



ST. ANNE'S COLLEGE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE, New Delhi. Affiliated to Anna University, Chennai)

Accredited by NAAC

ANGUCHETTYPALAYAM, PANRUTI – 607 106.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Subject Name : NATURAL LANGUAGE PROCESSING
SCIENCE

Subject code : AL3501

Regulation : 2021

Year / Branch : III / AIML

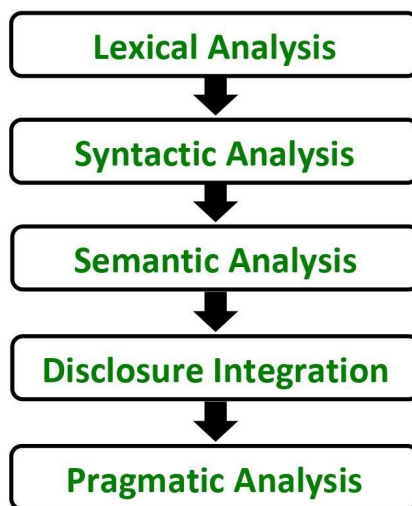
Semester : V

Faculty Name : Mr. R. MANICKAVASAGAN,
AP/AIML

UNIT I

The five general steps in Natural Language Processing (NLP) are: Lexical Analysis, Syntactic Analysis (Parsing), Semantic Analysis, Discourse Integration, and Pragmatic Analysis. These steps break down text into smaller parts, analyze grammatical structure, determine meaning, consider context, and understand the intended effect of the language.

Here's a more detailed look at each step:



1. **Lexical Analysis:**

This initial step involves breaking down the input text into smaller units called tokens, such as words or punctuation marks.

2. **Syntactic Analysis (Parsing):**

Syntactic analysis focuses on the grammatical structure of the sentence, ensuring it follows the rules of grammar.

3. **Semantic Analysis:**

This step determines the literal meaning of the words and sentences, considering their individual definitions and how they relate to each other.

4. **Discourse Integration:**

Discourse integration considers the context of the surrounding text, including previous sentences, to understand the overall meaning.

5. **Pragmatic Analysis:**

This final step focuses on the intended effect of the language, considering the context, the speaker's goals, and other factors to understand the meaning beyond the literal words.

Origins and challenges of Natural Language Processing (NLP)

Natural Language Processing (NLP), a key subfield of Artificial Intelligence, focuses on enabling computers to understand, interpret, and generate human language. The field has a rich history spanning decades, evolving from its early beginnings to the advanced models of today.

Origins of NLP

- **Early Beginnings (1950s-1960s):** NLP emerged as researchers such as Alan Turing explored the possibility of machines thinking and communicating, leading to the concept of the Turing Test. Early efforts focused on machine translation. The goal was to convert text from one language to another automatically, particularly from Russian to English during the Cold War. Rule-based systems, carefully crafted by linguists, were initially used to interpret and generate language based on grammatical rules according to Learn Electronics with me. ELIZA and SHRDLU are examples from this era.
- **Theoretical Foundations:** Noam Chomsky's work on transformational grammar provided a framework for analyzing language structure and influenced early computational models.
- **Statistical Revolution (1980s-1990s):** Limitations of rule-based systems, such as inflexibility and scalability issues, became apparent. This led to a shift towards statistical methods, using increased computational power and the availability of large text datasets (corpora). Probabilistic models like Hidden Markov Models (HMMs) and n-gram models became important for speech recognition and language modeling tasks.
- **Machine Learning and Deep Learning Era (2000s-Present):** Machine learning techniques further revolutionized NLP, with algorithms learning from data to improve tasks like text classification and named entity recognition. The rise of deep learning and neural network architectures, such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, enhanced understanding of context and sequence in language.

- **Transformer Models (2017-Present):** The introduction of the Transformer architecture marked a major breakthrough, enabling parallel processing of sequential data and outperforming previous models in various tasks. Models like BERT and GPT, based on the Transformer architecture, have achieved unprecedented performance in language understanding and generation tasks.

Challenges in NLP

- **Ambiguity and Context:** Human language is inherently ambiguous, with words and phrases having multiple meanings depending on context. Accurately interpreting the intended meaning remains a key challenge for NLP systems.
- **Data Quality and Availability:** Training robust NLP models requires large, high-quality, and diverse datasets. Obtaining labeled data can be costly and time-consuming, and some domains lack sufficient data.
- **Multilingualism and Language Variations:** NLP systems need to handle various languages, dialects, slang, and grammatical irregularities, which pose significant challenges for developing universally effective solutions.
- **Bias and Fairness:** NLP models, trained on datasets reflecting societal biases, can inadvertently perpetuate these biases, leading to unfair or discriminatory outcomes in various applications like resume screening or loan applications.
- **Semantic Understanding and Reasoning:** Understanding the deeper meaning, nuanced language (like sarcasm or irony), and performing commonsense reasoning remains difficult for machines.
- **Scalability and Performance:** Developing scalable NLP solutions that can efficiently handle large datasets and complex computations while maintaining high performance is a significant challenge.
- **Noise and Uncertainty:** Real-world language data often contains errors, misspellings, and grammatical inconsistencies, making it challenging for NLP systems to maintain accuracy and reliability.
- **Interpretability and Explainability:** As NLP models become more complex (like deep learning models), understanding their decision-making process becomes harder. Ensuring transparency and the ability to explain model outputs is crucial for building trust.
- **Ethical Considerations:** Ensuring the responsible and ethical use of NLP, especially regarding data privacy, security, potential misuse, and impact on society, is paramount. In

conclusion, NLP has come a long way from its rule-based beginnings but still faces numerous challenges related to the inherent complexities of human language. Addressing these challenges is vital for building more accurate, fair, and responsible NLP systems in the future.

Language Modeling: Grammar-based LM

Thinking

Kicking off 8 searches

Looking at 114 sites

Putting it all together

Grammar-based language models (LMs) were an important early approach to natural language processing (NLP) that relied on formal grammar rules to understand and generate language.

Here's how they worked:

Rule-Based Approach: These models use predefined sets of grammatical rules and syntactic structures (like nouns, verbs, phrases) and rules such as subject-verb agreement to determine valid sentences in a language.

Parsing and Generation: They use parsing to analyze sentences and ensure they adhere to the existing linguistic rules, and can also be used for generating grammatically correct sentences. For example, a grammar-based model might use a context-free grammar with rules like:

- Sentence →right arrow→Noun Phrase Verb Phrase
 - Noun Phrase →right arrow→Determiner Noun
 - Verb Phrase →right arrow→Verb Noun Phrase
 - Determiner →right arrow→"the" | "a"
 - Noun →right arrow→"cat" | "dog" | "treat"
 - Verb →right arrow→"chased" | "ate"
- Using these rules, sentences like “the dog chased the cat” can be generated.

- **Historical Examples:** Early grammar-based LMs include ELIZA, SHRDLU, and LUNAR.

Advantages

- **Grammatical Correctness:** They excel at generating grammatically correct sentences.
- **Structure Capture:** These models can effectively capture the underlying structure of a sentence.
- **Accuracy in Specific Contexts:** They provided very accurate results in specific contexts in the early stages of NLP.

Limitations

- **Rigidity:** Grammar-based models are often rigid and struggle with the natural variations, ambiguities, and complexities of language, including less common phrases or new slang not explicitly covered by the rules.

- **Scalability Challenges:** It's difficult to scale these models to handle large-scale data and complex language structures because manually implementing all language patterns is practically impossible.
- **Ambiguity in Parsing:** Context-free grammars, commonly used in grammar-based models, are often ambiguous and can produce more than a single parse tree for a sentence.

Shift towards Statistical and Neural Models

Due to these limitations, especially in handling data sparsity and the complexities of natural language, the field of language modeling largely shifted towards statistical and later neural network-based approaches. These data-driven models learn patterns and relationships directly from large text corpora, overcoming many of the rigidity and scalability issues inherent in rule-based systems.

In essence, while grammar-based LMs provided foundational insights into language structure and were important in the early development of NLP, their rule-based nature limited their robustness and scalability. This paved the way for the emergence of statistical and, later, neural language models that revolutionized the field by learning directly from data.

Regular expressions in statistical language modeling

Regular expressions (regex) are a sequence of characters that define a search pattern within text. While not a complete language modeling technique on their own, regex play a crucial role in the preprocessing and feature engineering steps for statistical language models. Here's how regex relates to statistical language modeling:

1. Text preprocessing and cleaning

- **Tokenization:** Regex can be used to define word boundaries and split a given text into individual words or tokens, forming the basis for statistical analysis. For example, `\s+` can be used to split text based on whitespace characters.
- **Removing special characters and punctuation:** Cleaning text data is essential for statistical models, and regex helps remove unwanted characters, symbols, or formatting. The pattern `[^\w\s]` matches any non-alphanumeric or non-whitespace characters, effectively removing punctuation and special characters.
- **Normalization:** Regex can be used to normalize text by converting different date formats into a standard one or standardizing spelling variations, which is important for aggregating data from diverse sources.

2. Feature engineering

- **Pattern Matching for specific structures:** Regex can identify specific patterns in text, allowing for the extraction of features relevant to the statistical model. For instance, `^b.*ed$` can identify words starting with 'b' and ending with 'ed'.

- **Named Entity Recognition (NER):** Regex can be used to search for patterns indicative of named entities, like names or titles. For example, a regex such as `[A-Z][a-z]+` can identify words starting with an uppercase letter followed by one or more lowercase letters, which is helpful in detecting potential proper nouns.

3. Data extraction and validation

- **Extracting specific information:** Regex can efficiently extract various types of data from unstructured text, such as phone numbers, email addresses, or URLs, which can then be used as features in a statistical language model.
- **Data validation:** Regex can validate if data adheres to a specific format or structure. For instance, in an email address, `\b[^\s@]+@\b` can check for a valid format, ensuring data integrity.

Limitations

While regex is valuable for preprocessing and feature engineering, they have limitations when it comes to directly building sophisticated language models:

- **Complexity:** As patterns become more complex, regex can be challenging to manage and debug.
- **Semantic understanding:** Regex operates purely on syntactic recognition and cannot resolve context-dependent ambiguities or understand the deeper meaning of words or phrases, according to one article.
- **Performance:** Highly complex regex patterns applied to large datasets can lead to performance issues due to backtracking.

In conclusion

Regex is a powerful and versatile tool for text processing in NLP, playing a critical role in data preprocessing, feature engineering, and data extraction for statistical language models. However, it's important to recognize their limitations and combine them with other NLP techniques like machine learning models for robust and accurate results, especially when dealing with complex language structures and semantic understanding.

Finite-state automata and English morphology

Finite-State Automata (FSA) play a significant role in analyzing English morphology, particularly in tasks like recognizing valid word forms.

What is an FSA?

An FSA is a mathematical model of computation that recognizes patterns in input sequences.

It consists of states and transitions between them, processing input symbols step-by-step.

If the automaton reaches an accepting state after processing the entire input, the input is considered valid or recognized.

Applying FSA to English morphology

- **Morphological Analysis:** FSAs are used in morphological parsing to break down words into their constituent morphemes (stems and affixes) according to SlideShare.
- **Representing Lexicons:** Large dictionaries and lexicons can be represented efficiently using FSAs, with states and transitions representing the valid sequences of morphemes and their associated features.
- **Regular Expressions:** FSA can represent regular languages, which are formally described by regular expressions. Regular expressions are commonly used to define patterns that match specific word structures, like the endings of plural nouns or verb tenses.
- **Building Morphological Analyzers:** FSAs are used to create the core components of morphological analyzers, including representing the lexicon, morphotactic rules (rules for combining morphemes), and orthographic rules (spelling changes).
- **Example: Pluralization:** An FSA could be designed to recognize plural forms in English. For instance, an input like "cats" would be parsed by an FSA that recognizes the stem "cat" and then the plural suffix "-s," mapping it to a representation like "cat +N +PL".

Advantages of FSA for morphological analysis

- **Rule-Based:** FSAs excel at applying predefined rules and patterns, making them suitable for handling regular morphological processes according to Medium.
- **Efficient Recognition:** FSAs can recognize valid word forms relatively quickly, performing recognition in linear time.
- **Clear Representation:** The visual representation of FSAs (state diagrams) provides a clear and intuitive way to understand the morphological rules being applied.

Limitations and shift to FSTs

- **Limited Output:** A basic FSA can only accept or reject an input string, meaning it can only determine if a word form is valid according to Lj Miranda. It cannot directly provide the underlying morphological analysis (stem, features).
- **Complexity with Irregularities:** Handling irregular forms (e.g., "went" as the past tense of "go") can require more intricate and complex FSAs or the use of finite-state transducers (FSTs).
- **Morphological Generation:** FSAs are not suited for generating word forms from their underlying representations. For this, FSTs are required.

Note: Finite-State Transducers (FSTs) are an extension of FSAs that map an input string to an output string. This allows FSTs to perform morphological parsing (input: "walked" -> output: "walk+V+PAST") and morphological generation (input: "walk+V+PAST" -> output: "walked"), addressing some limitations of basic FSAs in morphology tasks

Transducers for lexicon and rules,

Thinking

Kicking off 9 searches

Looking at 125 sites

Putting it all together

Transducers for representing lexicon and rules in NLP

Finite-state transducers (FSTs) are highly valuable tools in natural language processing (NLP) for both representing lexicons and implementing linguistic rules, particularly in areas like morphological analysis. They offer a compact and efficient way to handle the complexities of word structure and transformations.

1. Representing the lexicon with FSTs

- **Mapping Lexical Forms to Surface Forms:** An FST can be constructed to map the underlying lexical representation of a word (including its stem and grammatical features) to its surface form (the actual word as it appears in text). For instance, it can map "cat+N+PL" to "cats" or "walk+V+PAST" to "walked".
- **Structure of the FST:** The FST's states and transitions would be designed to capture the different parts of a word, like the stem and various prefixes or suffixes. Transitions are labeled with the lexical symbol (what's being added or removed) and the corresponding surface symbol. An epsilon (ϵ) may be used to represent an empty string, where a lexical symbol doesn't have a direct surface realization or vice-versa.
- **Efficiency:** Representing the lexicon as an FST is efficient because it avoids storing every possible word form explicitly. Instead, it stores the rules and patterns that generate those forms, significantly reducing memory usage.
-

2. Implementing rules with FSTs

- **Morphotactic Rules:** FSTs encode the morphotactic rules of a language. These rules determine the valid combinations of morphemes (stems and affixes). For example, an FST could ensure that a plural suffix "-s" is only attached to nouns and not to verbs.
- **Orthographic Rules (Spelling Changes):** FSTs handle orthographic rules that govern spelling changes when morphemes are combined. For instance, the rule for changing "y" to "i" before adding a suffix like "-es" (e.g., "city" → "cities") can be represented by a specific FST.
- **Phonological Rules:** FSTs also model phonological rules, which describe sound changes within words or across morpheme boundaries. For example, rules for assimilation or deletion of sounds can be captured by FSTs.
- **Context-sensitive Rewriting Rules:** FSTs can represent context-sensitive rewriting rules, which transform one string into another based on the surrounding context according to Wikipedia

3. Combining lexicon and rules with FSTs

- **Composition:** A key advantage of FSTs is their ability to be composed. This allows multiple FSTs, each representing a part of the lexicon or a specific rule, to be combined into a single, complex transducer.
- **Creating a Cascade:** In tasks like morphological parsing, a cascade of FSTs can be built, where the output of one FST becomes the input of the next according to SlideShare. For example, a word might first go through an FST for orthographic rules, then a lexicon FST to identify the stem and morphological features, and finally through FSTs for other linguistic processes.
- **Analysis and Generation:** Composing FSTs allows for both morphological analysis (mapping surface form to lexical representation) and morphological generation (mapping lexical representation to surface form), by simply reversing the composed FST.

In conclusion, FSTs provide a robust and efficient framework for representing linguistic knowledge, specifically the lexicon and rules that govern word structure and transformations, making them a cornerstone of computational morphology and other rule-based NLP applications.

Tokenization, detecting and correcting spelling errors, and minimum edit distance

These three concepts are fundamental to natural language processing (NLP), especially for tasks involving text normalization and understanding.

1. Tokenization

Definition: Tokenization is the process of breaking down a text into smaller units called tokens. These tokens can be words, sub words, characters, or even sentences, depending on the

chosen method according to Medium. It's typically the first step in preparing text for NLP analysis.

Importance: Tokenization simplifies text analysis and facilitates feature extraction for machine learning models. It helps in standardizing the input text, making it more manageable for algorithms to process according to Pickl.AI.

Methods: Common tokenization techniques include: according to Medium

Word tokenization: Splits text into individual words, working well for languages with clear word boundaries like English.

Character tokenization: Splits text into individual characters, useful for languages without clear word boundaries or for tasks like spelling correction.

Sub word tokenization: Divides words into meaningful parts or morphemes, helping handle rare or unknown words.

Sentence tokenization: Splits text into individual sentences, useful for tasks like summarization or translation.

Whitespace tokenization: Splits text at spaces, a simple and fast method but less precise for complex cases.

Rule-based tokenization (e.g., Regular Expressions): Uses predefined rules or patterns to split text.

Penn Treebank tokenization: A standard method for English that handles contractions, punctuation, and hyphenated words according to Geeks for Geeks.

Challenges: Tokenization faces challenges like ambiguity in language, handling out-of-vocabulary words, dealing with multilingual text, and special characters according to Medium. according to Debut Infotech.

2. Detecting and correcting spelling errors

Importance: Correcting spelling errors is vital for search engines, word processors, and other applications that rely on text input according to ResearchGate.

Methods:

Dictionary-based algorithms: Check input words against a dictionary of correctly spelled words. Words not found in the dictionary are considered potentially misspelled according to Scaler.

Edit distance: Calculate the minimum number of edits (insertions, deletions, or substitutions) needed to transform one word into another. Suggestions are then made based on words with the smallest edit distance from the misspelled word.

N-gram models: Compare character n-grams (sequences of characters of length n) to assess the similarity between words according to International Journal of Computer Applications | IJCA.

Deep learning methods: Use neural networks like recurrent neural networks (RNNs) or convolutional neural networks (CNNs) to learn the context and identify and correct spelling errors, including real-word errors (correctly spelled words used incorrectly) according to Nature.

Challenges: Context-free error correction algorithms have limitations and may fail to detect real-word errors according to Scaler.

3. Minimum edit distance (levenshtein distance)

Definition: Minimum edit distance, often referred to as Levenshtein distance, is a string metric that quantifies how dissimilar two strings are according to Wikipedia. It's calculated by determining the fewest number of single-character editing operations required to transform one string into the other.

Operations: The standard operations include:

Insertion: Adding a character.

Deletion: Removing a character.

Substitution: Replacing one character with another.

Calculation: The minimum edit distance is typically computed using dynamic programming, where a table is built up to store the edit distances between substrings according to SlideShare. Each cell in the table represents the minimum edit distance between the corresponding substrings, and the final value in the table represents the distance between the two original strings.

Applications: Minimum edit distance is widely used in NLP tasks such as:

Spell checking and correction: Identifying candidate corrections for misspelled words by selecting words from a dictionary that have a low edit distance to the misspelled word according to Wisdom ML.

Speech recognition: Comparing spoken words to potential matches.

Bioinformatics: Quantifying the similarity of DNA sequences.

Approximate string matching: Finding substrings that are within a certain edit distance of a given pattern.

These concepts are intricately linked. Tokenization provides the individual units for analysis, and spelling correction, aided by the minimum edit distance, ensures that the tokens are accurate, improving the overall quality and performance of NLP applications

Part A – 2 Marks Questions (Short answers)

1. What is a language model in NLP?

A language model assigns probabilities to sequences of words to predict the likelihood of a sentence in a language.

2. Define regular expressions.

Regular expressions are symbolic notations used to identify patterns in text, such as digits, letters, or specific sequences.

3. What is a Finite-State Automaton (FSA)?

FSA is a computational model consisting of states and transitions used to recognize regular languages and patterns.

4. Mention one challenge in statistical language modeling.

One key challenge is data sparsity—many word sequences may not appear in the training data.

5. What is tokenization in NLP?

Tokenization is the process of breaking text into individual units like words, subwords, or sentences.

6. Define morphology in the context of NLP.

Morphology is the study of word formation and structure, including prefixes, roots, and suffixes.

7. What is minimum edit distance?

It's the minimum number of operations (insertions, deletions, substitutions) required to transform one string into another.

8. What is a transducer in NLP?

A transducer is a finite-state machine that maps input strings to output strings, used in tasks like morphological analysis.

Part B – 16 Marks Question

Question:

Explain the origins and challenges of NLP. Discuss Language Modeling in detail including Grammar-based and Statistical LM. Explain Regular Expressions and Finite-State Automata. Describe English Morphology and how Transducers are used for lexicon and rules. Include processes like Tokenization, Detecting and Correcting Spelling Errors, and Minimum Edit Distance.

Answer:

1. Origins and Challenges of NLP:

Natural Language Processing (NLP) originated from computational linguistics, artificial intelligence, and machine learning, with the goal of enabling computers to understand,

interpret, and generate human language. Early NLP focused on rule-based systems, but over time statistical methods and deep learning took over.

Challenges in NLP include:

- **Ambiguity:** Words and sentences often have multiple meanings.
- **Context-dependency:** Meaning depends on context, which is hard for machines to interpret.
- **Data Sparsity:** Rare words or constructions may not appear in training data.
- **Grammar and syntax complexity:** Human language does not always follow strict rules.
- **Multilinguality:** Variations in languages add complexity.

2. Language Modeling:

Language modeling is a core component of NLP that helps in tasks like speech recognition, machine translation, and text generation.

a. Grammar-Based Language Models:

- These are based on formal grammars (like context-free grammars).
- Sentences are generated or recognized based on grammatical rules.
- **Advantage:** Linguistically interpretable.
- **Disadvantage:** Hard to scale to real-world, noisy data.

b. Statistical Language Models:

- Use probability distributions over sequences of words.
- **N-gram Models:** Estimate the probability of a word based on the previous $n-1$ words.
$$P(w_1, w_2, \dots, w_n) \approx \prod_{i=1}^n P(w_i | w_{i-1}, \dots, w_{i-n+1})$$
- **Smoothing techniques** like Laplace, Good-Turing, or Kneser-Ney are used to handle unseen sequences.
- **Advantage:** Easy to train with data.
- **Disadvantage:** Cannot capture long-range dependencies well.

3. Regular Expressions:

Regular expressions (regex) are sequences of characters defining search patterns.

- Used for text matching and extraction.
- Example: `[A-Z] [a-z] +` matches a capitalized word.
- Useful in preprocessing steps like tokenization, cleaning, etc.

4. Finite-State Automata (FSA):

FSA are machines with states and transitions used to recognize regular patterns in input.

- They process input one symbol at a time.
- Used in tasks like morphological parsing and lexical analysis.
- **Deterministic FSA (DFA)** and **Non-Deterministic FSA (NFA)** are common types.

5. English Morphology:

Morphology deals with the internal structure of words.

Types:

- **Inflectional morphology:** modifies a word's tense, number (e.g., “run” → “runs”).
- **Derivational morphology:** creates new words (e.g., “happy” → “unhappy”).

Morphological analysis helps in understanding word variations, lemmatization, and stemming.

6. Transducers for Lexicon and Rules:

Finite-State Transducers (FSTs) are automata that produce output as well as read input.

- Used for morphological analysis and generation.
- Example: Mapping “cats” to root “cat” + plural.
- Encodes both lexical entries (dictionary) and rules (like plural formation).

7. Tokenization:

Tokenization splits text into tokens—usually words, subwords, or sentences.

Challenges:

- Ambiguities with punctuation, contractions, and compound words.
- Language-specific rules (e.g., Chinese or Japanese lack whitespace).

Tools like **spaCy**, **NLTK**, and **BERT tokenizer** use advanced algorithms to handle such cases.

8. Detecting and Correcting Spelling Errors:

Spelling correction involves two steps:

- **Error detection:** Identify misspelled words.
- **Error correction:** Suggest the correct word.

Approaches include:

- **Dictionary lookup**
- **Phonetic similarity (Soundex)**

- **Statistical models based on language probabilities**

9. Minimum Edit Distance:

The minimum edit distance quantifies how dissimilar two strings are by counting the number of edit operations (insertions, deletions, substitutions).

Levenshtein distance is a common type.

Example:

- Transforming “**kitten**” to “**sitting**” requires 3 edits:
 - kitten → sitten (substitution)
 - sitten → sittin (substitution)
 - sittin → sitting (insertion)

Used in:

- Spelling correction
- Plagiarism detection
- DNA sequence analysis

1. Origins and Challenges of NLP

✦ Origins of NLP

- Emerged from the intersection of **linguistics**, **computer science**, and **artificial intelligence**.
- Initial systems were **rule-based** using hand-coded grammar rules.
- Shifted to **statistical methods** with the availability of large corpora (1990s onward).
- Recent trend: **Deep learning** and **transformer-based models** (e.g., BERT, GPT).

✦ Challenges in NLP

1. **Ambiguity:**

- Lexical: Words have multiple meanings (e.g., "bank").
- Syntactic: Multiple sentence structures possible.
- Semantic: Meaning depends on context.

2. **Context Sensitivity:**

- Same word can mean different things in different contexts.

3. **Resource Scarcity:**

- Lack of annotated data for many languages.

4. Morphological Complexity:

- Complex word forms (especially in agglutinative languages like Turkish or Finnish).

5. Idioms and Slang:

- Hard to model using literal language rules.

2. Language Modeling

✦ What is a Language Model?

- A **Language Model (LM)** assigns a probability to a sequence of words.
- Helps in predicting the next word in a sentence, used in applications like:
 - Speech recognition
 - Machine translation
 - Text generation

□ A. Grammar-Based Language Modeling

- Based on **formal grammars** (Context-Free Grammar, Phrase Structure Rules).
- Uses predefined **syntactic rules**.
- Example Rule:

$S \rightarrow NP VP$

$NP \rightarrow Det N$

$VP \rightarrow V NP$

✓ Pros:

- Human interpretable.
- Linguistically precise.

✗ Cons:

- Not robust to real-world data.
- Difficult to scale.

📊 B. Statistical Language Modeling

- Assigns probabilities to word sequences based on **training corpus**.

Types:

- **Unigram Model:** Each word is independent.
- **Bigram/Trigram Model:** Each word depends on previous 1 or 2 words.

$$P(w_1, w_2, \dots, w_n) \approx \prod_{i=1}^n P(w_i | w_{i-1}, \dots, w_{i-n+1})$$

📖 Smoothing Techniques (to handle unseen sequences):

- **Laplace Smoothing**
- **Good-Turing Discounting**
- **Kneser-Ney Smoothing**

3. Regular Expressions (RegEx)

★ Definition:

A **regular expression** is a sequence of characters defining a search pattern used for:

- Pattern matching
- Text extraction
- Tokenization

✂ Examples:

- `[A-Za-z ]+`: matches words.
- `\d{3}-\d{2}-\d{4}`: matches a SSN format (e.g., 123-45-6789).
- `^The`: matches sentences starting with “The”.

4. Finite-State Automata (FSA)

★ Definition:

A **Finite-State Automaton** is a machine with:

- A finite number of states
- Transitions based on input symbols
- Start and accepting states

🔄 Used in:

- Token recognition
- Morphological analysis
- Lexical analyzers in compilers

5. English Morphology

✦ Definition:

Morphology is the study of word structure, including roots, prefixes, and suffixes.

🔍 Types:

- **Inflectional Morphology:**
Changes word form to express tense, number, etc.
e.g., walk → walked
- **Derivational Morphology:**
Changes word class or meaning.
e.g., teach → teacher

📁 Common Morphological Operations:

- Affixation (prefix, suffix)
- Compounding
- Reduplication

6. Transducers for Lexicon and Rules

✦ Definition:

Finite-State Transducers (FSTs) are automata that map input to output, useful in:

- Morphological analysis
- Spelling correction
- Lexicon building

🔄 Example:

Input: `cats`

FST maps → Root: `cat` + Suffix: `+PL`

Output: Lemma `cat`, feature `plural`

7. Tokenization

✦ Definition:

Tokenization is the process of splitting text into **tokens** (words, punctuation, subwords).

☐ Challenges:

- Punctuation: "Let's go!" → "Let", "'s", "go"

- Compound words: "e-mail", "New York"
- Language-specific issues: no spaces in Chinese or Japanese

🔧 Tools:

- NLTK
- spaCy
- BERT WordPiece tokenizer

8. Detecting and Correcting Spelling Errors

🔍 Spelling Error Detection

- Use dictionaries or trained language models to flag unknown words.

🔧 Correction Techniques:

- **Edit Distance-based Correction**
- **Phonetic Matching (Soundex, Metaphone)**
- **Confusion Set Modeling**
 - E.g., their/there/they're

9. Minimum Edit Distance

✦ Definition:

Measures the **minimum number of operations** needed to convert one string to another.

🔢 Operations:

- Insertion
- Deletion
- Substitution

□ Levenshtein Distance Algorithm

- Dynamic programming approach
- Cost of:
 - Insertion = 1
 - Deletion = 1
 - Substitution = 1 (or 0 if characters match)

■ Example:

Transform “kitten” → “sitting”

- kitten → sitten (substitute 'k' → 's')
- sitten → sittin (substitute 'e' → 'i')
- sittin → sitting (insert 'g')
→ Edit distance = **3**