Intelligent Systems,
MIPT

Natural Language Processing

L1 / 10.02.26

- Course page
- GitHub

Content

Lecture 1: Intro to NLP, Basic Text Processing, Tokenization

- About the course
- NLP Research Questions and Tasks
- Text Segmentation, Stemming, Lemmatization
- Tokenization: BPE / Word Piece / Unigram

Course logistics in brief

- **Online classes: Tue 14:30 – 16:00**
- **Course consists of 14 lectures & seminars**
- **Approximate syllabus:**

  > NLP Basics: tokenization, text preprocessing, text representations
  > Text & Language Models: embeddings, n-gram models, RNNs, LSTMs, seq2seq, attention
  > Transformers & LLMs: Transformer, pre-training (MLM/CLM), prompting, fine-tuning, PEFT
  > Scaling & Optimization: distributed training, MoE, efficient inference, quantization
  > Retrieval & Agents: Information Retrieval, RAG, agent-based systems
  > Post-training: alignment, RLHF, DPO

- **Final mark = 0.3 Oral Exam + 0.7 Homework**
- **Communication:**

Homework & Exam & Policy

- **Homework consists of 4 programming assignments**

  > For most of the homeworks you will need GPU access. Kaggle / Colab will be enough, otherwise it will be agreed.
  > Possibly we have some bugs in code :(Don’t hesitate to contact us. If you let us know after the deadline, we can’t help you.
  > Bug fixes and homework suggestions (fully implemented in the code) are encouraged and rewarded!

- **All submitted content should be your own content, written yourself! However, you may discuss the homework with others in the chat.**

- **The exam is taken at the end of the course on the topics covered. Theoretical minimum and the main ticket.**

Aim of the course

The purpose of this course is to build a modern, end-to-end NLP mindset with enough mathematical foundations to reason about methods and evaluation:

> Formulate NLP problems and evaluate them correctly

> Understand text as a mathematical object

> Master current Transformer workflows

> Build scalable and practical NLP systems

Team

German Gritsai

Bachelor's: MIPT
Master's: MIPT, UGA

Senior NLP Engineer @
Antiplagiat

Anastasiia Vozniuk

Bachelor's: MIPT
Master's: MIPT, TU Graz

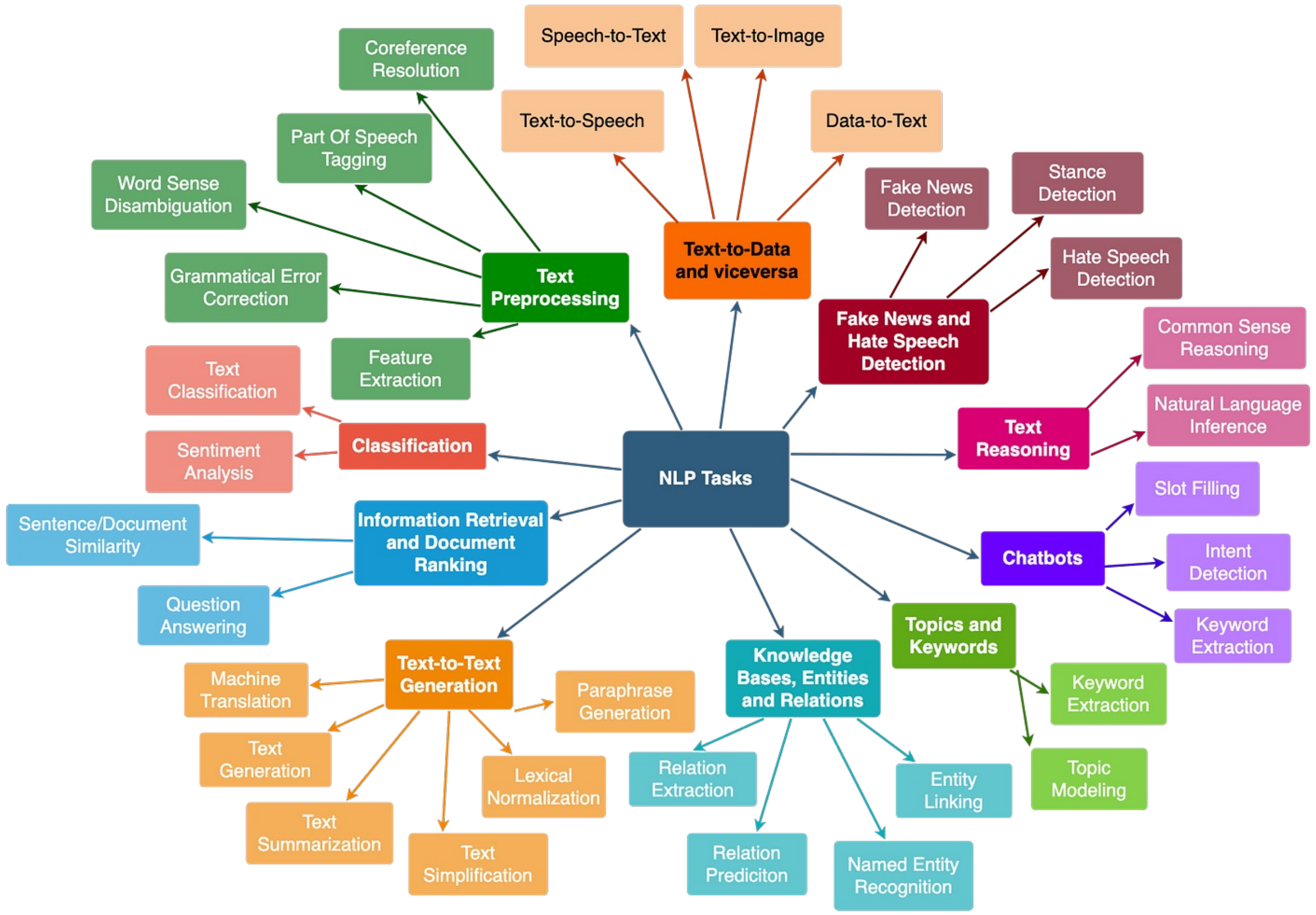
Senior NLP Engineer @
Antiplagiat

Ildar Khabutdinov

Bachelor's: MIPT
Master's: MIPT, UGA

Senior NLP Engineer @
Sber, GigaChat

Tasks overview



Example: NLP Tasks

"Yesterday I visited 'Sakhalin' restaurant on Tverskaya Street. I ordered soup and salmon sashimi. They were fresh, but the waiter Ivan was inattentive. Prices are high, but the interior is beautiful. I'll return for the food, but not because of the service."

NER

"Who? Where? What entities appear?"

Sakhalin	→ **ORG**
Tverskaya Street	→ **LOC**
Ivan	→ **PERSON**
soup, salmon	→ **FOOD**

SENT. ANALYSIS

"What is the overall sentiment?"

Overall	→ **mixed/neutral**
— food	→ **positive**
— service	→ **negative**
— prices	→ **negative**
— interior	→ **positive**

CATEGORIZATION

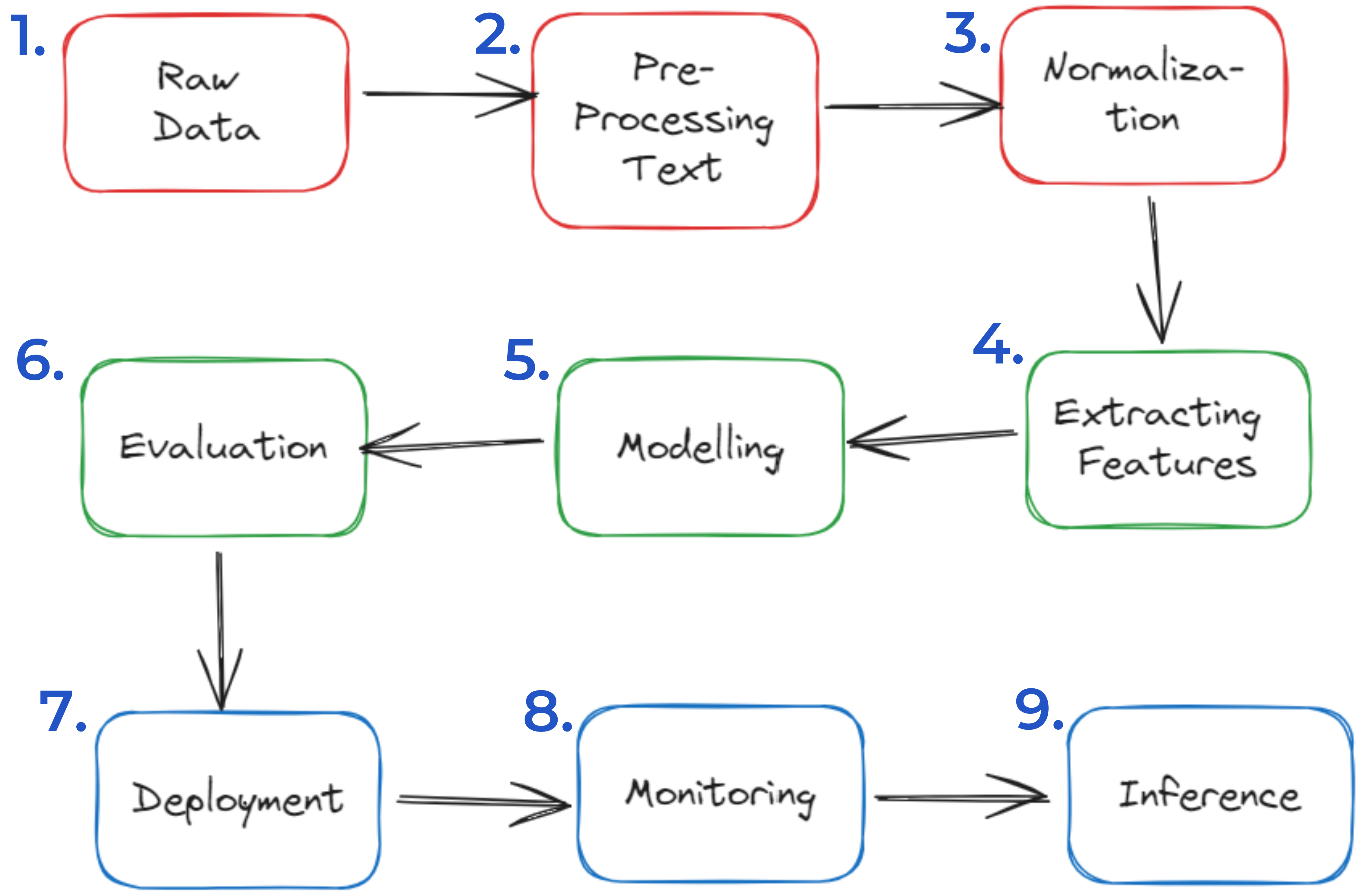
"What category/genre does this belong to?"

restaurant review
(vs. news article, recipe, ads)

Example: NLP Tasks

ASR	🎤 ***"find Japanese restaurants on Tverskaya"***	→	*"What was spoken?"*	→	`"find Japanese restaurants on Tverskaya"`
MACHINE TRANSLATION	***"Вчера я посетил ресторан «Сахалин» на Тверской"***	→	*“How to express this in English?"*	→	***"Yesterday I visited 'Sakhalin' restaurant on Tverskaya Street"***
QUESTION ANSWERING	***"I ordered soup and salmon sashimi. They were fresh, but ..."*** *Context*		***"Was the food fresh?"*** *Question*	→	***”fresh” / “yes”***
DIALOGUE SYSTEMS	***"Is Sakhalin good for dinner?"***	→	*"How should a system respond helpfully?»*	→	***“Sakhalin has fresh seafood and a beautiful interior, but some guests mention slow service. Would you like reservation help?"***
TEXT SUMMARIZATION	***"Yesterday I visited... I'll return for the food, but not because of the service."***	→	*"What's the essence in one sentence?”*	→	***"Mixed review of 'Sakhalin': praised food and interior, criticized service and prices."***

NLP pipeline



Datasets in the field

- **Dataset types**

  > Supervised / weak / synthetic
  > In-domain vs out-of-domain (domain shift)
  > Multilingual effects (tokenization, morphology)

- **Label quality**

  > Ambiguity → annotator disagreement
  > Noise sets an upper bound
  > Always do error analysis

- **LLM pretraining data**

  > Web-scale data with heavy filtering and deduplication
  > Hidden biases and possible contamination of benchmarks

Text preprocessing

Let D be a collection of text documents.
Each document is a raw string containing both alphabetic and non-alphabetic characters.
Processing text in this form is impractical — machine learning requires numeric features. Therefore, we must first transform the data into a structured representation and normalize it.

- Regular expressions
- Lemmatization and Stemming
- Tokenization
- Parts of Speech (POS)

Regular expressions

- A string template that can be used to check text for compliance
- There are many tools and online services for working with regular expressions. Python has two main modules: re and regex
- Good for: removing noise with fixed patterns [URLs, emails, phone numbers, @mentions, hashtags]
  Not good for: context-sensitive tasks [U.S.A., don't]

Pattern	Matches
`r"[mM]ary"`	`Mary` or `mary`
`r"[1234567890]"`	Any one digit

Pattern	Matches	Examples
`r"[^A-Z]"`	Not upper case	`Oyfn pripetchik`
`r"[^Ss]"`	Neither 'S' nor 's'	`I have no exquisite reason"`
`r"[^.]"`	Not a period	`Our resident Djinn`
`r"[e^]"`	Either e or ^	`Look up ^ now`

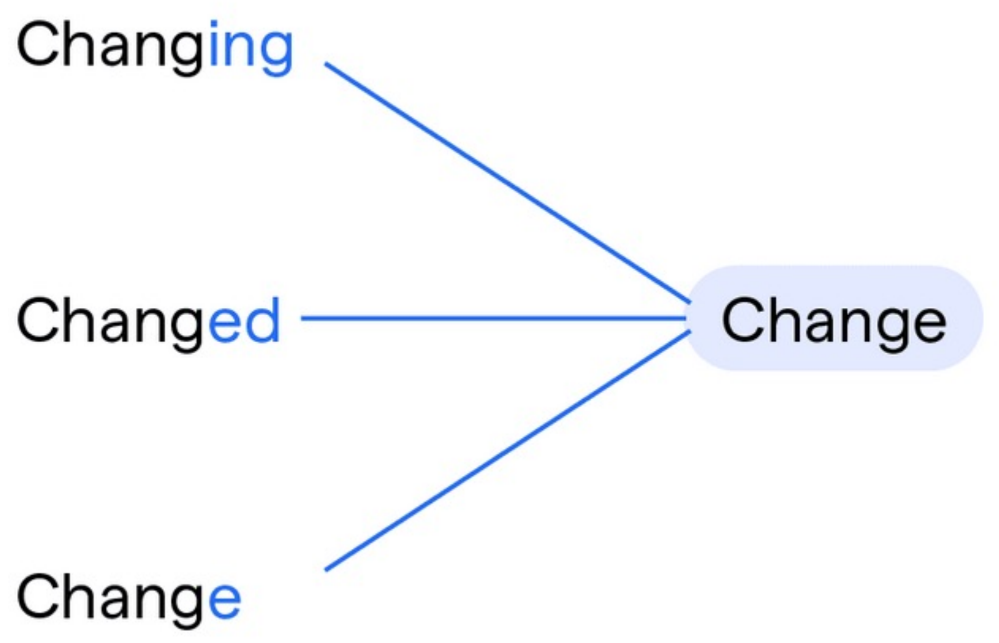
Lemmatization

- Text preprocessing normalization task concerned with bringing words down to their root forms

- Improves performance in sparse-data scenarios (topic modeling, traditional ML)

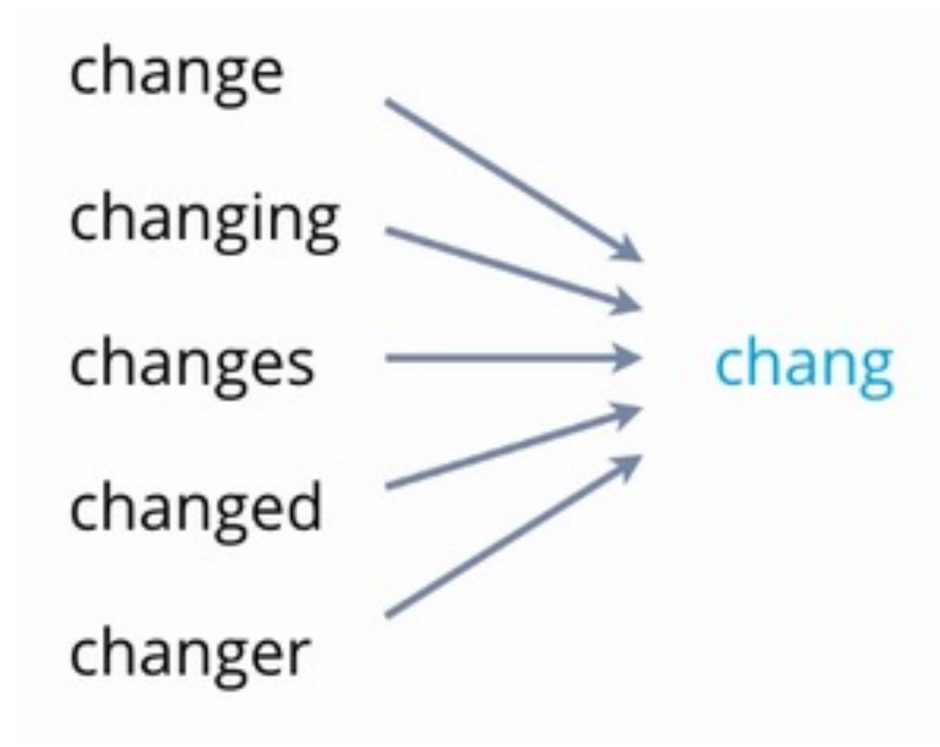
- Lemmatizer uses POS tag to disambiguate:

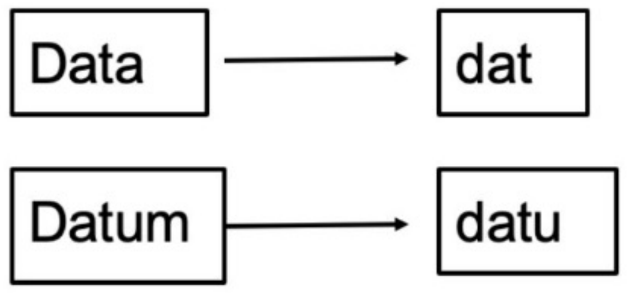
  > leaves → leaf (noun)
  > but leaves in "he leaves the room" → leave (verb)

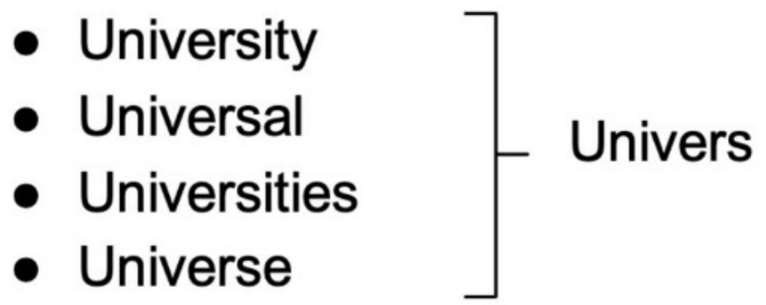


Stemming

- Text preprocessing normalization task concerned with bluntly removing word affixes (prefixes and suffixes) to keep the root
- Mechanical chopping, no POS or context
- Porter [1979], Snowball [Porter 2], Lancaster [1990]
- Understemming & overstemming







Practical tools

Regular expressions	re, regex, urlextract
Normalization	unicodedata, unidecode
Stemming	nltk.stem.PorterStemmer, nltk.stem.SnowballStemmer, pystem2
Lemmatization	spaCy, pymorphy2, nltk.WordNetLemmatizer, stanza,
Stop Words	nltk.corpus.stopwords

- Tokenization ...

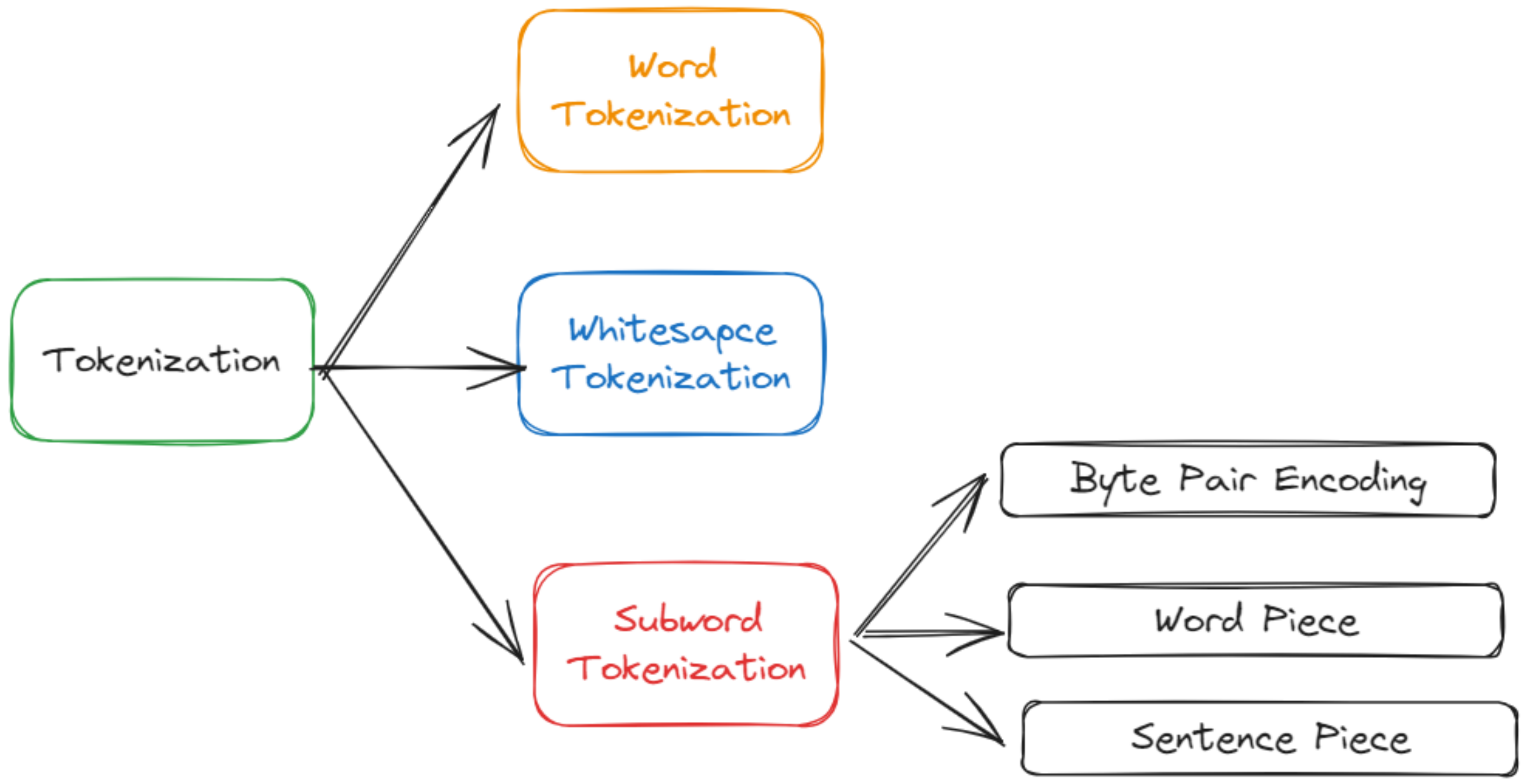
Tokenization

- Breaking down text into smaller, manageable units called tokens
- These tokens can be words, subwords, characters, or other meaningful units, depending on the tokenization strategy used

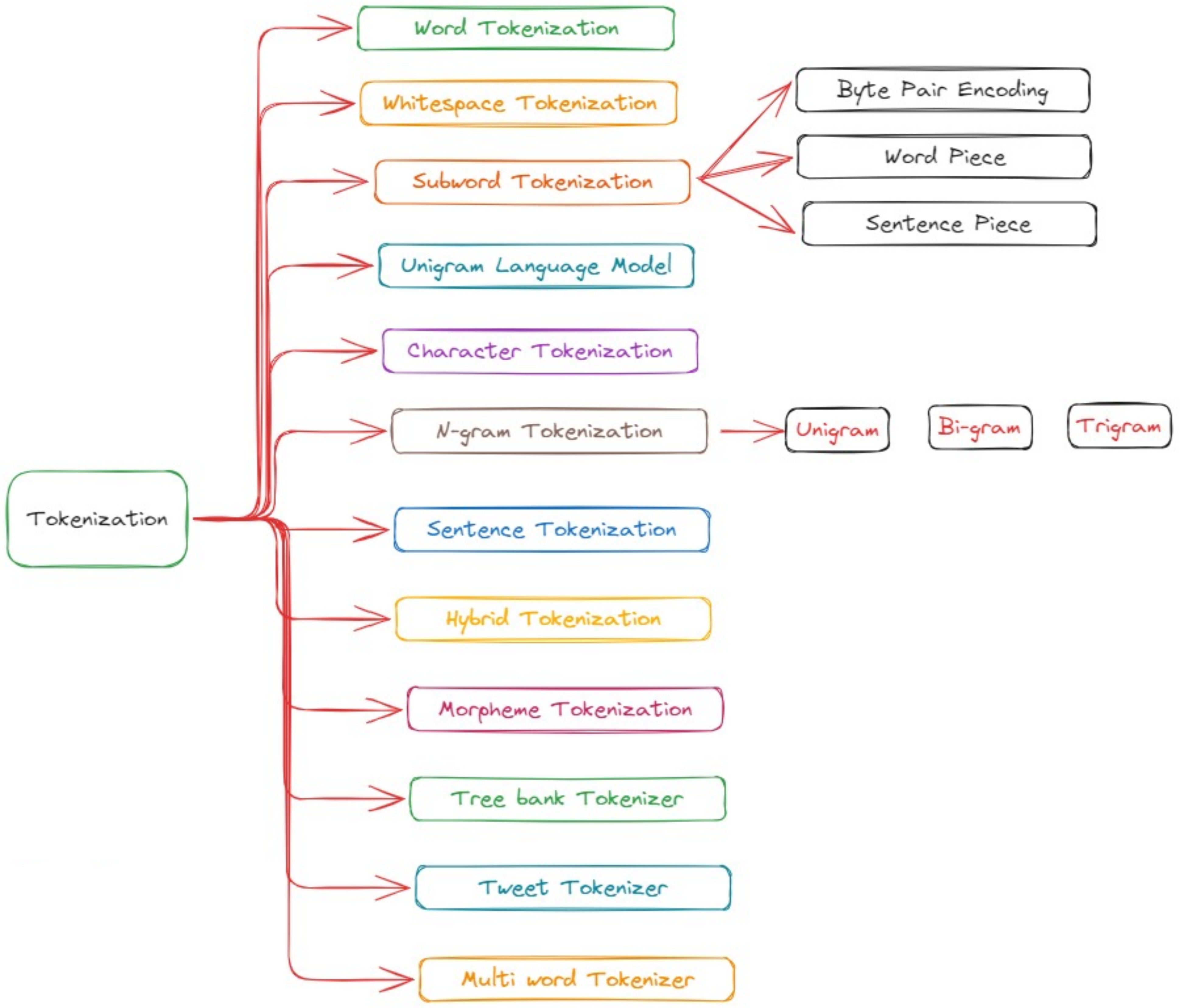
> Character-level: "Hello!" → ['H', 'e', 'l', 'l', 'o', '!']
> Word-level: "Hello, world!" → ['Hello,', 'world!']
> Whitespace: "Hi world!" → ['Hi,', 'world!']

> Subword: "unhappiness" → ["un", "happi", "ness"]
"running" → ["run", "ning"]

- The model operates with indices rather than symbols. Without tokenization, there is no input to the model



Tokenization Overview



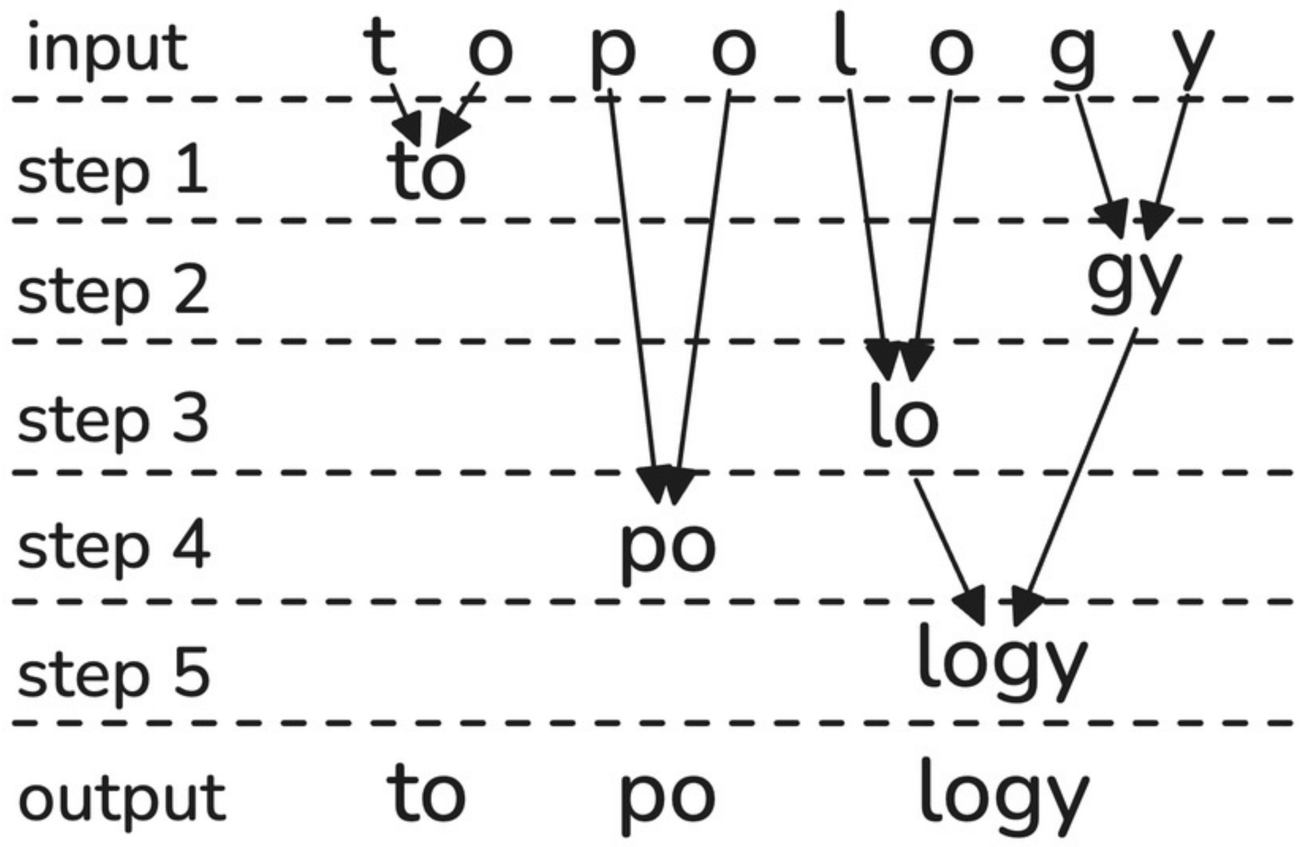
Subword: BPE

- Tokenization technique that iteratively merges the most frequent pairs of characters or subwords in the text to create a compact and efficient vocabulary

  > Example 1: "unhappiness" might be tokenized as ["un", "happiness"].
  > Example 2: "lowering" might be tokenized as ["low", "er", "ing"]

- Widely used: GPT, RoBERTa

- Good for: compact vocabulary, handle rare words
  Not good: arbitrary segmentation, data dependency, longer seq



Subword: Word Piece

- Tokenization technique similar to BPE but differs slightly in how it selects subword units: $\text{score} = (\text{freq\_of\_pair})/(\text{freq\_of\_first\_element} \times \text{freq\_of\_second\_element})$

  > Example 1: "unhappiness" might be tokenized as ['un', '##happiness’]
  > Example 2: ”running" might be tokenized as ['run', '##ning']

- Widely used: BERT

- Good for: compact vocabulary, handle rare words, language-agnostic, reduced OOV
  Not good: complexity in training, data dependency, longer seq

Subword: Unigram

- Probabilistic model (SLM) that uses individual tokens (subwords or words) to estimate the probability of a sequence of tokens in a given text

  > Example 1: "unhappiness" might be tokenized as ["un", "happ", "iness"]

- Widely used: T5, GPT-3

- Starts with an excessively large vocabulary → iteratively removes the least probable tokens until the target size is reached

```
While |V| > target_size:
    1. For each sentence in the corpus:
       └ Find the optimal token segmentation using the current V
         (Viterbi algorithm / dynamic programming)

    2. For each token t ∈ V, estimate its "importance":
       └ How much would the overall corpus likelihood decrease
         if t were removed from the dictionary?
       └ Formula: loss(t) = log p(t) × count(t)
         (tokens with a small contribution to likelihood are candidates for removal)

    3. Remove the p% least important tokens (typically p = 10–20%)
```

Practical tools: tokenization

Word-level	spaCy, NLTK, pymorphy
BPE	tokenizers.BPETokenizer, GPT2Tokenizer
Word Piece	BertTokenizer
Unigram	tokenizers.Unigram, XLNetTokenizer

Average vocab size: from 32k to 256k

If you are using a transformer → tokenization only (no lemmatization before it!)

If you are using classical models → tokenization + lemmatization / stemming

Recap

- NLP = task formulation + data + evaluation (not just a model)
- Text → tokens → vectors → model
- Subword tokenization (BPE / WordPiece / Unigram) solves OOV but changes cost + errors.
- Datasets & preprocessing

Next:

- Deep dive into vector representations