSTs:** Some FSTs use weights on transitions to represent probabilities or costs associated with particular transformations.

5. **Tools and Libraries:** Various NLP libraries and frameworks provide tools for working with FSTs, such as OpenFST, HFST (Helsinki Finite-State Technology), and the Python library Pynini.

In summary, Finite State Transducers are versatile computational models used in NLP for tasks that involve regular or structured text transformations. They are particularly well-suited for tokenization, stemming, morphological analysis, spelling correction, and other tasks where sequence transformations are required.

2.2 Finite State Transducers

We will now introduce *finite state transducers* (or FSTs); another finite state machine that allows to produce output recording the structure of the input.

2.2.1 What are Finite State Transducers?

A finite state transducer essentially is a finite state automaton that works on two (or more) tapes. The most common way to think about transducers is as a kind of ``translating machine''. They read from one of the tapes and write onto the other. This, for instance, is a transducer that translates `a`s into `b`s:

`a:b` at the arc means that in this transition the transducer reads `a` from the first tape and writes `b` onto the second.

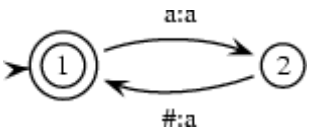
Transducers can, however, be used in other modes than the translation mode as well: in the generation mode transducers write on both tapes and in the recognition mode they read from both tapes. Furthermore, the direction

of translation can be turned around: i.e. `a:b` can not only be read as ``read `a` from the first tape and write `b` onto the second tape'', but also as ``read `b` from the second tape and write `a` onto the first tape''.

So, the above transducer behaves as follows in the different modes.

- generation mode: It writes a string of `a`s on one tape and a string `b`s on the other tape. Both strings have the same length.
- recognition mode: It accepts when the word on the first tape consists of exactly as many `a`s as the word on the second tape consists of `b`s.
- translation mode (left to right): It reads `a`s from the first tape and writes an `b` for every `a` that it reads onto the second tape.
- translation mode (right to left): It reads `b`s from the second tape and writes an `a` for every `f` that it reads onto the first tape.

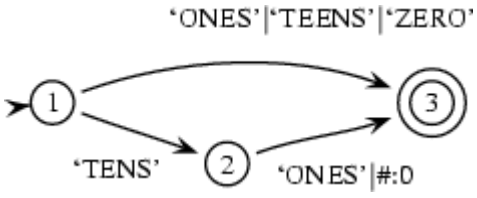
Transitions in transducers can make jumps going from one state to another without doing anything on either one or on both of the tapes. So, transitions of the form `a:#` or `#:a` or `#:#` are possible. Here is an example:



And what does this transducer do?

- generation mode: It writes twice as many `a`s onto the second tape as onto the first one.
- recognition mode: It accepts when the second tape has twice as many `a`s as the first one.
- translation mode (left to right): It reads `a`s from the first tape and writes twice as many onto the second tape.
- translation mode (right to left): It reads `a`s from the second tape and writes half as many onto the first one.

Similar as with FSAs, we can also use categories to label the arcs and provide a kind of lexicon which translates these categories into real labels, i.e. labels of the form `X:Y`. Here is an example translating English number terms into numbers.



And here is the lexicon that maps the category labels to standard FST transition labels:

```
lex(one:1,`ONES').
lex(two:2,`ONES').
lex(three:3,`ONES').
lex(four:4,`ONES').
lex(five:5,`ONES').
lex(six:6,`ONES').
lex(seven:7,`ONES').
lex(eight:8,`ONES').
lex(nine:9,`ONES').
lex(eleven:11,`TEENS').
lex(twelve:12,`TEENS').
...
lex(twenty:20,`TENS').
...
lex(zero:0,`ZERO').
```

2.2.2 FSTs in Prolog

In implementing finite state transducers in Prolog, we will follow the same strategy that we used for FSAs: we represent an FST as a static data structure that other programs manipulate.

Here is how we represent our first transducer, the `a` to `b` translator.

```
:- op(250,xfx,:).

initial(1).
final(1).
arc(1,1,a:b).
```

To be able to write `a:b` as the label of the arc, we have to define `:` as an infix operator as is done by the operator definition.

Our second transducer, the `a` doubler, looks like this in Prolog representation:

```
:- op(250,xfx,:).

initial(1).
final(1).
arc(1,2,a:a).
arc(2,1,#:a).
```

Now, we need a program that can manipulate these data structures and carry out the transduction. We will first assume that there are no jump arcs and extend `recognize1` from the last chapter to work as a transducer.

The base case is that both tapes are empty and the FSA is in a final state. In Prolog:

```
transduce1(Node,[],[]) :-
    final(Node).
```

In the recursive case, we make a transition from one node to another which is licensed by some `arc` definition in the database. As in the last chapter we define a predicate `traverse1` to check that the transition is indeed licensed.

```
transduce1(Node1,Tape1,Tape2) :-
    arc(Node1,Node2,Label),
    traverse1(Label,Tape1,NewTape1,Tape2,NewTape2),
    transduce1(Node2,NewTape1,NewTape2).

traverse1(L1:L2,[L1|RestTape1],RestTape1,[L2|RestTape2],RestTape2).
```

Finally, we define the following driver predicate `testtrans1`. It can be called with both arguements instantiated, only one of them instantiated, or both uninstantiated depending on which mode we want to use the transducer in.

```
testtrans1(Tape1,Tape2) :-
    initial(Node),
    transduce1(Node,Tape1,Tape2).
```

We can use this program to transduce `a`s to `b`s with our first transducer. To be able to use the second transducer, the `a` doubler, as well, we need a program that can handle transitions involving jumps. What do we have to change for that? Well, the only thing that changes is the way that the tapes are treated when making a transition. This is taken care of by the `traverse1` predicate and this is all we have to adapt. (Remember that when extending the recognizer of the last chapter to handle jump arcs, we also only changed the `traverse1` predicate.)

So, what are the possibilites how a tape can be affected by a transition? There are four:

- We have to jump on both tapes.
- We have to jump on the first but not on the second tape.
- We have to jump on the second but not on the first tape.
- We have to jump on neither tape (this is what `traverse1` does).

The Prolog definition of `traverse2` therefore has four clauses:

```
traverse2('#':'#',Tape1,Tape1,Tape2,Tape2).
traverse2('#':L2,Tape1,Tape1,[L2|RestTape2],RestTape2).
traverse2(L1:'#',[L1|RestTape1],RestTape1,Tape2,Tape2).
traverse2(L1:L2,[L1|RestTape1],RestTape1,[L2|RestTape2],RestTape2).
```

refer ppt

Porter Stemmer

Morphology

https://youtu.be/ejqzMfyP6OI?si=TG3XKTtkXkWZIO2N

https://youtu.be/b5_u52GoPpE?si=MRZI3jtdpyL9WkSm

POS Tagging

N-gram model

N-gram is a sequence of the N-words in the modeling of NLP. Consider an example of the statement for modeling. “I love reading history books and watching documentaries”. In one-gram or unigram, there is a one-word sequence. As for the above statement, in one gram it can be “I”, “love”, “history”, “books”, “and”, “watching”, “documentaries”. In two-gram or the bi-gram, there is the two-word sequence i.e. “I love”, “love reading”, or “history books”. In the three-gram or the tri-gram, there are the three words sequences i.e. “I love reading”, “history books,” or “and watching documentaries” [3]. The illustration of the N-gram modeling i.e. for N=1,2,3 is given below in Figure 2 [5].

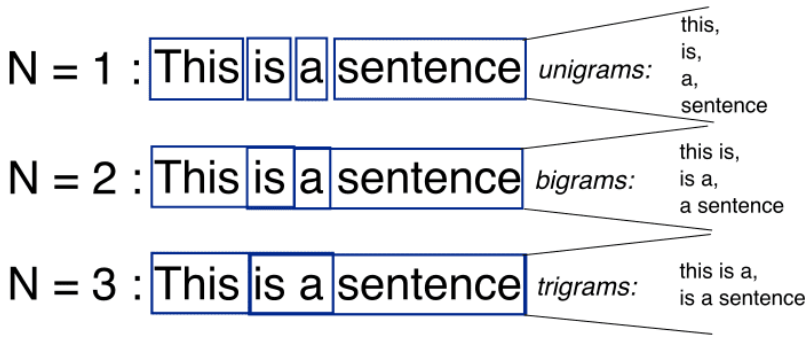


Figure 2 Uni-gram, Bi-gram, and Tri-gram Model

For N-1 words, the N-gram modeling predicts most occurred words that can follow the sequences. The model is the probabilistic language model which is trained on the collection of the text. This model is useful in applications i.e. speech recognition, and machine translations. A simple model has some limitations that can be improved by smoothing, interpolations, and back off. So, the N-gram language model is about finding probability distributions over the sequences of the word. Consider the sentences i.e. "There was heavy rain" and "There was heavy flood". By using experience, it can be said that the first statement is good. The N-gram language model tells that the "heavy rain" occurs more frequently than the "heavy flood". So, the first statement is more likely to occur and it will be then selected by this model. In the one-gram model, the model usually relies on that which word occurs often without pondering the previous words. In 2-gram, only the previous word is considered for predicting the current word. In 3-gram, two previous words are considered. In the N-gram language model the following probabilities are calculated:

P (“There was heavy rain”) = P (“There”, “was”, “heavy”, “rain”) = P (“There”) P (“was” |“There”) P (“heavy”| “There was”) P (“rain” |“There was heavy”).

As it is not practical to calculate the conditional probability but by using the “*Markov Assumptions”*, this is approximated to the bi-gram model as [4]:

P (“There was heavy rain”) ~ P (“There”) P (“was” |“'There”) P (“heavy” |“was”) P (“rain” |“heavy”)

Applications of the N-gram Model in NLP

In speech recognition, the input can be noisy. This noise can make a wrong speech to the text conversion. The N-gram language model corrects the noise by using probability knowledge. Likewise, this model is used in machine translations for producing more natural statements in target and specified languages. For spelling error corrections, the dictionary is useless sometimes. For instance, "in about fifteen minutes" 'minuets' is a valid word according to the dictionary but it is incorrect in the phrase. The N-gram language model can rectify this type of error.

The N-gram language model is generally at the word levels. It is also used at the character levels for doing the stemming i.e. for separating the root words from a suffix. By looking at the N-gram model, the languages can be classified or differentiated between the US and UK spellings. Many applications get benefit from the N-gram model including tagging of part of the speech, natural language generations, word similarities, and sentiments extraction. [4].

Limitations of N-gram Model in NLP

The N-gram language model has also some limitations. There is a problem with the out of vocabulary words. These words are during the testing but not in the training. One solution is to use the fixed vocabulary and then convert out vocabulary words in the training to pseudowords. When implemented in the sentiment analysis, the bi-gram model outperformed the uni-gram model but the number of the features is then doubled. So, the scaling of the N-gram model to the larger data sets or moving to the higher-order needs better feature selection approaches. The N-gram model captures the long-distance context poorly. It has been shown after every 6-grams, the gain of performance is limited.