Computational Linguistics & NLP.

---

1. What is Computational Linguistics?

- **Interdisciplinary field** combining **linguistics + computer science**
- Focuses on **computational processing of natural (human) language**
- Goal: Enable machines to **understand, generate, and interpret meaningful language**

---

2. Why Do We Need Computational Linguistics?

Key Motivations

1. **Language Complexity**
    - Human language is complex, ambiguous, and context-dependent
    - Machines need structured computational models to understand it
2. **Communication Gap**
    - Reduces the gap between **human language** and **machine understanding**
3. **Automation**
    - Essential for tasks like:
        - Translation
        - Speech recognition
        - Text generation
        - Summarization
4. **Information Processing**
    - Massive amounts of text data need automated analysis
    - Language understanding can reveal new insights
5. **Globalization**
    - Enables communication across **multiple languages worldwide**

---

3. How Does Computational Linguistics Work?

Main Approaches

1. Rule-Based Approach

- Uses **linguistic rules**, grammar, and dictionaries
- Focuses on sentence structure and word meaning
- Used in **early language systems**

2. Statistical & Machine Learning Approach

- Uses **large datasets** and probabilistic models
- Learns patterns from examples
- Improves automatically with more data
- Modern systems include models like **GPT** and **BERT**

---

4. Types of Computational Linguistics

1. **Theoretical**
    - Builds computational models to explain how language works
2. **Applied**
    - Develops practical applications (speech recognition, chatbots)
3. **Cognitive**
    - Models language processing based on **human cognition**

---

5. Key Terminology (High Priority)

- **Phonetics & Phonology** – study of speech sounds
- **Morphology** – word formation and structure
- **Semantics** – meaning of words and sentences
- **Pragmatics** – meaning based on context
- **Corpus** – large collection of text data
- **NLP** – enabling machines to process human language
- **Tokenization** – splitting text into words/subwords
- **Parsing** – grammatical analysis of sentences
- **NER (Named Entity Recognition)** – identifying names, dates, places

- **POS Tagging** – assigning grammatical labels
- **Word Embeddings** – vector representation of words

---

6. Applications of Computational Linguistics

- **Machine Translation** – e.g., Google Translate
- **Speech Recognition** – e.g., Siri
- **Text-to-Speech / Speech-to-Text**
- **Chatbots & Virtual Assistants** – e.g., Alexa
- **Sentiment Analysis** – customer reviews, opinions
- **Grammar Checking** – e.g., Grammarly

---

7. Challenges in Computational Linguistics

- **Ambiguity** (same word, multiple meanings)
- **Context understanding** (sarcasm, irony)
- **Low-resource languages** (lack of data)
- **Idioms, humor, and cultural nuances**
- **Bias** due to skewed training data

---

8. One-Line Summary (For Revision)

Computational Linguistics enables machines to understand and generate human language using linguistic rules and machine learning techniques, powering applications like translation, speech recognition, and chatbots.

🔑 **Key Terms in Computational Linguistics (Made Easy)**

🗣️ **1. Phonetics & Phonology**

👉 Study of how words **sound** and how sounds are **organized**.

💡 **Example:** Alexa or Siri listening carefully to your **voice sounds** to understand what you said.

🧩 **2. Morphology**

👉 Study of how **words are made** from smaller parts.

💡 **Example:** A spell checker knows that **"running"** comes from **"run"**.

💬 **3. Semantics**

👉 Study of the **meaning** of words and sentences.

💡 **Example:** Google knows the word **"apple"** can mean a **fruit** or a **company**, depending on the sentence.

🧠 **4. Pragmatics**

👉 Understanding **language in context** — not just words, but the **situation**.

💡 **Example:** A chatbot knows that "Can you open the door?" is a **request**, not a **question**.

📚 **5. Corpus**

👉 A **large collection of texts** used to study or train computers on language.

💡 **Example:** Wikipedia articles are used as a **text collection** to train language models like ChatGPT.

🤖 **6. NLP (Natural Language Processing)**

👉 Technology that helps computers **understand human language**.

💡 **Example:** ChatGPT, Google Translate, or Siri use NLP to talk and understand.

✂️ 7. Tokenization

👉 Breaking big text into **smaller parts** like words or sentences.

💡 **Example:** “I love pizza” → [“I”, “love”, “pizza”]

📜 8. Parsing

👉 Checking **grammar structure** of a sentence.

💡 **Example:** Grammarly checks if a sentence like “She go to school” is **wrong** and suggests “She goes to school.”

🏷️ 9. NER (Named Entity Recognition)

👉 Finding **names, dates, places**, etc. in text.

💡 **Example:** In “Elon Musk founded SpaceX in 2002,” NER finds:

- Person: *Elon Musk*
- Organization: *SpaceX*
- Year: *2002*

🧩 10. POS Tagging (Part of Speech Tagging)

👉 Telling what part of speech each word is — noun, verb, adjective, etc.

💡 **Example:** “Dogs bark loudly” →

Dogs (noun), bark (verb), loudly (adverb)

🔢 11. Word Embeddings

👉 Turning words into **numbers** so that computers can understand them.

💡 **Example:** “King” and “Queen” have **similar numbers** because they have related meanings.

History of NLP

📖 **The Story of Natural Language Processing (NLP)**

(Easy to remember, like a timeline of discovery)

🕰️ 1950 – The Birth of the Idea: The Turing Test

👨‍💼 *Character:* **Alan Turing** (The Father of AI)

💡 *What happened:*

Turing asked a simple but powerful question —

"Can a machine think like a human?"

He created the **Turing Test** — a way to check if a machine's conversation is **indistinguishable** from a human's.

💬 *Simple example:*

If you chat with a computer and can't tell it's a machine, it passes the test.

📊 *Data Science Connection:*

This was the **first evaluation metric** for "machine intelligence" — similar to how we measure accuracy or F1 score today!

💻 1954 – The Georgetown–IBM Experiment

🚀 *Event:* The first **Machine Translation** project.

💡 *What happened:*

IBM's computer translated **60 Russian sentences** into **English** automatically.

💬 *Example:*

Russian → English: "Это тест." → "This is a test."

📊 *Data Science Connection:*

This was the **first NLP dataset**! It showed how **language data** could be used to **train models** — the foundation of today's **translation systems** like Google Translate.

💬 1964–1966 – ELIZA: The First Chatbot

🤖 *Character:* ELIZA the “Psychotherapist Bot.”

💡 *What happened:*

ELIZA used **pattern matching** to hold simple conversations, pretending to be a therapist.

💬 *Example:*

User: *“I feel sad.”*

ELIZA: *“Why do you feel sad?”*

📊 *Data Science Connection:*

ELIZA is the **ancestor of ChatGPT!**

She used **text pattern recognition** — the earliest form of **rule-based NLP**.

🌙 1971 – LUNAR System

🚀 *Event:* NASA built a system to answer questions about **moon rocks** using English.

💡 *What happened:*

You could ask:

“What is the composition of basalt rock?”

and it would give a factual answer from its database.

📊 *Data Science Connection:*

This was an **early Question-Answering (QA) System** — what we now do with **LLMs** and **knowledge graphs**.

🧠 1976 – Conceptual Dependency Theory

💡 *What happened:*

Scientists tried to make computers understand the **meaning behind sentences**, not just words.

They used **conceptual primitives** — like basic building blocks of meaning.

💬 *Example:*

“The cat ate the fish” → understood as *an action (eat)* done by *an agent (cat)* on *an object (fish)*.

📊 *Data Science Connection:*

This was an early version of **semantic modeling** — the concept behind **embedding spaces** and **transformers** today.

🏷️ **1980s – POS Tagging (Part of Speech Tagging)**

💡 *What happened:*

Computers learned to label words as **nouns**, **verbs**, **adjectives**, etc.

💬 *Example:*

"She runs fast." → [She (pronoun), runs (verb), fast (adverb)]

📊 *Data Science Connection:*

POS tagging became a **core preprocessing step** — part of the **text cleaning pipeline** every data scientist uses before modeling.

🔢 **1986 – Statistical Machine Translation**

💡 *What happened:*

NLP shifted from **rule-based systems** to **data-driven models**.

Instead of manually coding rules, systems started **learning patterns** from lots of text data.

💬 *Example:*

If "bonjour" often appears with "hello", the model learns the translation automatically.

📊 *Data Science Connection:*

This was the **birth of NLP using probabilities and statistics** — paving the way for **machine learning-based NLP**.

📜 **1990 – WordNet**

💡 *What happened:*

A huge **English word database** was built, showing how words relate in meaning.

💬 *Example:*

happy → related words: *joyful*, *cheerful*, *content*

📊 *Data Science Connection:*

WordNet became the **training dictionary** for NLP — used in **semantic analysis**, **sentiment detection**, and **word similarity** tasks.

🔍 1997 – Hidden Markov Models (HMMs)

💡 *What happened:*

HMMs were used to understand **sequences**, like words in a sentence or sounds in speech.

💬 *Example:*

Predicting the next word in "I am going ___" → *home* or *out*

📊 *Data Science Connection:*

This was the start of **sequence modeling** — the foundation of **speech recognition**, **POS tagging**, and **time-series prediction**.

🧠 2001 – IBM Watson

💡 *What happened:*

IBM built **Watson**, an AI system that could **understand questions** and **find answers**.

In 2011, Watson **beat humans** on the quiz show *Jeopardy!*

💬 *Example:*

Question: "This planet is known as the Red Planet."

Watson: "What is Mars?"

📊 *Data Science Connection:*

Watson used **machine learning + NLP + search algorithms** — like a data scientist combining **text analysis and predictive models**.

🌐 2006 – Google Translate

💡 *What happened:*

Google launched **real-time translation** between many languages.

It started with **statistical models**, then later used **neural networks**.

💬 *Example:*

Type: “Hello” → “Hola” (Spanish)

📊 *Data Science Connection:*

Showed the **power of large datasets** and **model training** — a practical example of **NLP in production**.

💡 **2013 – Word2Vec**

💡 *What happened:*

Google created **Word2Vec**, which taught computers to understand **word meanings through numbers (vectors)**.

💬 *Example:*

king - man + woman = queen 👑

📊 *Data Science Connection:*

Word2Vec introduced **word embeddings** — the idea that **semantic relationships** can be captured mathematically.

This is used in **recommendation systems** and **text classification**.

🔥 **2018 – BERT (by Google)**

💡 *What happened:*

BERT allowed machines to understand **context from both directions** — before and after a word.

It made huge improvements in NLP tasks like **question answering** and **sentiment analysis**.

💬 *Example:*

“The bank by the river” vs. “The bank gave me a loan.”

→ BERT understands the difference from context.

📊 *Data Science Connection:*

BERT started the **Transformers era** — used in **text embeddings**, **QA systems**, and **transfer learning**.

🤯 **2019–Present – Large Language Models (LLMs)**

💡 *What happened:*

Models like **GPT**, **Claude**, **Gemini**, etc. can now **generate, summarize, and translate** human-like text.

They're trained on **huge datasets** with **trillions of parameters**.

💬 *Example:*

ChatGPT writing essays, answering questions, generating code, or summarizing reports.

📊 *Data Science Connection:*

LLMs are **data-hungry deep learning models**.

They show the **power of data**, **scaling**, and **transformer architectures** — the cutting edge of NLP in data science.

🧻 **Quick Summary Table**

Year	Innovation	Easy Example	Data Science Connection
1950	Turing Test	Can a machine think?	Evaluation metric
1954	IBM Experiment	Russian → English	First translation dataset
1966	ELIZA	Chatbot therapist	Pattern recognition
1971	LUNAR	Moon rock Q&A	Early QA system
1976	Concept Theory	Cat eats fish → meaning	Semantic modeling
1980s	POS Tagging	Labeling words	Text preprocessing
1986	Statistical MT	Data-driven translation	Probabilistic modeling
1990	WordNet	Word meanings	Semantic networks
1997	HMMs	Predict next word	Sequence modeling
2001	IBM Watson	Quiz AI	NLP + ML integration
2006	Google Translate	Multilingual text	Big data in NLP

2013	Word2Vec	King–man+woman=Queen	Word embeddings
2018	BERT	Context understanding	Transformers
2019+	GPT, LLMs	Human-like text	Deep learning NLP

◇ COMPLETE NLP PREPROCESSING PIPELINE (WITH LIBRARIES)

---

1. Text Acquisition

- Collecting raw text from PDFs, web pages, databases, social media, speech outputs
- Data may be unstructured and noisy
  Library: requests, BeautifulSoup, scrapy, pandas

---

2. Text Cleaning / Noise Removal

- Remove HTML tags, URLs, emails
- Remove special characters, symbols, extra whitespaces
- Handle emojis (remove or convert)
- Remove unwanted numbers (task-dependent)
  Library: re, BeautifulSoup, emoji

---

3. Case Normalization

- Convert text to lowercase for consistency
- Avoids duplicate vocabulary
  Library: Python built-in (str.lower())

---

4. Sentence Segmentation

- Splits text into meaningful sentences
- Handles abbreviations and punctuation properly
  Library: nltk, spacy

---

5. Tokenization

- Splits text into tokens (words/subwords)
- Types: word, sentence, subword
  Library: nltk, spacy, transformers

---

6. Noise Filtering (Punctuation & Digits)

- Remove punctuation and numeric tokens
- Task-specific decision
  Library: string, re, nltk

---

7. Stop Word Handling

- Removes common words with low semantic value
- Sometimes retained for sentiment analysis
  Library: nltk, spacy, sklearn

---

8. Text Normalization

- Expand contractions (can't → cannot)
- Standardize spelling and abbreviations
  Library: contractions, textblob

---

9. Handling Repeated Characters

- Normalizes elongated words
- Example: soooo → so

  Library: re

---

10. Handling Negations ⭐

- Preserves sentiment meaning
- Example: not good → not_good

  Library: nltk, spacy

---

11. Emoji & Emoticon Processing

- Converts emojis into text meaning
- Important for social media NLP

  Library: emoji, emot

---

12. Slang & Informal Language Handling

- Converts slang to formal language
- Example: idk → I do not know

  Library: Custom dictionary, slangify, re

---

13. Spelling Correction

- Corrects misspelled and real-word errors

  Library: textblob, pyspellchecker, symspellpy

---

14. Stemming

- Converts words to root form
- Fast but less accurate

  Library: nltk

---

15. Lemmatization

- Converts words to dictionary base form
- More accurate than stemming

  Library: nltk, spacy

---

16. Part-of-Speech (POS) Tagging

- Assigns grammatical roles to words

  Library: nltk, spacy

---

17. Named Entity Recognition (NER)

- Identifies names, locations, organizations, dates

  Library: spacy, nltk, flair

---

18. Chunking (Shallow Parsing)

- Groups tokens into phrases (NP, VP)

  Library: nltk

---

19. Dependency Parsing

- Identifies grammatical relationships between words
  Library: spacy

---

20. Syntax Tree / Constituency Parsing

- Generates full parse trees
  Library: nltk, benepar

---

21. Coreference Resolution ⭐

- Resolves pronouns to actual entities
- Example: He → John
  Library: spacy-coref, neuralcoref, allenNLP

---

22. Text Canonicalization

- Converts multiple forms into a single standard form
- Example: USA, U.S.A → United States
  Library: Custom mapping, fuzzywuzzy

---

23. Handling Multilingual Text

- Language detection
- Language-specific preprocessing pipelines
  Library: langdetect, fastText

---

24. Handling Domain-Specific Terms

- Normalizes medical, legal, financial terms
  Library: Custom dictionary, scispacy

---

25. Handling Speech-to-Text Noise

- Removes fillers (uh, um)
- Normalizes pauses
  Library: re, pydub

---

26. Vocabulary Reduction

- Remove rare words
- Replace rare words with <UNK> token
  Library: collections, sklearn

---

27. Feature Extraction / Vectorization

- Converts text into numerical form

Traditional

- Bag of Words
- TF-IDF
  Library: sklearn

Modern

- Word embeddings (Word2Vec, GloVe)
- Contextual embeddings (BERT)
  Library: gensim, transformers

---

28. Data Augmentation (Text Level)

- Synonym replacement
- Back translation
- Random deletion/insertion

  Library: nlpaug, textattack

---

29. Handling Imbalanced Text Data

- Oversampling / undersampling

  Library: imblearn

---

30. Bias Detection & Ethical Preprocessing

- Detects and mitigates bias in text data

  Library: aif360, fairlearn

---

31. Final Model-Ready Text

- Cleaned
- Normalized
- Tokenized
- Vectorized

  Library: Depends on model (sklearn, pytorch, tensorflow)

🔹 **FINAL MASTER FLOW (EXAM PERFECT)**

Raw Text
→ Cleaning
→ Normalization
→ Sentence Segmentation
→ Tokenization
→ Stopword & Negation Handling
→ Lemmatization / Stemming
→ POS / NER / Coreference
→ Parsing
→ Feature Extraction
→ Data Augmentation & Bias Check
→ Model Input

🔹 **FINAL REMAINING NLP PREPROCESSING STEPS (ULTRA-ADVANCED)**

---

32. Sentence Boundary Disambiguation

- Handles complex sentence endings (e.g., “Dr. Smith lives in the U.S.”)
- More accurate than basic sentence segmentation

  Library: spacy, nltk-punkt

---

33. Subword Segmentation (BPE / WordPiece)

- Breaks unknown or rare words into subwords
- Essential for transformer models
  Library: sentencepiece, tokenizers, transformers

---

34. Handling Out-of-Vocabulary (OOV) Words

- Replaces unseen words with <UNK>
- Or uses subword embeddings
  Library: gensim, transformers

---

35. Text Deduplication

- Removes duplicate or near-duplicate sentences/documents
- Prevents model bias and data leakage
  Library: datasketch, simhash

---

36. Text Length Normalization

- Padding and truncation of sequences
- Required for deep learning models
  Library: keras.preprocessing, transformers

---

37. Sentence / Document Segmentation

- Splits long documents into logical sections
  Library: spacy, nltk

---

38. Readability & Quality Filtering

- Removes low-quality or unreadable text
  Library: textstat, langdetect

---

39. Named Entity Normalization (Entity Linking)

- Maps entities to canonical knowledge base entries
  Library: spacy, wikidata, blink

---

40. Temporal Expression Normalization

- Converts time expressions into standard format
- Example: “yesterday” → 2026-01-20
  Library: HeidelTime, SUTime

---

41. Toxicity & Content Filtering

- Removes abusive or unsafe language
  Library: detoxify, perspective

---

42. Privacy & PII Masking

- Masks personal information
- GDPR-compliant preprocessing
  Library: presidio, spacy

---

43. Multimodal Alignment (Text + Speech / Image)

- Aligns text with audio or image features
  Library: whisper, CLIP

---

44. Dataset Versioning & Reproducibility

- Tracks preprocessing changes over time
  Library: DVC, MLflow

🧠 **FINAL ABSOLUTE MASTER PIPELINE (100%)**

Raw Data

→ Cleaning & Noise Removal

→ Normalization & Canonicalization

→ Sentence Boundary Detection

→ Tokenization / Subword Segmentation

→ Stopwords, Negation & Emoji Handling

→ Lemmatization / Stemming

→ POS / NER / Entity Linking

→ Coreference & Discourse Processing

→ Parsing (Dependency & Constituency)

→ Vocabulary & OOV Handling

→ Vectorization & Embeddings

→ Augmentation & Bias Mitigation

→ Deduplication & Quality Filtering

→ Privacy, Safety & Ethics

→ Model-Ready Input

(Correct Order + DOs & DON'Ts + Examples)

---

1. Raw Text Collection
What happens:
Text is collected from sources like websites, PDFs, social media, speech-to-text systems.
DO

- Store raw text safely and unchanged
- Keep a copy of original data

DON'T

- Clean or modify text during collection

Why:
Raw data is needed for debugging and reproducibility.

---

2. Remove HTML Tags, URLs, Emails
What happens:
Unnecessary markup and links are removed.
DO

- Remove <html>, , URLs, emails early
- Use regex or HTML parsers

DON'T

- Remove punctuation or words yet

Example
" Hello visit https://site.com"
→ "Hello visit"
Why:
These elements add noise and confuse tokenizers.

---

3. Encoding & Whitespace Normalization
What happens:
Fix Unicode issues and spacing problems.
DO

- Normalize accents
- Remove extra spaces and line breaks

DON'T

- Change casing or meaning

Why:
Prevents tokenizer and parser errors.

---

4. Expand Contractions & Slang
What happens:
Short forms are expanded into full forms.
DO

- Expand early
- Use dictionaries

DON'T

- Expand after lemmatization

Example
"can't wait idk"
→ "cannot wait I do not know"
Why:
Preserves grammatical and semantic meaning.

---

5. Emoji & Emoticon Handling
What happens:
Emojis are converted into text meaning.
DO

- Convert emojis to words when sentiment matters

DON'T

- Remove emojis blindly

Example
"I am happy 😊"
→ "I am happy happy"
Why:
Emojis carry strong emotional signals.

---

6. Spelling Correction
What happens:
Corrects obvious spelling errors.
DO

- Correct common mistakes
- Apply before stemming/lemmatization

DON'T

- Correct named entities

Example
"goood product"
→ "good product"
Why:
Misspellings create fake vocabulary.

---

7. Sentence Segmentation
What happens:
Text is split into sentences.
DO

- Segment sentences before word tokenization

DON'T

- Tokenize words first

Example
"Dr. Smith left. He slept."
Why:
Sentence boundaries affect meaning and parsing.

---

8. Case Normalization (Task-Aware)

What happens:
Text is converted to lowercase selectively.
DO

- Lowercase common words
- Preserve acronyms and proper nouns

DON'T

- Lowercase everything blindly

Example
"NASA launched Rocket"
→ "NASA launched rocket"
Why:
Capitalization helps NER and entity detection.

---

9. Tokenization (Word / Subword)
What happens:
Text is split into tokens.
DO

- Use standard tokenizers
- Use subword tokenization for transformers

DON'T

- Split text manually using spaces

Example
"I love NLP"
→ ["I", "love", "NLP"]
Why:
Correct tokenization is foundation for all NLP tasks.

---

10. Punctuation Handling
What happens:
Punctuation is removed or normalized.
DO

- Remove after sentence splitting

DON'T

- Remove punctuation too early

Why:
Punctuation helps sentence detection.

---

11. Stopword Handling
What happens:
Common words are removed.
DO

- Remove stopwords only if task allows

DON'T

- Remove negation words

Example
"this is good"
→ "good"

Why:
Stopwords sometimes carry meaning.

---

12. Negation Handling
What happens:
Negations are preserved explicitly.
DO

- Join negation with word

DON'T

- Ignore negations

Example
"not happy"
→ "not_happy"
Why:
Negation flips sentiment meaning.

---

13. Number Handling
What happens:
Numbers are processed carefully.
DO

- Keep dates, prices, quantities

DON'T

- Remove all numbers blindly

Why:
Numbers often carry crucial information.

---

14. POS Tagging
What happens:
Each token is assigned a grammatical role.
DO

- Apply after tokenization

DON'T

- POS tag stemmed text

Why:
POS tags guide lemmatization and parsing.

---

15. Lemmatization
What happens:
Words are reduced to dictionary base form.
DO

- Use POS-aware lemmatization

DON'T

- Lemmatize before POS tagging

Example
"running" → "run"
Why:
Keeps words meaningful and consistent.

---

16. Named Entity Recognition (NER)
What happens:
Identifies real-world entities.
DO

- Preserve original casing

DON'T

- Lemmatize entity names

Example
"New York" ≠ "New Yorks"
Why:
Entities must remain unchanged.

---

17. Entity Normalization
What happens:
Maps entities to standard names.
DO

- Normalize after NER

DON'T

- Normalize before detection

Example
"U.S.A" → "United States"

---

18. Chunking
What happens:
Groups words into phrases.
DO

- Apply after POS tagging

DON'T

- Chunk raw text

---

19. Dependency & Syntax Parsing
What happens:
Identifies grammatical relationships.
DO

- Parse clean, structured sentences

DON'T

- Parse noisy text

---

20. Coreference Resolution
What happens:
Links pronouns to actual entities.
DO

- Apply after NER and parsing

DON'T

- Apply on broken sentences

---

21. Vocabulary Control

What happens:

Removes rare words or replaces with <UNK>.

DO

- Do after lemmatization

DON'T

- Remove important rare words blindly

---

22. Feature Extraction / Vectorization

What happens:

Text is converted into numbers.

DO

- Vectorize only after all preprocessing

DON'T

- Modify text after this step

---

23. Padding & Truncation

What happens:

Makes sequences equal length (DL only).

DO

- Pad after tokenization

DON'T

- Pad raw text

---

24. Data Augmentation

What happens:

Creates new text samples.

DO

- Augment clean text only

DON'T

- Augment noisy text

---

25. Bias, Toxicity & PII Filtering

What happens:

Ensures ethical and legal compliance.

DO

- Apply before training

DON'T

- Remove protected attributes blindly

---

26. Final Model-Ready Text

What happens:

Clean, structured, numerical input for models.

DO

- Lock preprocessing pipeline

DON'T

- Change text after this stag

⏳ **FINAL GOLDEN RULE**
Never destroy meaning before structure, and never change structure after numbers.

Regex Symbols Cheat Sheet

Symbol / Syntax	Meaning / Description	Example Match
.	Any single character except newline	a.c → matches "abc", "axc"
^	Start of a line or string	^Hello → matches "Hello world" but not "He said Hello"
$	End of a line or string	world$ → matches "Hello world"
*****	0 or more repetitions of the previous pattern	go* → "g", "go", "goo"
+	1 or more repetitions of the previous pattern	go+ → "go", "goo"
?	0 or 1 repetition (makes it optional)	colou?r → "color" or "colour"
{n}	Exactly *n* repetitions	a{3} → "aaa"
{n,}	At least *n* repetitions	a{2,} → "aa", "aaa", "aaaa"
{n,m}	Between *n* and *m* repetitions	a{2,4} → "aa", "aaa", "aaaa"
[]	Character class (match any one inside)	[abc] → "a", "b", "c"
[^]	Negated class (match anything *except* inside)	[^0-9] → any non-digit
-	Range inside character class	[A-Z] → uppercase letters
()	Capturing group (captures matched text)	(abc)+ → captures "abc", "abcabc"
(?:)	Non-capturing group	(?:abc)+ → matches same as above, but doesn't capture
|	OR (alternation)	cat|dog → "cat" or "dog"
****	Escape special character	\. → matches "." literally
\d	Any digit ([0-9])	\d\d → "42"
\D	Any non-digit ([^0-9])	\D+ → "abc"
\w	Any "word" character (letters, digits, underscore)	\w+ → "Hello_123"
\W	Any non-word character	\W → "!", "@", " "
\s	Any whitespace (space, tab, newline)	\s+ → " "
\S	Any non-whitespace	\S+ → "word"
(?=...)	Positive lookahead (must be followed by …)	\d(?=px) → digit before "px"
(?!...)	Negative lookahead (must *not* be followed by …)	\d(?!px) → digit not before "px"
(?<=...)	Positive lookbehind (must be preceded by …)	(?<=#)\w+ → word after #
(?<!...)	Negative lookbehind (must *not* be preceded by …)	(?<!@)\w+ → word not after @
`	`	Alternation (either this or that)
\A	Start of string (like ^, but not multiline)	\AHello
\Z	End of string (like $, but not multiline)	world\Z
\b	Word boundary	\bcat\b → "cat" as a whole word

\B	Non-word boundary	\Bcat\B → "concatenate"

💡 **Quick Examples**

Pattern	Matches	Doesn't Match
\d{3}-\d{2}-\d{4}	123-45-6789	123456789
https?://\S+	http://abc.com, https://xyz.org	ftp://abc.com
[A-Za-z_]\w*	Valid Python variable names	123name
#(\w+)	#AI, #python	#

🔹 TEXT NORMALIZATION — INCLUDES (⭐ = USED MORE OFTEN)

---

1. Case Normalization ⭐

- Convert text to lowercase (task-aware)
- Preserve acronyms and entities (NASA ≠ nasa)

Example:
"NLP Is Fun" → "nlp is fun"

---

2. Unicode & Encoding Normalization

- Standardize Unicode characters and accents

Example:
"café" → "cafe"

---

3. Whitespace Normalization ⭐

- Remove extra spaces, tabs, and line breaks

Example:
"hello world" → "hello world"

---

4. Contraction Expansion ⭐

- Expand shortened forms

Example:
"can't" → "cannot"

---

5. Slang & Informal Language Normalization ⭐

- Convert slang into standard language

Example:
"idk" → "I do not know"

---

6. Spelling Normalization ⭐

- Correct common misspellings and typos

Example:
"goood" → "good"

---

7. Repeated Character Normalization ⭐

- Reduce elongated words

Example:
"soooo happy" → "so happy"

---

8. Emoji & Emoticon Normalization ⭐

- Convert emojis into textual meaning

Example:
😊 → happy

---

9. Punctuation Normalization ⭐

- Normalize or remove punctuation

Example:
"hello!!!" → "hello!"

---

10. Number Normalization ⭐

- Standardize numeric expressions

Example:
"ten" → "10"

---

11. Date & Time Normalization

- Convert temporal expressions into standard format

Example:
"yesterday" → 2026-01-20

---

12. Abbreviation Normalization ⭐

- Expand abbreviations

Example:
"U.S.A" → "United States"

---

13. Unit Normalization

- Standardize measurement units

Example:
"5 km" → "5000 meters"

---

14. Entity Canonicalization ⭐

- Map different forms of an entity to one standard form

Example:
"USA", "U.S." → "United States"

---

15. Negation Normalization ⭐

- Preserve negation meaning

Example:
"not good" → "not_good"

---

16. Language Normalization (Multilingual)

- Normalize mixed-language text

Example:
"Hinglish" → English

---

17. Text Case Folding (Advanced)

- Normalize case without losing semantic meaning

Example:
"Apple" (company) ≠ "apple" (fruit)

Text Representation Technique

🔹 **TEXT REPRESENTATION TECHNIQUES (NLP)**

Text representation is the process of converting text into **numerical form** so that machines can understand and process it.

---

1. One-Hot Encoding

- Represents each word as a binary vector
- Vector length = vocabulary size

Example
cat → [0,0,1,0,0]
Pros

- Simple
- Easy to implement

Cons

- Very high dimensional
- No semantic meaning

Use case

- Small vocabulary experiments

---

2. Bag of Words (BoW) ⭐

- Represents text by **word frequency**
- Ignores word order

Example
Sentence: "I love NLP NLP"
Vector: {I:1, love:1, NLP:2}
Pros

- Simple
- Effective for basic ML models

Cons

- No semantics
- No word order

Library

- sklearn

---

3. N-grams ⭐

- Extends BoW by considering word sequences

Example
Bigrams of "I love NLP"
→ ["I love", "love NLP"]
Pros

- Captures local context

Cons

- Larger feature space

Library

- sklearn

---

4. TF-IDF (Term Frequency – Inverse Document Frequency) ⭐

- Weighs words by importance
- Reduces impact of frequent but unimportant words

Formula (idea)

TF × IDF

Pros

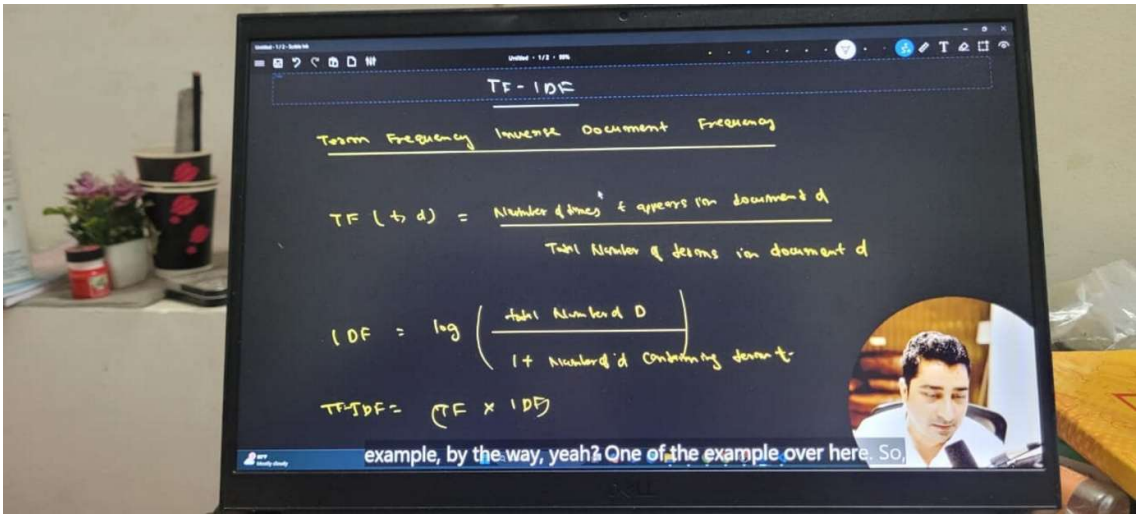
- Better than BoW
- Reduces stopword dominance

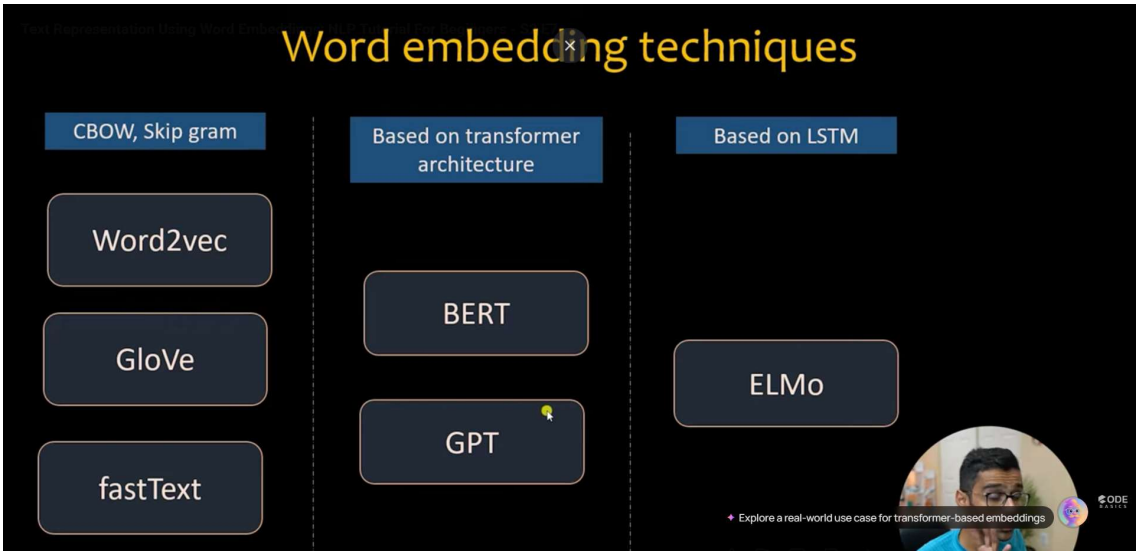
Cons

- Still no deep semantics

Library

- Sklearn
-





🔹 1. Recap: Why TF–IDF?

Till now, we studied:

- One Hot Encoding
- Bag of Words
- Bag of N-Grams

All three treat every word equally —
each word gets the same importance or *weight*.
But in reality, not all words are equally important.

👉 For example, in a sentence like
"People watch campus selection"
The words *people*, *watch*, *campus* — all get weight = 1 in BoW.
But intuitively, maybe *campus* is more specific and meaningful than *people*.
So, we need a method that gives more weight to meaningful or rare words
and less weight to common or generic words like "is", "the", "a", "people".
That's where TF–IDF comes in.

🔹 2. What is TF–IDF?

TF–IDF stands for Term Frequency–Inverse Document Frequency
It is used to measure how important a word is in a specific document,
relative to all other documents in the corpus.

🔸 Intuition:

A word is important if:

- It appears many times in a particular document (high TF),
- But appears rarely across other documents (high IDF).

Such words describe the topic of that document best.

🔹 3. Working (Step-by-Step)

Let's break it down 👇

🍀 Step 1: Term Frequency (TF)

It measures how frequently a word occurs in a document.

✅ TF ranges between 0 and 1.
It can also be seen as the probability of that word in the document.
Example:
Document: "People watch campus"

- Total words = 3
- Frequency of "campus" = 1
  → TF(campus, doc) = 1 / 3 = 0.33

🍀 Step 2: Inverse Document Frequency (IDF)

It measures how rare a word is across all documents in the corpus.

✅ If a word appears in every document,
its IDF → 0 (not important).
✅ If it appears in few documents,
its IDF → high (important).

🍀 Step 3: Combine TF and IDF

Finally, multiply both:

TF-IDF(t,d)=TF(t,d)×IDF(t)TF\text{-}IDF(t, d) = TF(t, d) \times IDF(t)TF-IDF(t,d)=TF(t,d)×IDF(t)
This gives the final weight of a word in a document.

🧮 Example

Word	TF (Doc1)	DF	IDF = log(N/DF)	TF×IDF
people	1/3	2	log(4/2)=0.30	0.10
watch	1/3	1	log(4/1)=0.60	0.20
campus	1/3	3	log(4/3)=0.12	0.04

So, the word “watch” gets highest importance → most meaningful.

🔹 4. Why “log” in IDF?
Because without log, very rare words would get *extremely high* values.
Log helps normalize the scale — so no word dominates too strongly.
📘 In interview:
Log prevents extremely large IDF values for rare words and maintains balance between TF and IDF.

🔹 5. TF–IDF Implementation in Python

```
from sklearn.feature_extraction.text import TfidfVectorizer
docs = [
  "People watch campus selection",
  "Campus placement drive",
  "Write comments on campus",
  "Watch people comment online"
]
# Create TF-IDF object

vectorizer = TfidfVectorizer()

# Fit and transform

X = vectorizer.fit_transform(docs)
# Show vocabulary

print(vectorizer.get_feature_names_out())

# Show TF-IDF matrix

print(X.toarray())


# Show IDF values
print(vectorizer.idf_)
```

✅ By default, sklearn uses log((N + 1) / (df + 1)) + 1
to prevent division by zero and handle smoothing.

🔹 6. Interpretation
- Each row → a document
- Each column → a word
- Each value → importance of that word in that document

Words common in all documents get small weights.
Rare and specific words get large weights.

🔹 7. Advantages of TF–IDF

Advantage	Explanation
🎯 Weights important words	Words unique to a document get higher score
🧠 Captures document-specific importance	Focuses on discriminative terms
⚙️ Works well for search engines	Helps ranking documents by relevance
🪶 Lightweight and interpretable	Easy to compute and explain

🔹 8. Disadvantages of TF–IDF

Disadvantage	Explanation
⚡ Sparsity	Still produces high-dimensional sparse vectors
🧩 No semantic understanding	Treats “good” and “great” as totally unrelated
🚫 OOV problem	Words not in vocabulary are ignored
🗄️ Large vocab = large feature space	High memory and computation
💬 Static meaning	Does not adapt contextually to sentences

🔹 9. Applications

- 🔍 Search Engines → Rank documents based on TF–IDF similarity
- 💬 Text Classification → Spam detection, sentiment analysis
- 📄 Information Retrieval Systems
- 🧩 Keyword Extraction

🔹 10. Interview Questions & Answers

Q1. What is TF–IDF?
A statistical measure that evaluates how important a word is to a document in a collection of documents.

Q2. What does TF represent?
It represents the frequency of a word in a document.

Q3. What does IDF represent?
It represents how rare a word is across the entire corpus.

Q4. Why do we take the log of IDF?
To reduce the effect of extremely rare words and keep scores balanced.

Q5. What is the range of TF and IDF values?

- TF → 0 to 1
- IDF → ≥ 0 (higher for rare terms)

Q6. Where is TF–IDF used in real-world systems?
Used in search engines, document similarity, keyword extraction, and information retrieval systems.

🔹 11. Advantages over BoW and N-Grams

Feature	BoW	N-Grams	TF–IDF
Captures frequency	✅	✅	✅
Captures importance	❌	❌	✅
Word order	❌	✅ (partially)	❌
Dimensionality	High	Very High	High
Handles OOV	Partial	Partial	Partial

Semantic understanding	❌	❌	❌

🔹 12. Custom Features (Hand-Crafted Features)

Sometimes, automatic text features (like TF–IDF or BoW) are not enough.
We can create our own domain-specific features, called custom features.

✳️ Examples (for Sentiment Analysis):

Feature	Description
Number of positive words	Count of words like "good", "excellent"
Number of negative words	Count of "bad", "terrible", "sad"
Positive/Negative ratio	pos_count / neg_count
Length of review	Word count or character count
Number of exclamation marks	Indicates emotion intensity
Presence of emojis or uppercase	"LOVED IT!! 😍" → strong sentiment

⚙️ Combination

In real-world NLP projects:
You combine automated features (TF–IDF, Word2Vec, etc.) + custom features
→ called Hybrid Features.
These improve model performance significantly.

🔹 13. Limitations Common to All Classical Techniques

1. Sparse representation
2. No semantic or contextual understanding
3. Fails on synonyms (e.g., *beautiful ≠ gorgeous*)
4. Sensitive to vocabulary changes
5. Can lead to overfitting with huge vocabularies

✅ That's why advanced methods like Word2Vec, GloVe, FastText, and BERT were developed — to learn semantic and contextual embeddings.

🏵️ Summary Table

Technique	Captures	Strength	Weakness
One-Hot	Presence of word	Simple	No meaning, sparse
BoW	Frequency	Counts words	Ignores order
N-Grams	Local context	Captures short sequences	High dimension
TF–IDF	Importance	Weighted words	Sparse, no semantics
Word2Vec	Meaning	Captures semantics	Needs large data

💬 Final Interview Summary

TF–IDF = "Statistical representation that weighs words by importance — high for frequent words in one doc, low for words common across corpus. Used widely in IR systems and NLP preprocessing."

5. Word Embeddings (Static) ⭐

- Represents words as dense vectors
- Captures semantic similarity

a) Word2Vec

- CBOW, Skip-gram models

b) GloVe

c) FastText

Example

king − man + woman ≈ queen

Pros

- Semantic meaning
- Low dimensional

Cons

- Same vector for a word in all contexts

Library

- gensim

---

6. Document Embeddings

- Represents entire sentences or documents

a) Doc2Vec

b) Averaged Word Embeddings

Pros

- Whole document representation

Cons

- Loses word-level details

Library

- gensim

---

7. Contextual Word Embeddings ⭐ **(MOST IMPORTANT)**

- Word meaning depends on context

Examples

- “bank” (river bank vs money bank)

Models

- BERT
- RoBERTa
- GPT

Pros

- Best semantic understanding
- State-of-the-art results

Cons

- Computationally expensive

Library

- transformers

---

8. Sentence Embeddings ⭐

- Represents entire sentence meaning

Examples

- Sentence-BERT
- Universal Sentence Encoder

Pros

- Ideal for similarity, search, QA

Library

- sentence-transformers

---

9. Character-Level Representation

- Represents text as characters instead of words

Pros

- Handles spelling errors
- Handles OOV words

Cons

- Longer sequences

Library

- keras, torchtext

---

10. Subword-Level Representation ⭐

- Breaks words into subwords (BPE, WordPiece)

Example

unhappiness → un + happy + ness

Pros

- Handles unknown words well

Library

- sentencepiece, tokenizers

---

11. Topic Modeling Representation

- Represents documents by topic distribution

Examples

- LDA
- LSA

Pros

- Interpretability

Library

- gensim

---

12. Graph-Based Representation

- Represents text as dependency or knowledge graphs

Pros

- Useful for reasoning tasks

Library

- networkx, spacy

🧠 **EXAM ONE-LINE ANSWER**

Text representation techniques convert textual data into numerical vectors using frequency-based, embedding-based, or contextual approaches so that machines can process language.

🔑 **QUICK SELECTION GUIDE**

Task	Best Technique
Text Classification	TF-IDF, BERT
Sentiment Analysis	Word Embeddings, BERT
Search / Similarity	Sentence Embeddings
Topic Modeling	LDA
Chatbots	Contextual Embeddings