
IRT
SAINT
EXUPÉRY

NLP to LLMs

Master AIBT/ValDom

NLP to LLMs

mouhcine.mendil@irt-saintexupery.com

Mouhcine MENDIL

Topics:
- NLP tasks
- Classical and advanced preprocessing for NLP
- ML Workflow for NLP

joseba.dalmau@irt-saintexupery.com

Joseba DALMAU

Topics:
- Classical DL techniques
- Transformers
- Feature space

Time Schedule

	Monday 02/02	Tuesday 03/02	Wednesday 04/02	Thursday 05/02	Friday 06/02
8h45-10h30	CM - M	BE - J/M	CM - J	CM - J	
10h45-12h30	CM - M	BE - J/M	CM - J	BE - J/M	
Lunch Break					
13h45-15h30	CM - M	BE - J/M	BE - J/M		Exam - J/M 14h-16h
15h45-17h30		BE - J/M	BE - J/M		

NLP to LLMs

Written Exam: February 6, 14h00-15h45

MCQs + course questions + exercises

- The students will be allowed a single A4 two-sided handwritten sheet of paper with the notes they deem useful for the exam.
- The instructors will provide the students with sample exercises so that they can train themselves for the exam.

Ressources

GitHub
https://github.com/jdalch/Valdom-NLP2LLM

NLP Tasks

Text Classification

Text Classification

Goal: assign predefined labels or categories to text

Process:

- The model processes the input as a whole to predict the most probable label (e.g., "positive," "negative," or "neutral").
- The final prediction corresponds to the label with the highest probability.

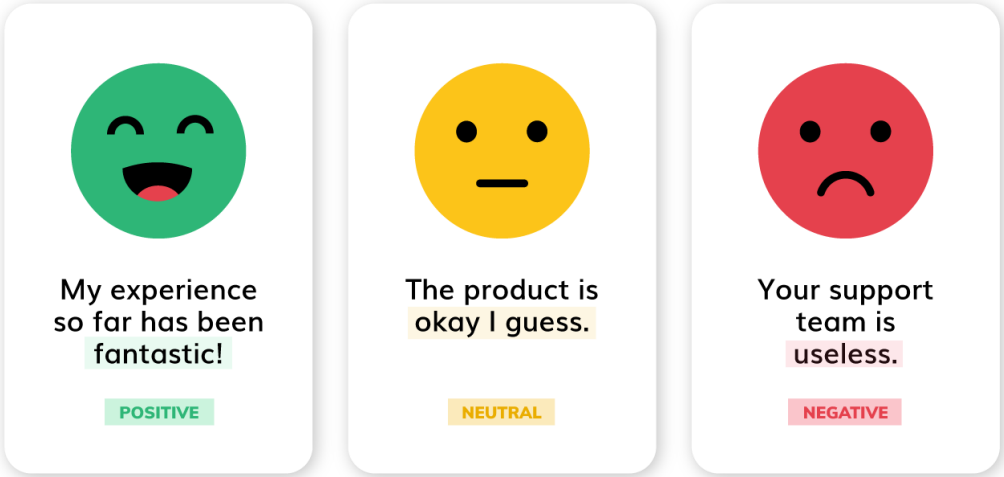
Key Insight:
The model considers the entire input text at once to make its decision.

Sentiment Analysis

Goal : Determine whether a text expresses a positive, negative, or neutral opinion.

Example :

- *Text*: “I love this product!”
- *Output*: “Positive feeling”



Language Detection

Goal : Identify the language of a text.

Example :

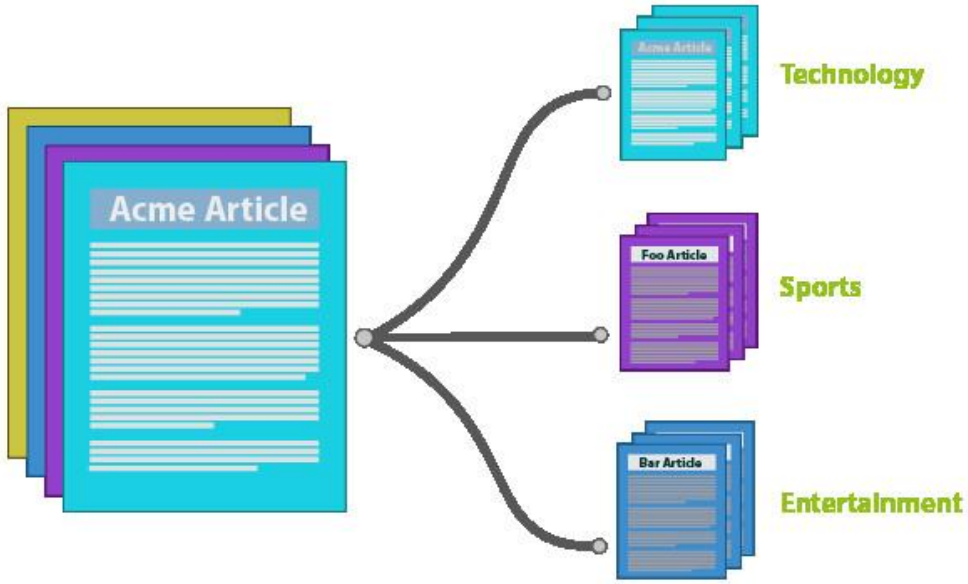
- *Text*: “Hello, how are you?”
- *Output*: “Language: French”

Topic Classification

Goal : Identify a text's topic or theme.

Example :

- *Text*: “PSG won the match last night.”
- *Output*: “Theme: Sport”



Grammatical Classification

Goal : Identify whether a sentence is grammatically correct.

Example :

- *Text*: “The cat eats mouse one.”
- *Output*: “Incorrect sentence”

Text Classification (general)

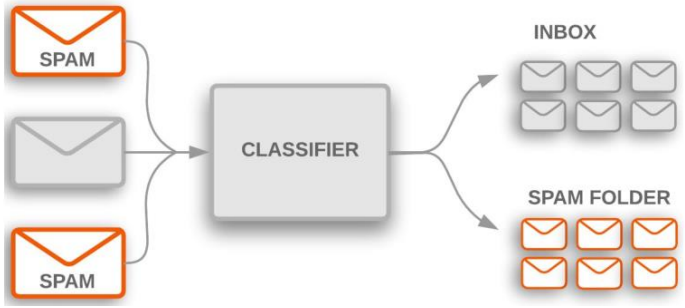
Goal : Assign a general category to a text (binary, multi-class or multi-label).

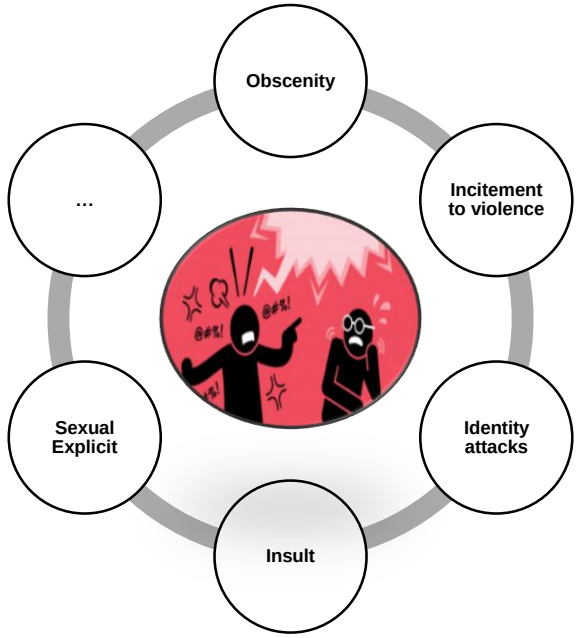
Text Classification (general)

Goal : Assign a general category to a text (binary, multi-class or multi-label).

Example :

- *Binary classification*: Spam or non-spam in e-mails.
- *Multi-class classification*: Article categories (sport, politics, science).
- *Multi-label classification*: Assign multiple labels (e.g. “politics” and “environment”).





NLP Tasks

Information Extraction

Information Extraction

Goal: Assigning a label to each token in a sequence.

Process:

- The input text is split into smaller units (tokens).
- For each token, the model predicts a label.
- Labels are combined to extract structured information (e.g., "John Doe" → person name).

Key Insight:
The model processes each token individually, but uses context from the entire sentence for predictions.

Named Entity Recognition - NER

Goal : Identify specific entities in a text and classify them (People, Places, Organizations, etc.).

Example :

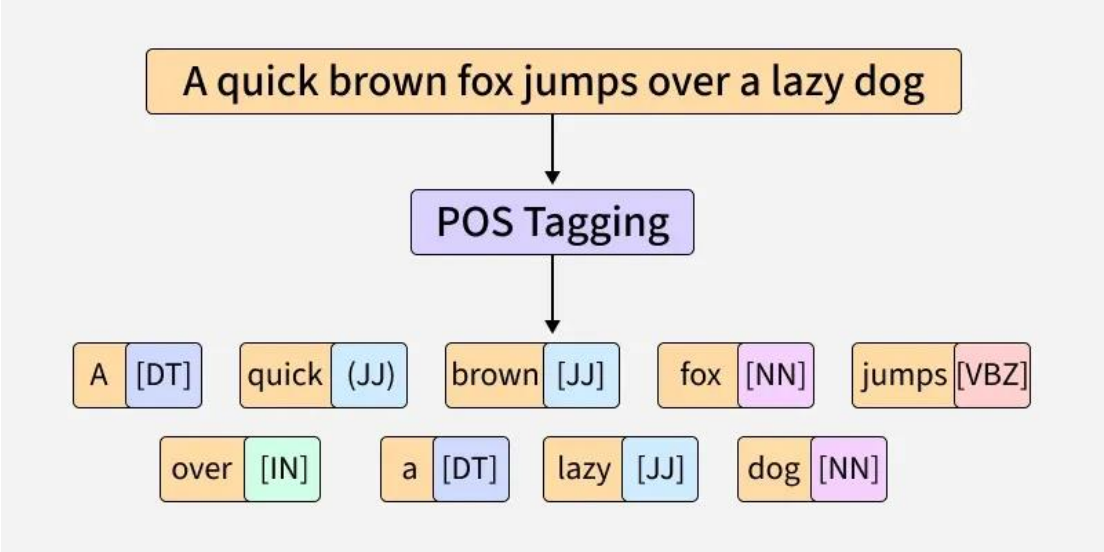
- *Text*: “Marie left for Paris yesterday.”
- *Output*: {“Marie” : Person, “Paris” : Place, “yesterday” : Date}

Part-of-Speech (POS) Tagging

Goal : Assign a grammatical class to each word (verb, noun, adjective, etc.).

Example :

- *Text*: “The cat eats a mouse.”
- *Output*: {“The”: Determiner, “cat”: Noun, “eats”: Verb, “mouse”: Noun}

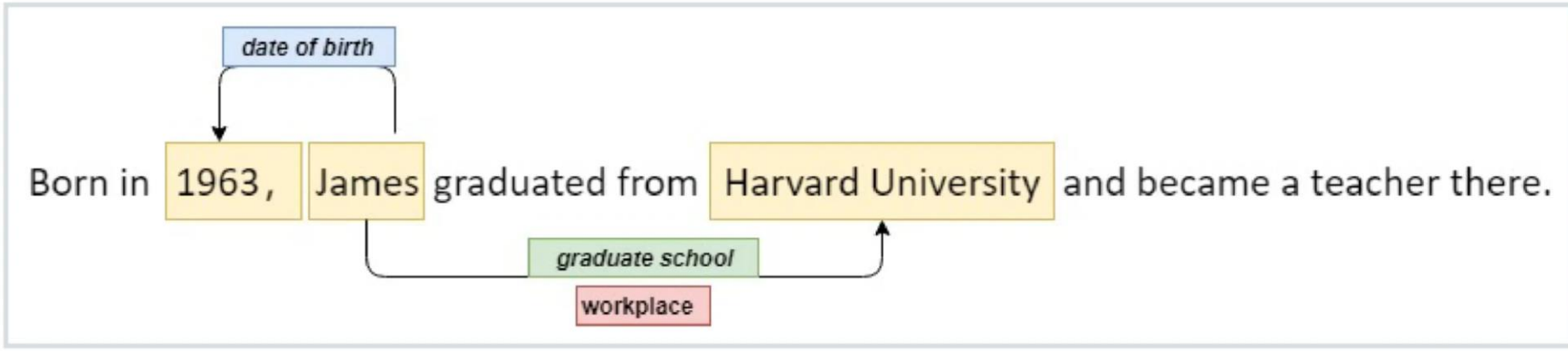


Relation Extraction

Goal : Identify and classify relationships between entities in a text.

Example :

- *Text*: “Marie works at Google.”
- *Output*: {“Marie”, “employee”, “Google”}



NLP Tasks

Text Generation

Text Generation

Goal: Producing coherent and contextually relevant text based on given input

Process:

- The model predicts the next token based on the input and previously generated tokens.
- At each step, it selects the most probable token from the model's vocabulary.
- This process is repeated recursively until the text is complete (or a stop condition is reached).

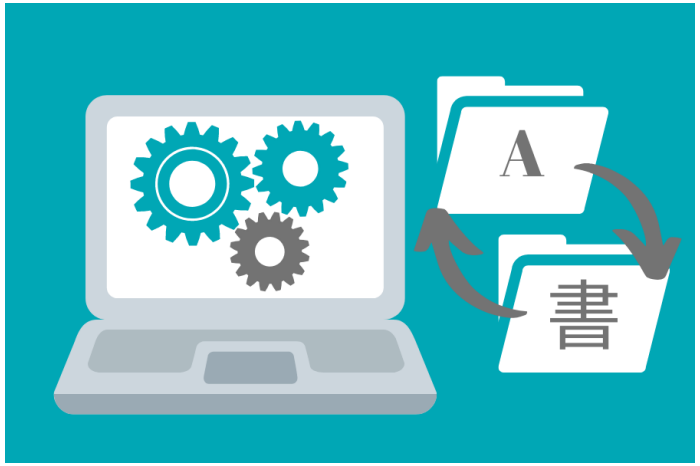
Key Insight:
Text generation is a sequential process where each token depends on what has already been generated.

Machine Translation

Goal : Translate a text from a source language to a target language.

Example :

- *Text*: "Hello, how are you?"
- *Output*: "Bonjour, comment ça va ?"

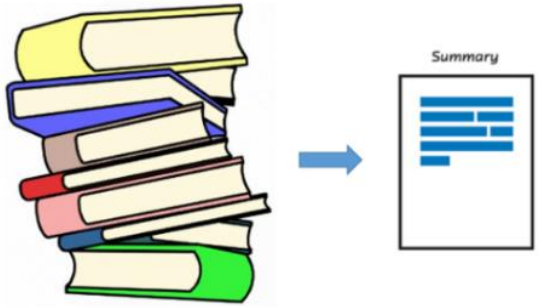


Text Summarization

Goal : Generate a concise, informative summary from a longer text.

Example :

- *Text*: “Artificial intelligence is revolutionizing many sectors, especially medicine, where it is enabling better diagnostics.”
- *Output*: “AI improves medical diagnostics.”

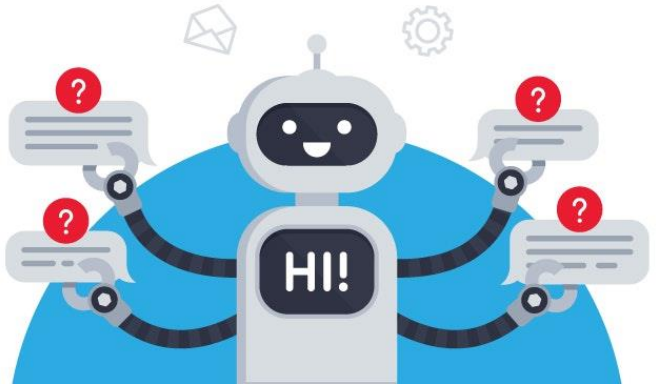


Question Answering - QA

Goal : Generate a text response based on a question and context.

Example :

- *Context*: “The Eiffel Tower was built in 1889.”
- *Question*: “When was the Eiffel Tower built?”
- *Output*: “In 1889.”



Free Text Generation

Goal : Generate fluid text from minimal input or a prompt.

Example :

- *Text*: “Tell a story about a magical cat.”
- *Output*: “Once upon a time, there was a cat who lived in an enchanted forest and could talk to animals.”

Image-to-Text

Goal : Generate a text description from an image.

Example :

- *Input*:

- *Output*: “A baby dog sleeping on a beige blanket.”

Data-to-Text Generation

Goal : Convert structured data into natural text.

Example :

- *Input*:

Name	Age	Occupation
John	30	Doctor

- *Output*: “John is 30 and a doctor.”

Data-to-Text Generation

Goal : Convert structured data into natural text.

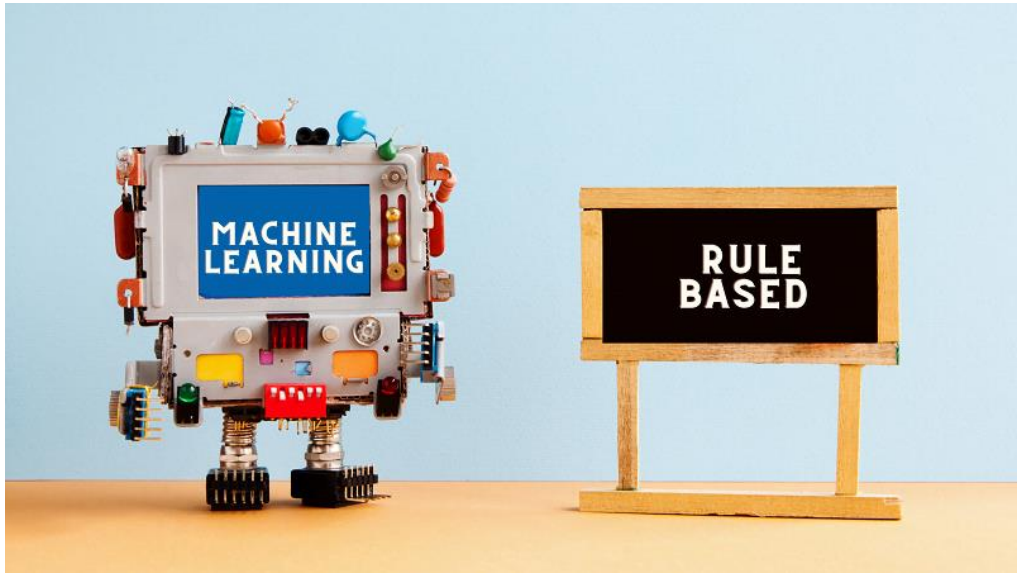
Example :

- *Input*:

- *Output*: “Hello everyone!”

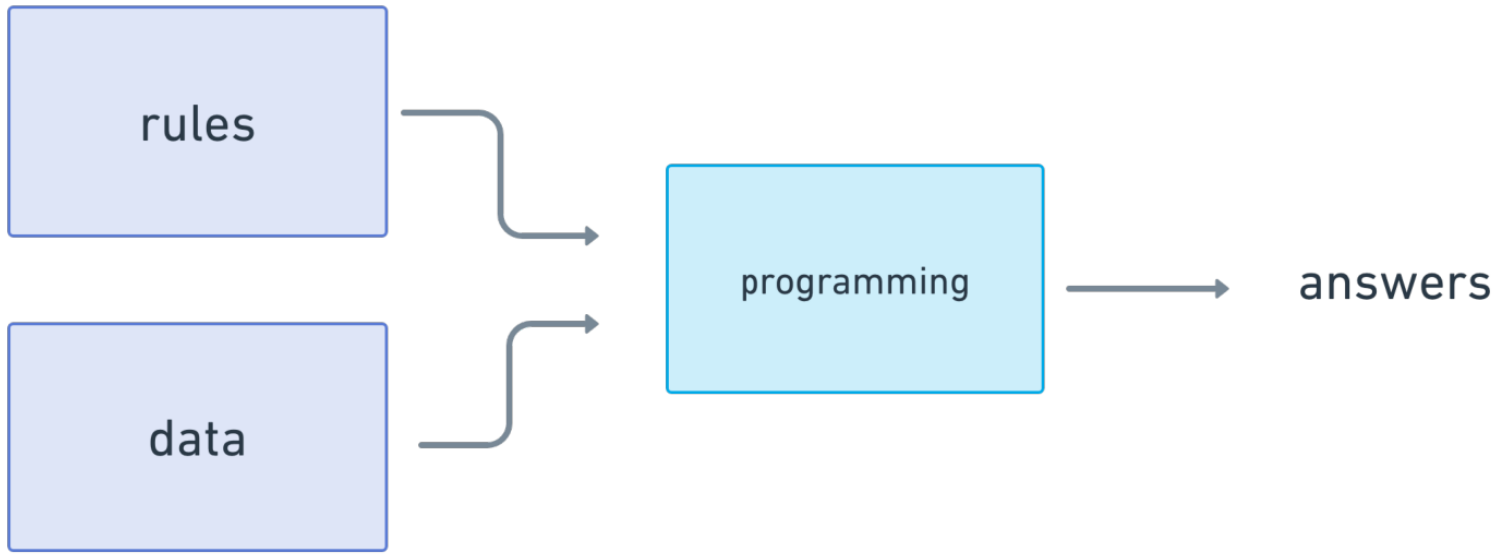
Questions ?

NLP: Rule Based vs ML



Rules-based Systems

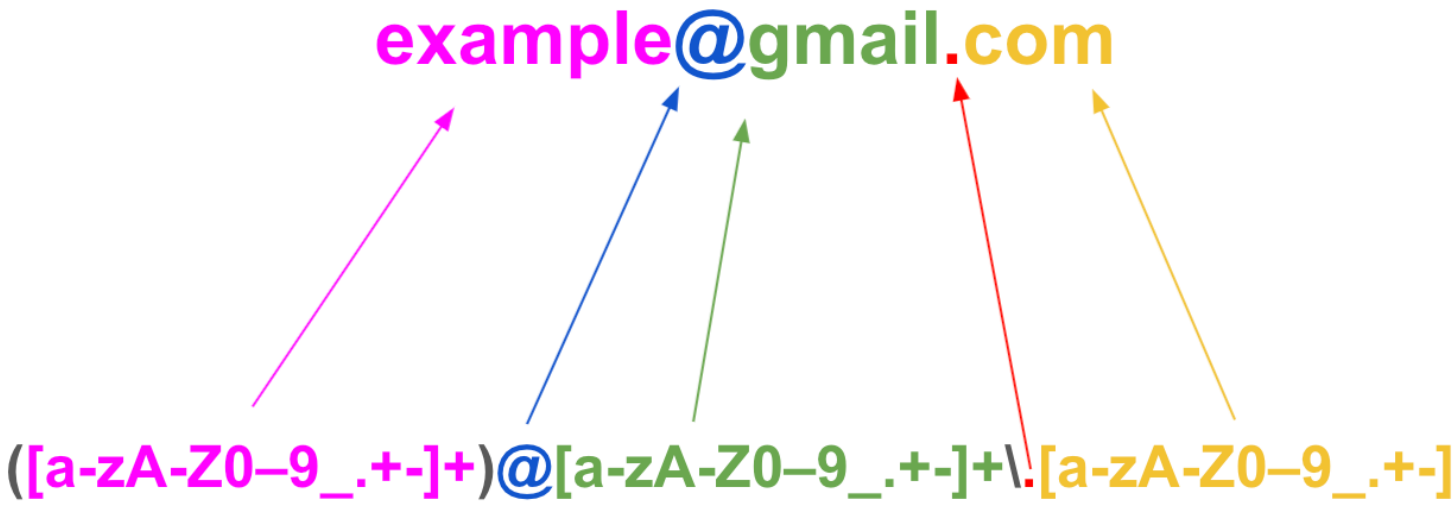
- Early approaches (60’s): symbolic methods, hand-written rules



FRENCH INSTITUTES OF TECHNOLOGY

Rules-based Systems

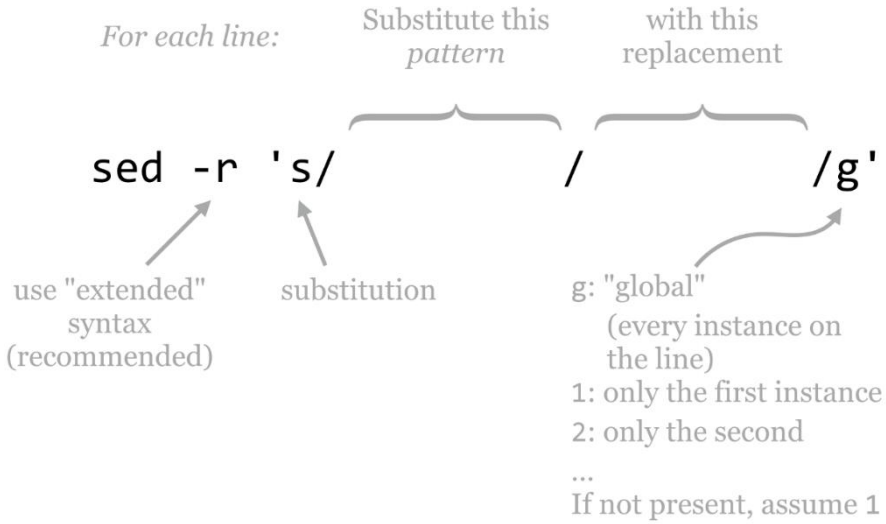
- Regular expressions: language for specifying a matching pattern in text search



Rules-based Systems

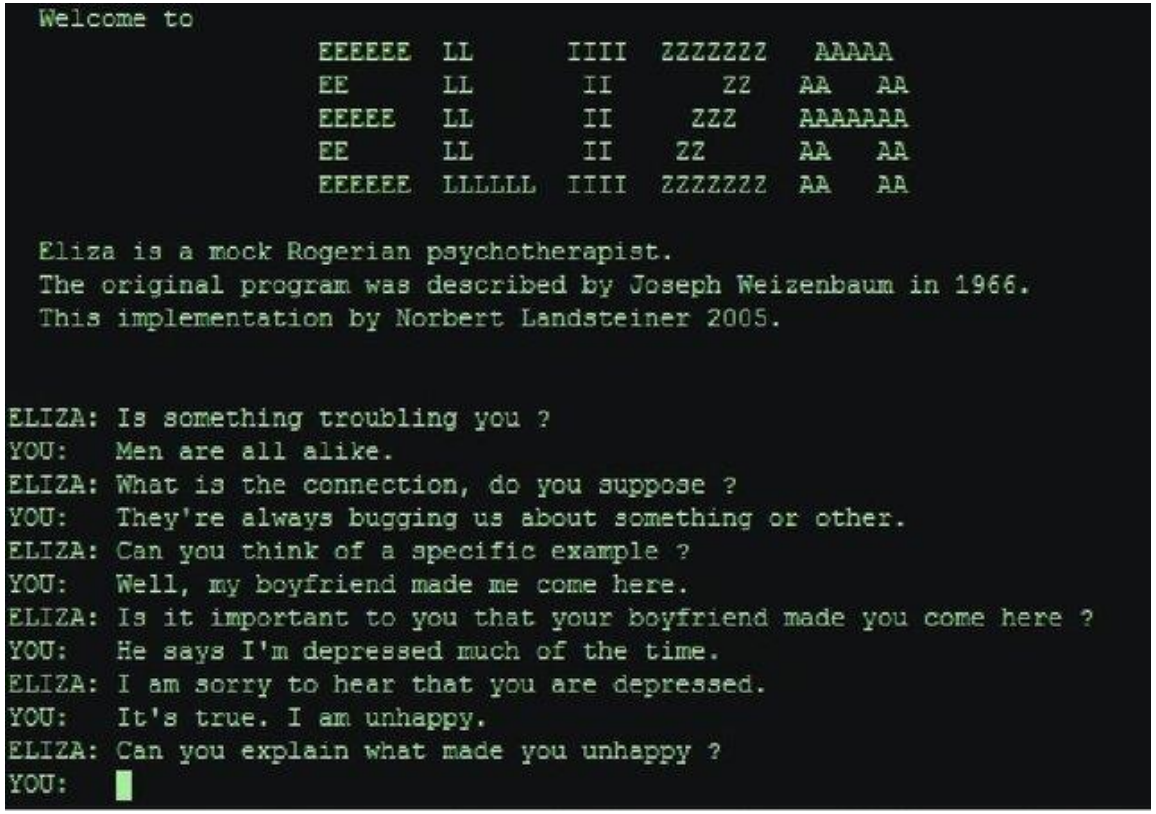
- Regular expressions: language for specifying a matching pattern in text search
- Useful for text normalization and substitution

```
In [3]: import re

        text = "i love python and PYTHON is great for many tasks"
        pattern = r'\bpython\b'
        repl = "Python"
        result = re.sub(pattern, repl, text, flags=re.IGNORECASE)

In [4]: print(result)

        i love Python and Python is great for many tasks
```



Rules-based Systems

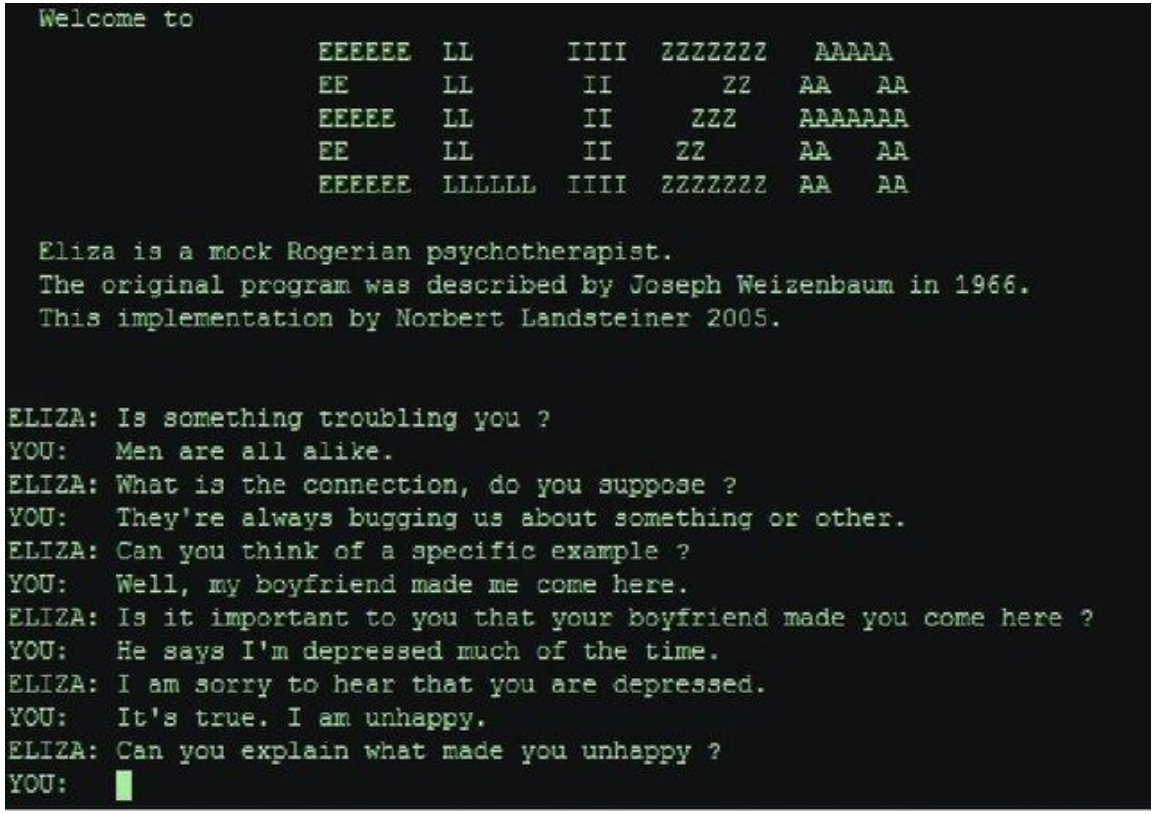
- ELIZA simulates a Rogerian psychologist



https://www.masswerk.at/elizabot/

Rules-based Systems

- ELIZA simulates a Rogerian psychologist
- Hard-coded rules based on pattern matching and substitution



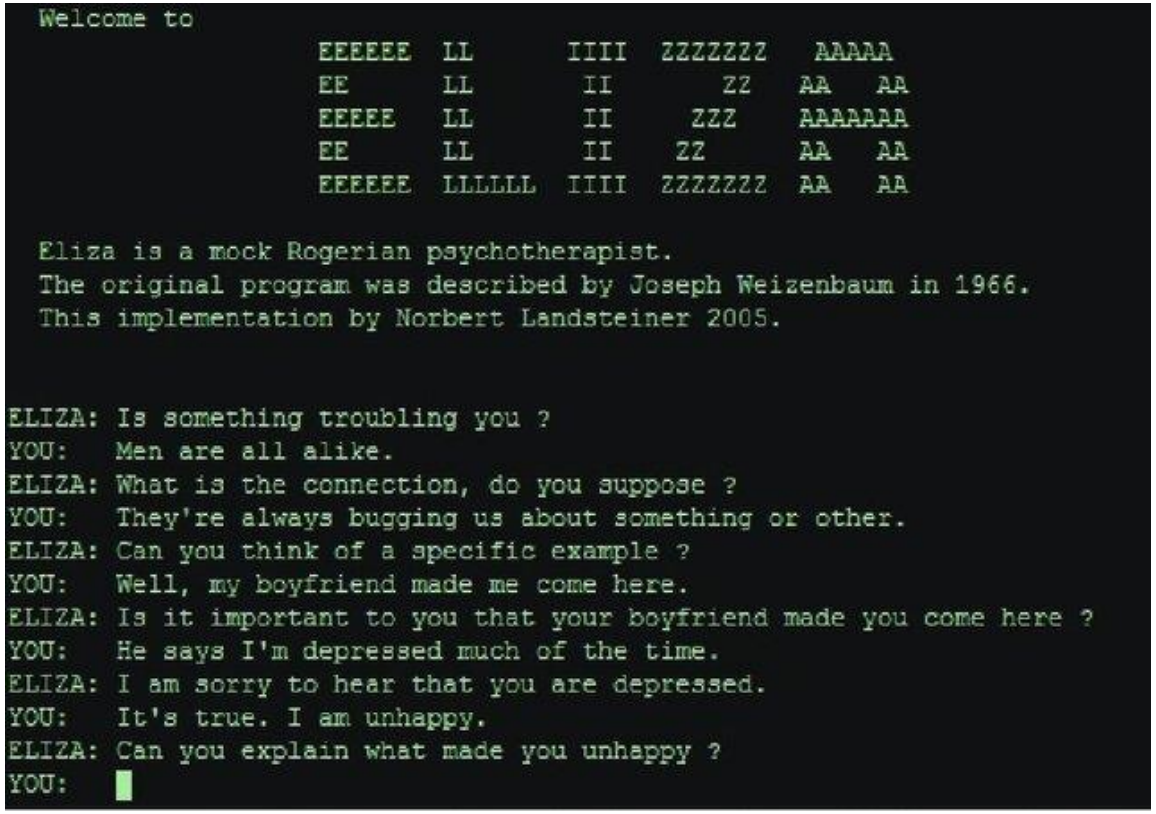
https://www.masswerk.at/elizabot/

Rules-based Systems

- ELIZA simulates a Rogerian psychologist
- Hard-coded rules based on pattern matching and substitution

Example of a condition and possible answers:

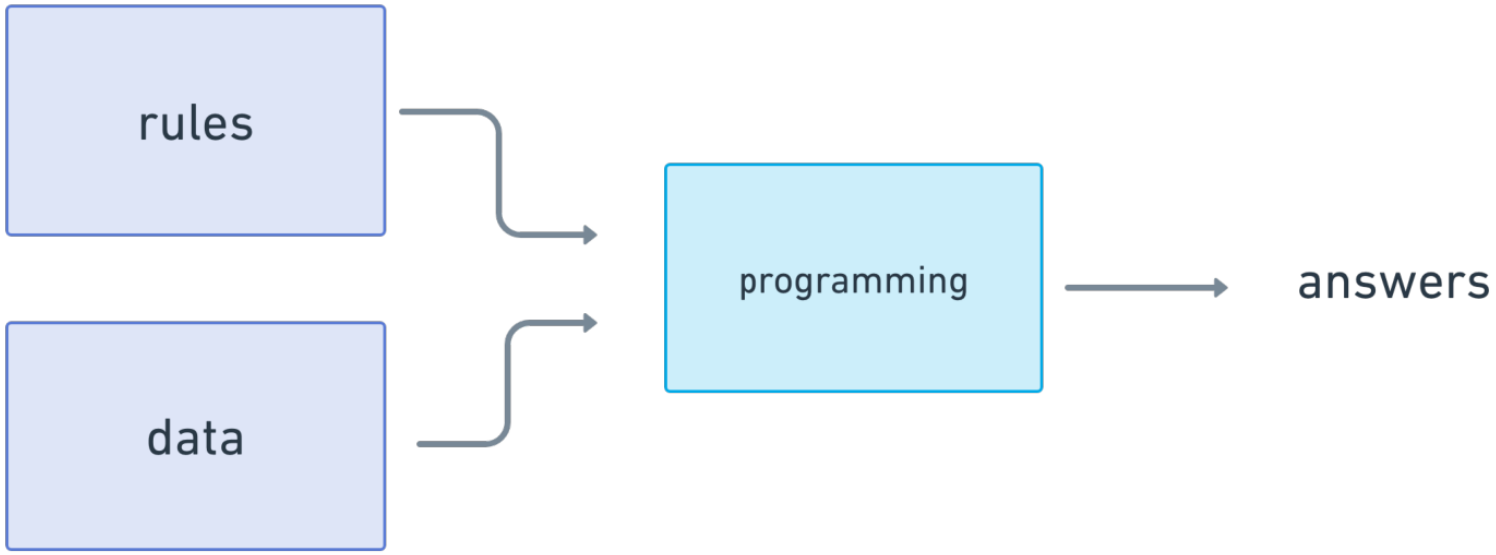
```
"?*x I want ?*y": [
        "What would it mean if you got ?y?",
        "Why do you want ?y?",
        "Suppose you got ?y soon.“
        …
        ]
```



https://www.masswerk.at/elizabot/

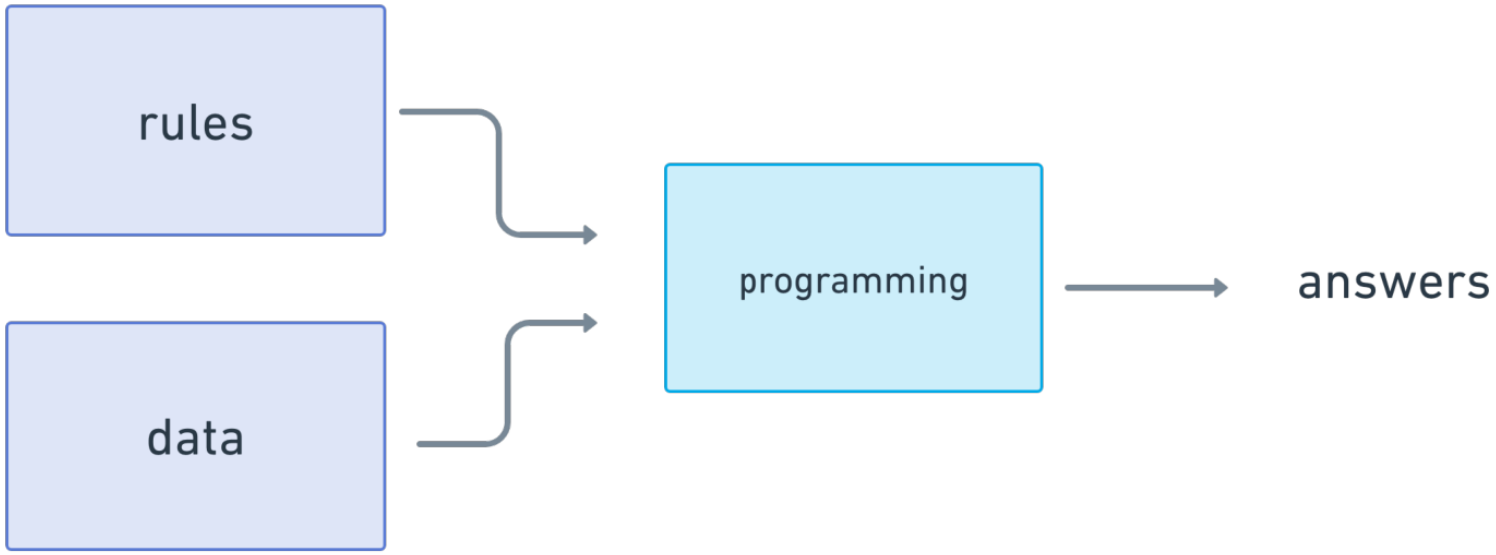
Rules-based Systems

- Early approaches (60’s): symbolic methods, hand-written rules
- Advantages: based on expert knowledge, precise



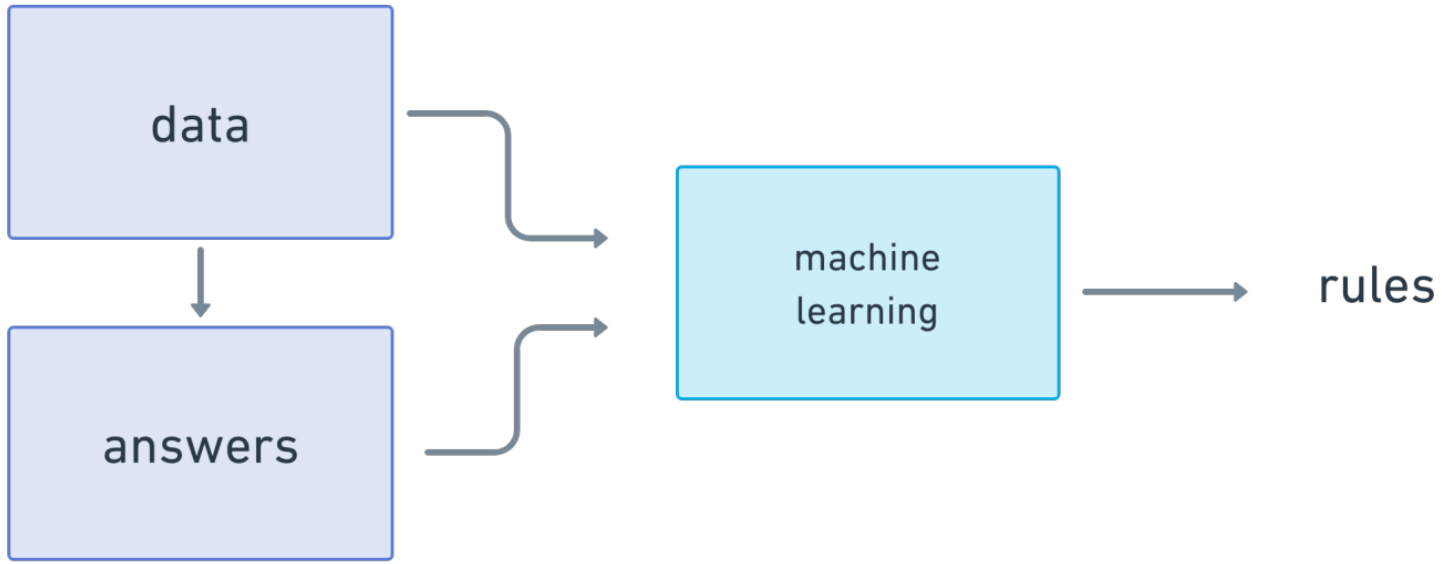
Rules-based Systems

- Early approaches (60’s): symbolic methods, hand-written rules
- Advantages: based on expert knowledge, precise
- Downsides: lack of coverage, expensive to build and maintain



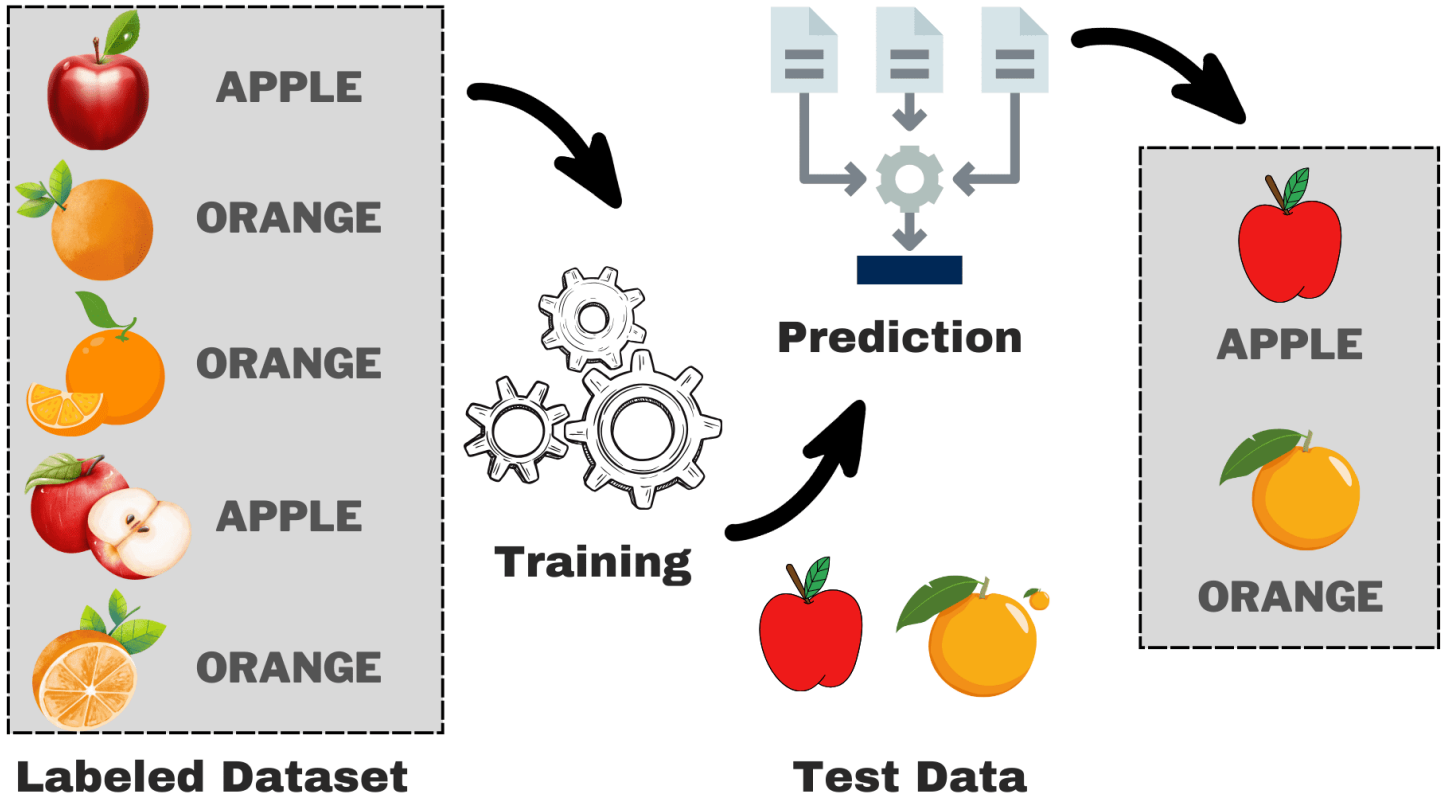
Learning from Data (Supervised)

- Learn rules automatically: machine learning (90's), neural methods (2010's)



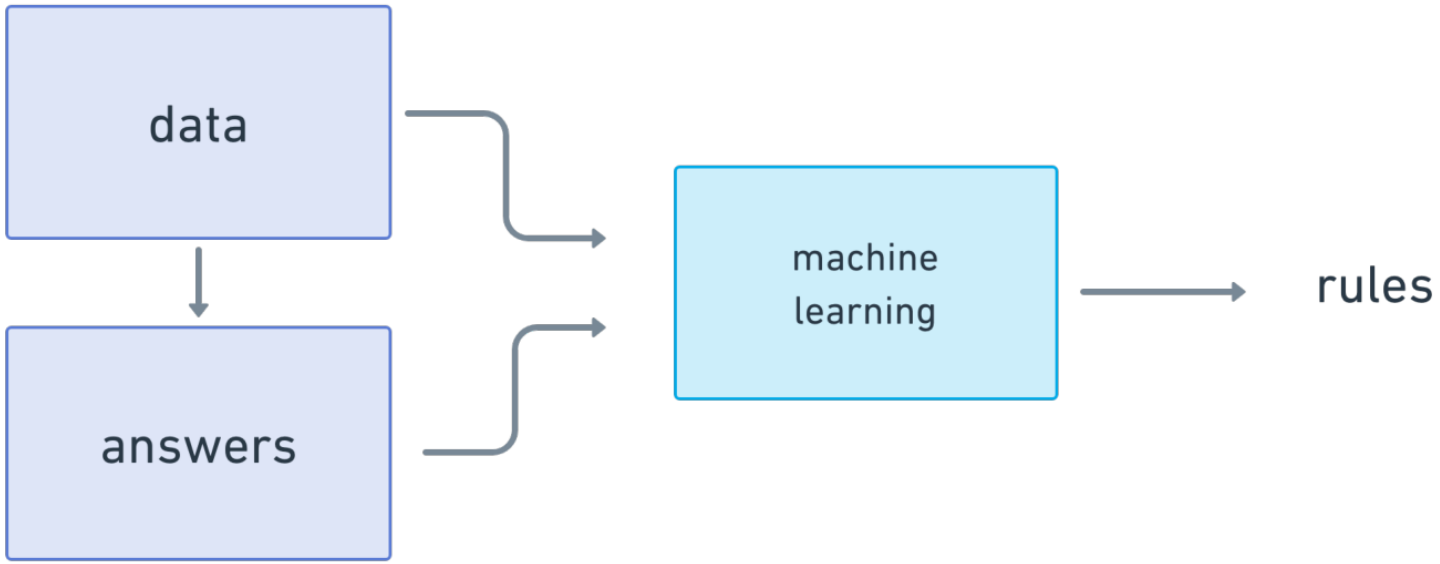
Learning from Data (Supervised)

- Learn rules automatically: machine learning (90's), neural methods (2010's)



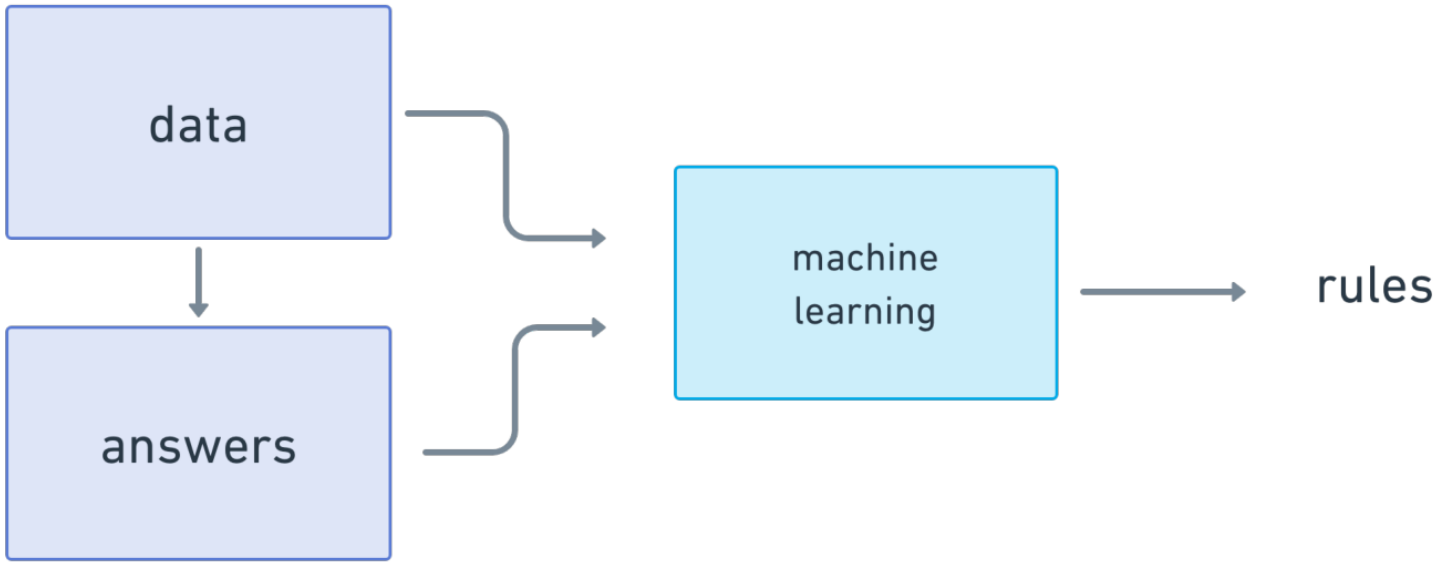
Learning from Data (Supervised)

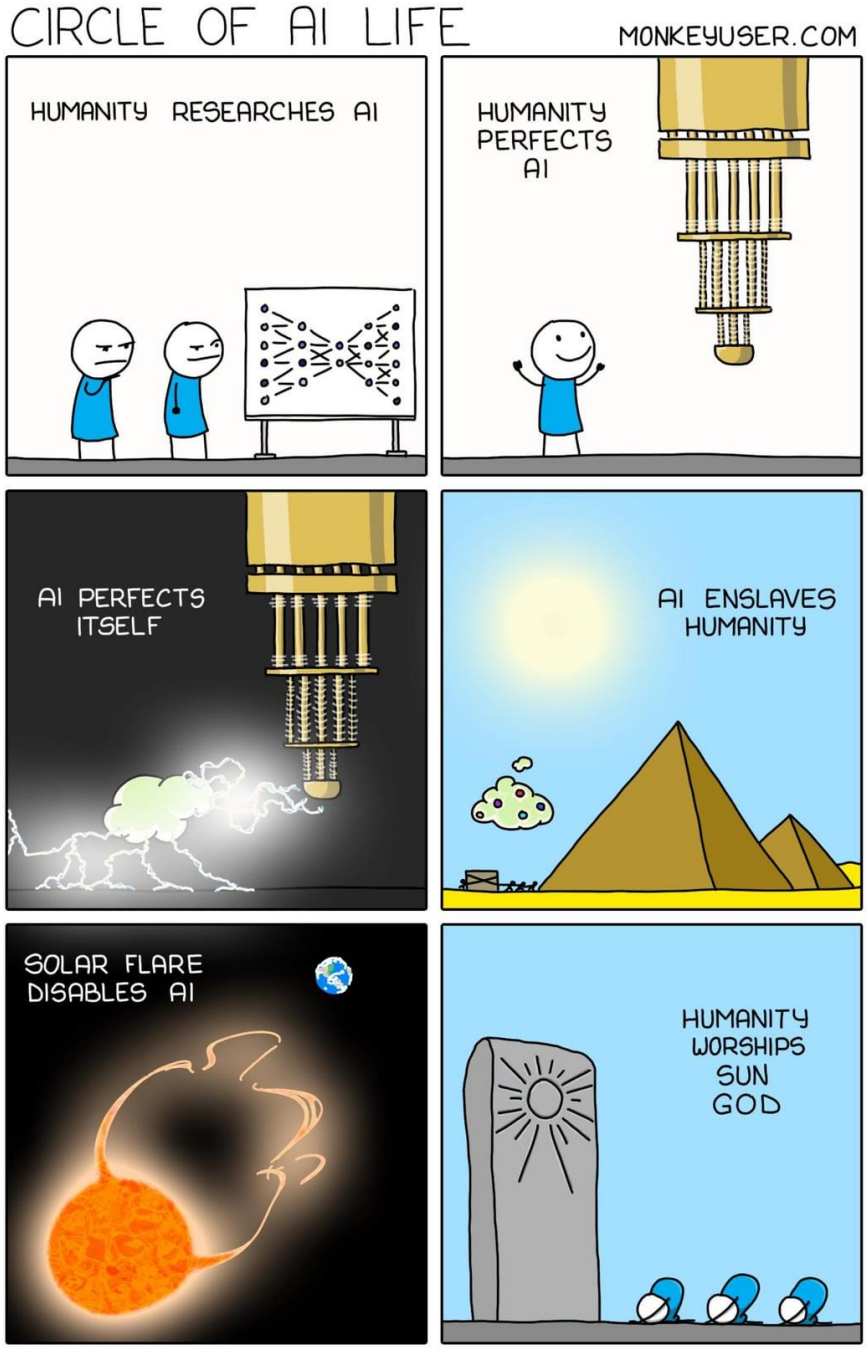
- Learn rules automatically: machine learning (90's), neural methods (2010's)
- Advantages: improved performance and generalization, fast



Learning from Data (Supervised)

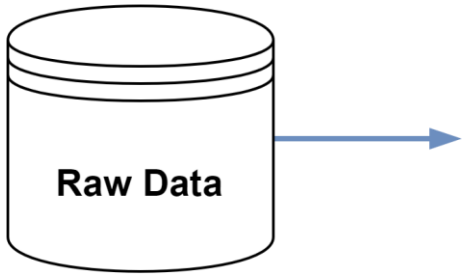
- Learn rules automatically: machine learning (90's), neural methods (2010's)
- Advantages: improved performance and generalization, fast
- Downsides: complex, hard to interpret



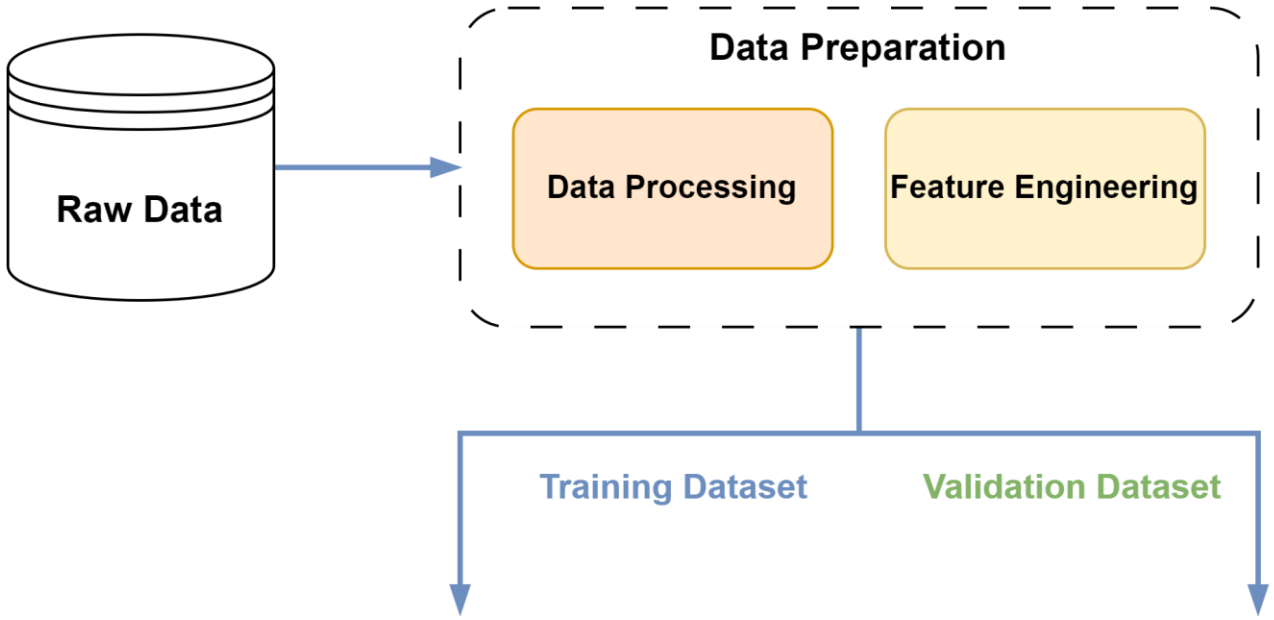


Questions so far ?

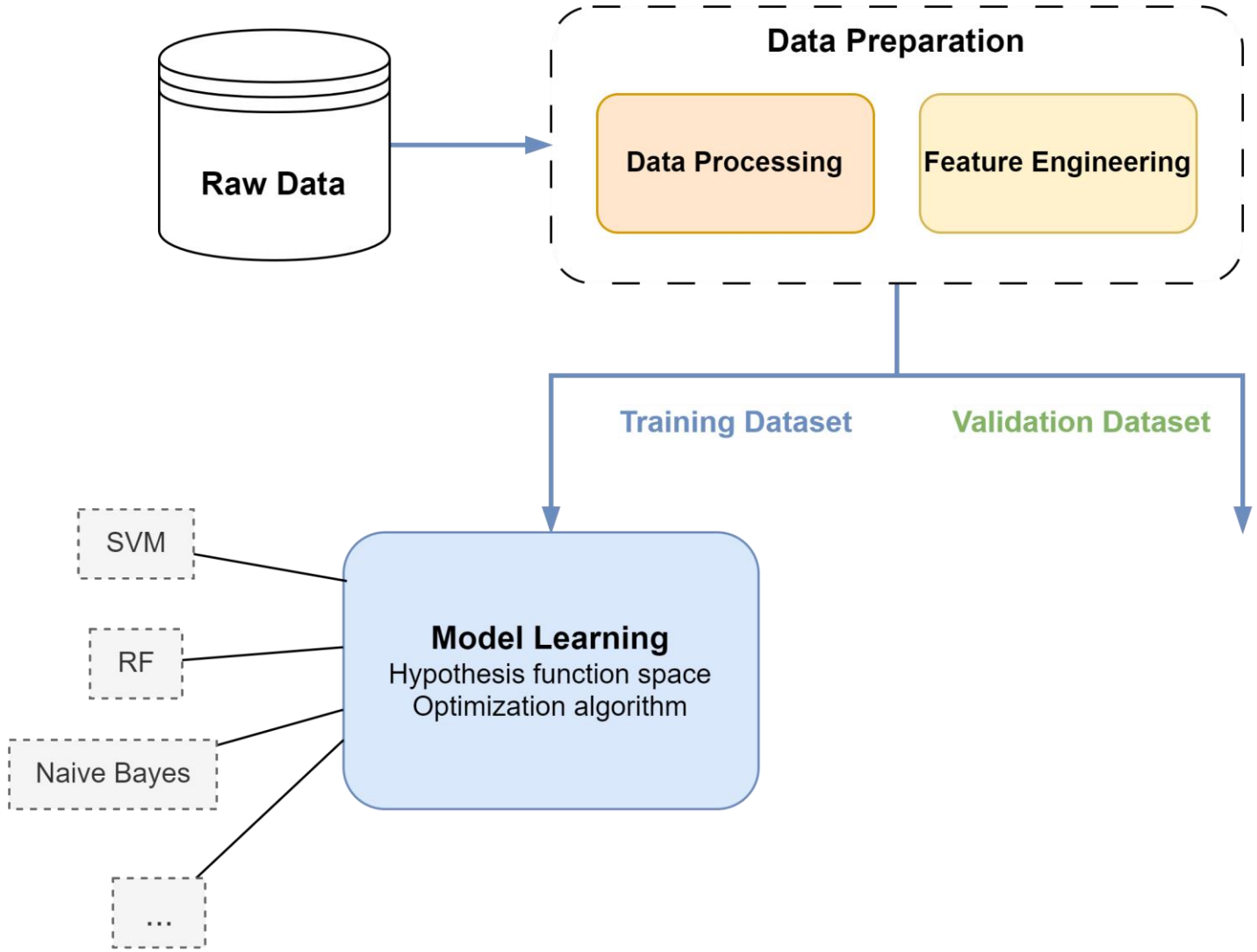
ML Workflow (Supervised)



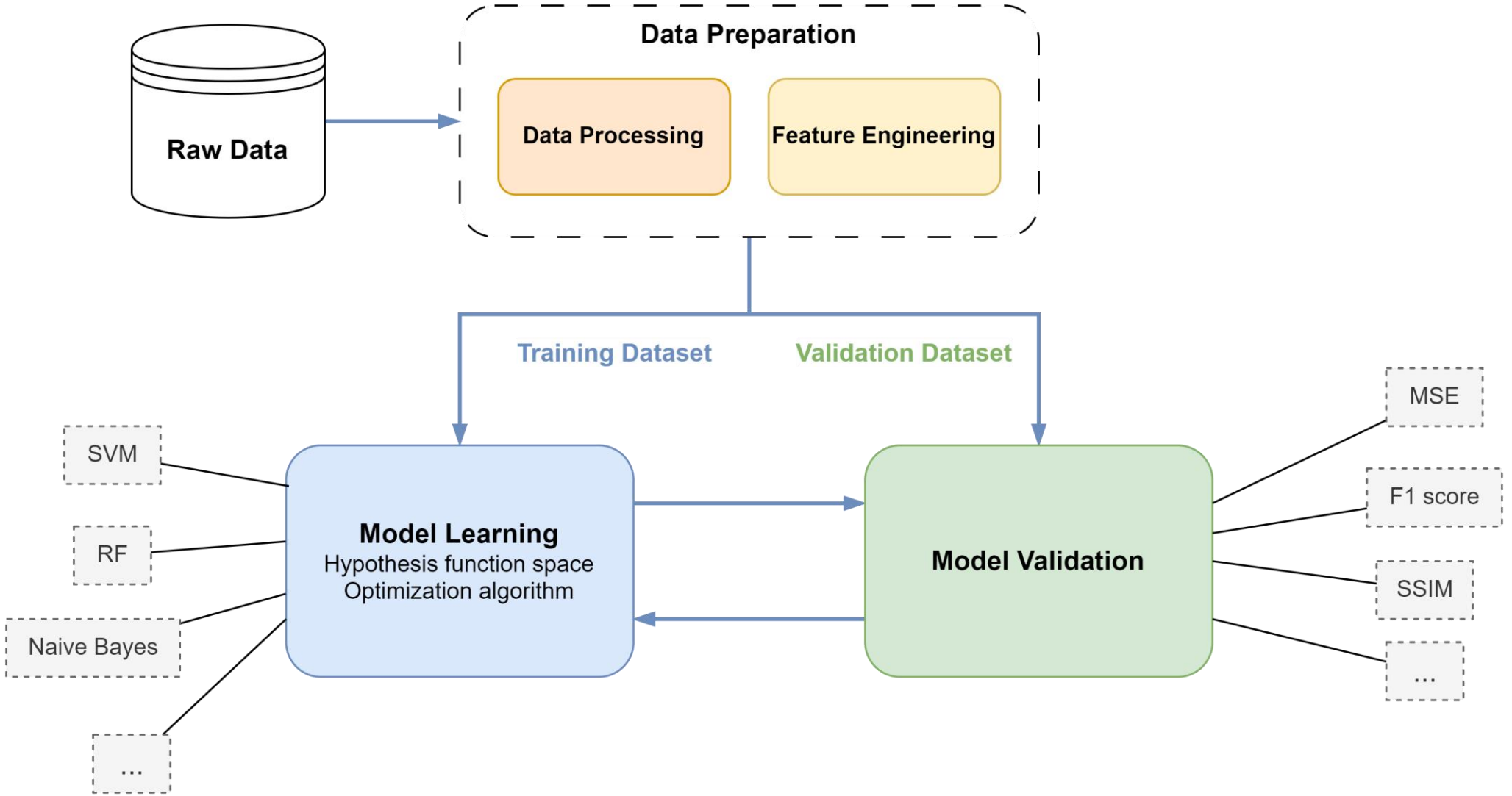
ML Workflow (Supervised)



ML Workflow (Supervised)

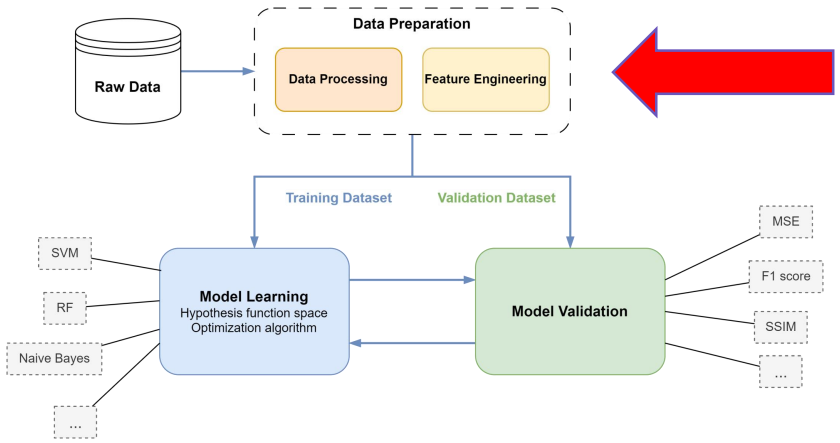


ML Workflow (Supervised)



ML Workflow for NLP

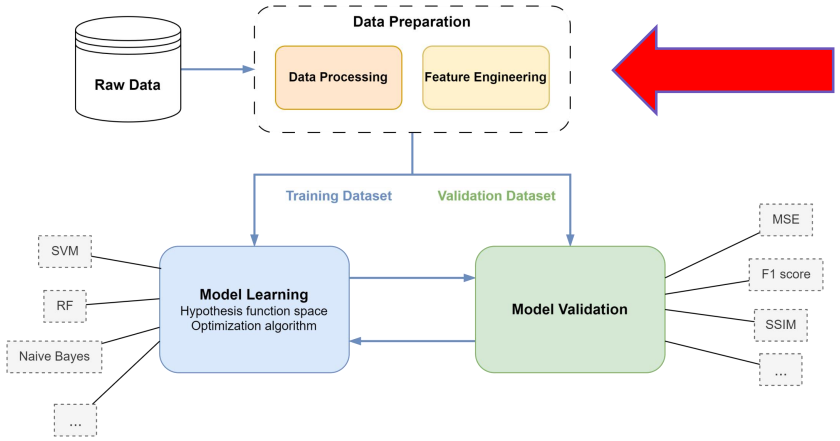
- How to prepare data for learning ?



ML Workflow for NLP

- How to prepare data for learning ?

- Models need for numerical features. How to go from sentences/words/tokens to vector representation?

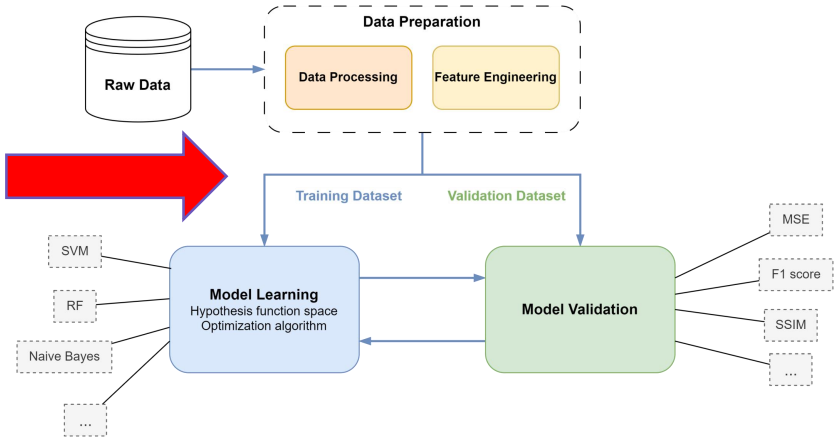


ML Workflow for NLP

- How to prepare data for learning ?

- Models need for numerical features. How to go from sentences/words/tokens to vector representation?

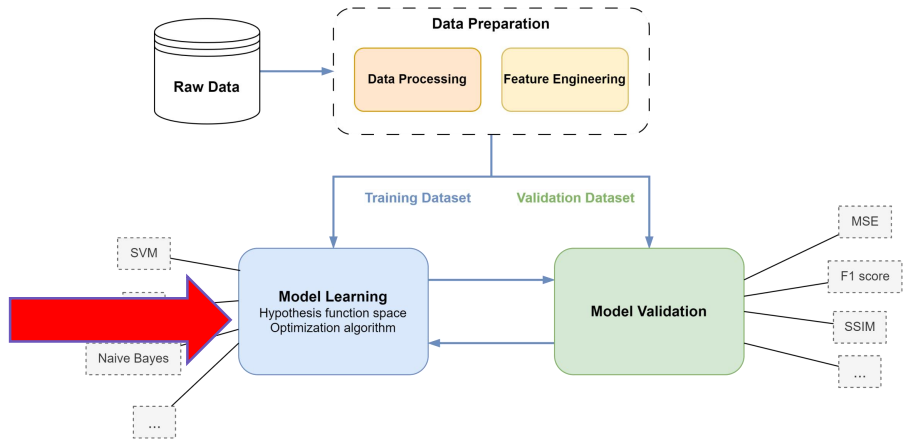
- How to split data into train/val/test subsets ?



FRENCH INSTITUTES OF TECHNOLOGY

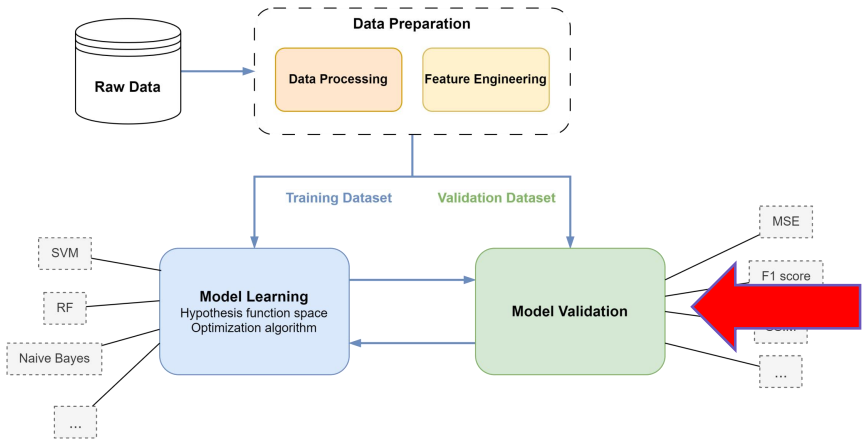
ML Workflow for NLP

- How to prepare data for learning ?
- Models need for numerical features. How to go from sentences/words/tokens to vector representation?
- How to split data into train/val/test subsets ?
- Are there suitable models for learning ?

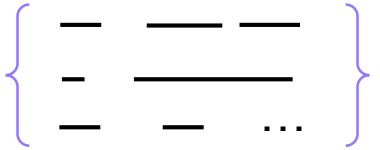


ML Workflow for NLP

- How to prepare data for learning ?
- Models need for numerical features. How to go from sentences/words/tokens to vector representation?
- How to split data into train/val/test subsets ?
- Are there suitable models for learning ?
- Metrics for evaluation ?



Some Terminology

Corpus	Document	Token	Vocabulary
		–	
Collection of documents	Collection of tokens	Collection of characters	Unique tokens

Corpus

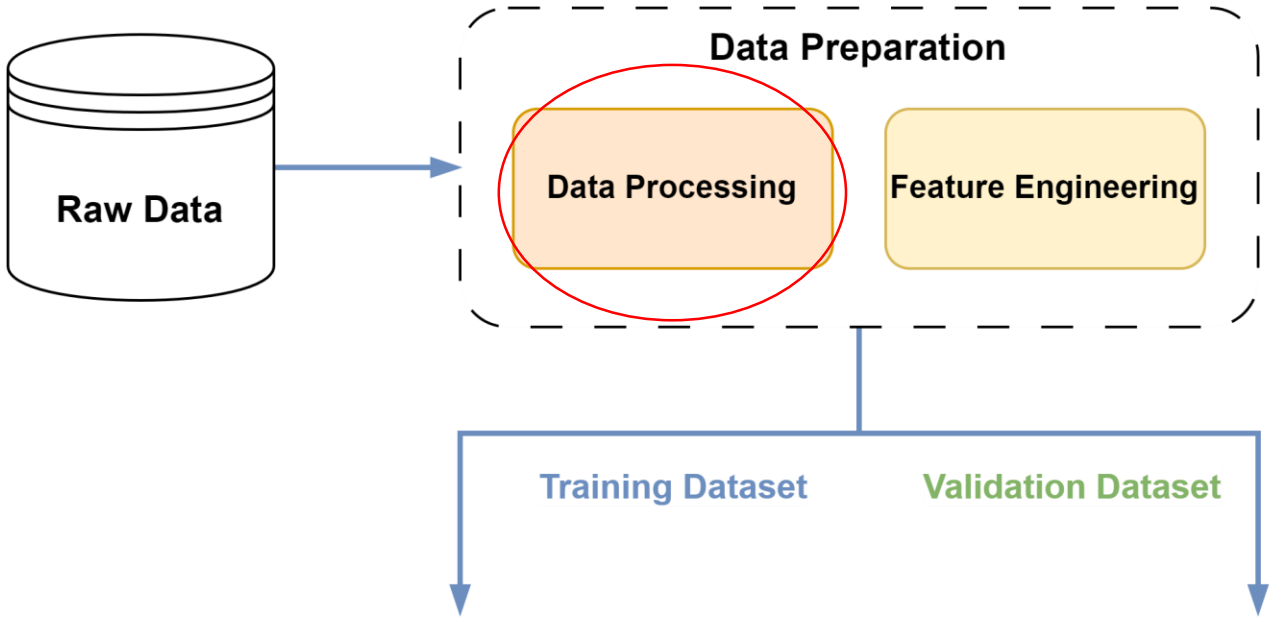
Document 1: "Cyber threats pose a risk to organizations."

Document 2: "Inadequate cyber defenses expose organizations to cyber risks."

Vocabulary (after lemmatization and sorting by alphabetical order)

[cyber, defense, expose, inadequate, organization, pose, risk, threats]

ML Workflow for NLP: Data Preprocessing

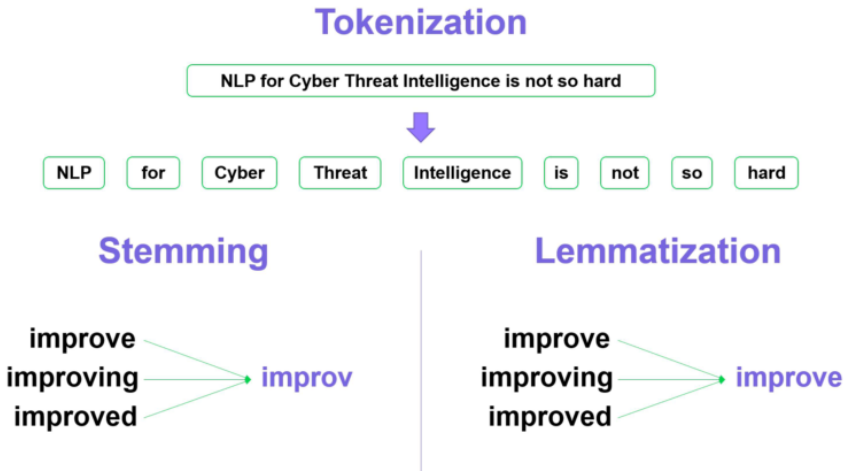


Preprocessing Methods or NLP

ML for NLP: From language to Vectors

Data Preparation

- Preprocessing & normlalization: lower case, special characters, stop words, stemming, tokenizing, …

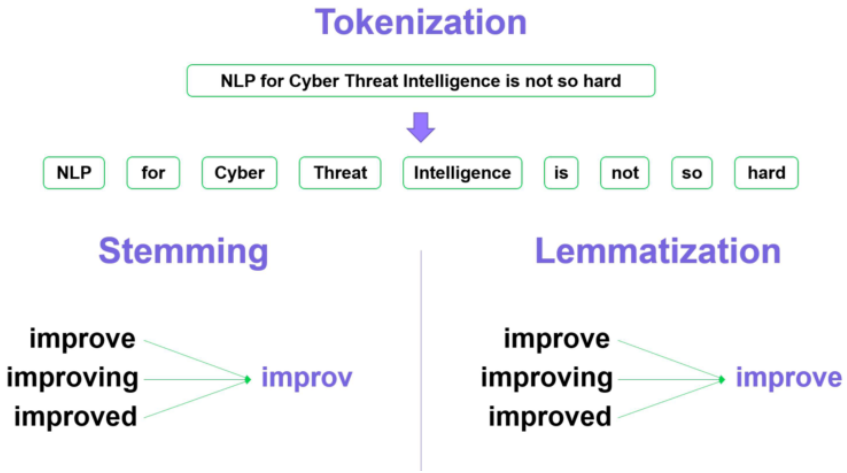


ML for NLP: From language to Vectors

Data Preparation

- Preprocessing & normlalization: lower case, special characters, stop words, stemming, tokenizing, …

- Numerical feature vectors: ML methods requires numerical features

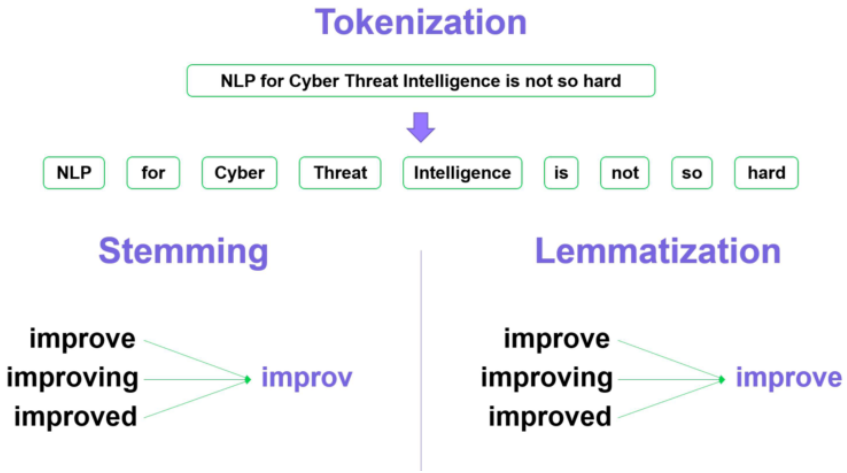


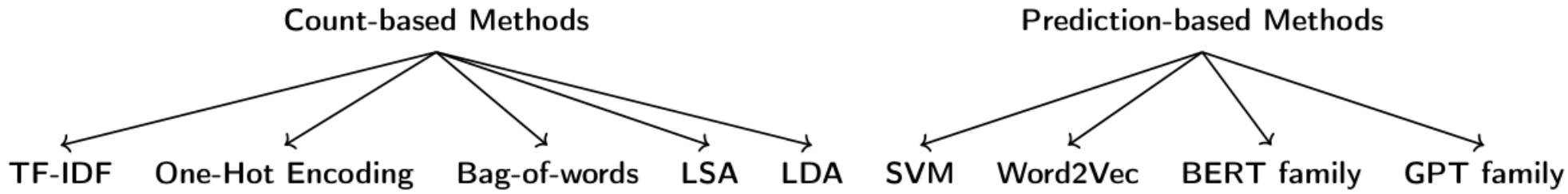
ML for NLP: From language to Vectors

Data Preparation

- Preprocessing & normlalization: lower case, special characters, stop words, stemming, tokenizing, …

- Numerical feature vectors: ML methods requires numerical features





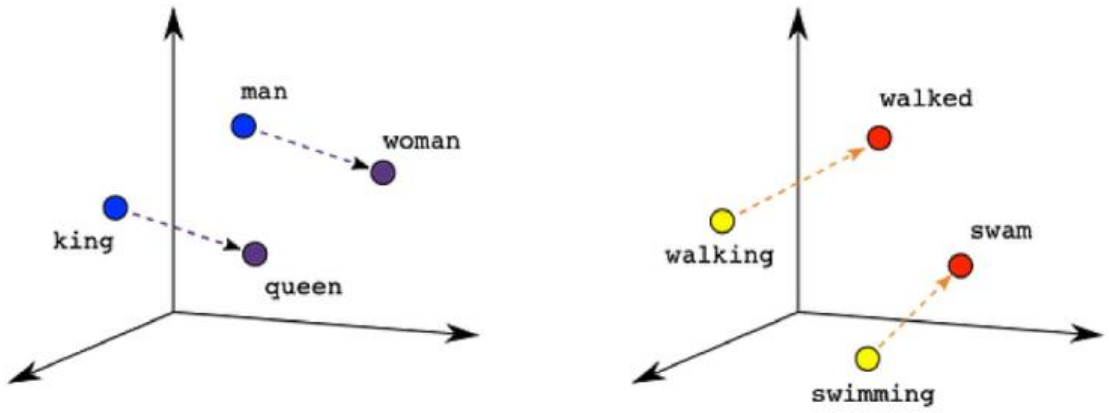
FRENCH INSTITUTES OF TECHNOLOGY

ML for NLP: From language to Vectors

Data Preparation

- Preprocessing & normlalization: lower case, special characters, stop words, stemming, tokenizing, …

- Numerical feature vectors: ML methods requires numerical features



Corpus

Document 1: "Cyber threats pose a risk to organizations."

Document 2: "Inadequate cyber defenses expose organizations to cyber risks."

Vocabulary (after lemmatization and sorting by alphabetical order)

[cyber, defense, expose, inadequate, organization, pose, risk, threats]

One-hot Encoding of Document 1 (with the Vocabulary)

Doc1\Voc	cyber	defense	expose	inadequate	organization	pose	risk	threats
Cyber	1	0	0	0	0	0	0	0
threats	0	0	0	0	0	0	0	1
pose	0	0	0	0	0	1	0	0
risk	0	0	0	0	0	0	1	0
organization	0	0	0	0	1	0	0	0

BoW (with the Vocabulary)

Doc\Voc	cyber	defense	expose	inadequate	organization	pose	risk	threats
Document 1	1	0	0	0	1	1	1	1
Document 2	2	1	1	1	1	0	1	1

TF-IDF of Document 2 (with the Corpus & Vocabulary)

	cyber	defense	expose	inadequate	organization	pose	risk	threats
# docs w/ the token	2	1	1	1	2	1	2	1
IDF	$\log(\frac{2}{2})$	$\log(\frac{2}{1})$	$\log(\frac{2}{1})$	$\log(\frac{2}{1})$	$\log(\frac{2}{2})$	$\log(\frac{2}{1})$	$\log(\frac{2}{2})$	$\log(\frac{2}{1})$
TF(*,Doc2)	$\frac{2}{7}$	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$
TF - IDF(*,Doc2)	0	0.043	0.043	0.043	0	0.043	0	0

fit FRENCH INSTITUTES OF TECHNOLOGY

Classical Preprocessing Approaches

Simpler and rule-based methods:

Focused on breaking text into smaller components (e.g., words or characters).
Relied heavily on word frequency and patterns in the text.

Key Characteristics:

Count-based techniques to measure how often words appear in text.
Statistical methods to identify relationships between words and their contexts.
Distributional Hypothesis: similarity in meaning results in similarity of linguistic distribution (Harris 1954)

Advantages:

- **Simplicity**: Easy to implement and interpret. + Works well for small datasets or straightforward tasks.
- **Computational Efficiency**: Requires less processing power compared to neural models.
- **Domain-Specific Customization**: Rules can be tailored to specific applications (e.g., legal or medical texts).
- **Explainability**: Outputs are often easier to understand and debug.

Text Cleaning

Objective: Enhance text data quality before vectorization.

Steps:

- **Removing unnecessary characters:** punctuation, emojis, special characters.
- **Normalization:**
  - Lowercasing text.
  - Removing accents.
  - Stemming, lemmatization
- **Stopword removal:** Removing non-informative words (e.g., "the," "and").

Text Cleaning

Objective: Enhance text data quality before vectorization.

Steps:

- **Removing unnecessary characters:** punctuation, emojis, special characters.
- **Normalization:**
  - Lowercasing text.
  - Removing accents.
  - Stemming, lemmatization
- **Stopword removal:** Removing non-informative words (e.g., "the," "and").

Note for recent deep learning models:

- Lighter cleaning to preserve context.
- No stopword removal or stemming.

Recommended steps:

- Remove unnecessary characters (punctuation, emojis) depending on the task.
- Simple normalization (lowercasing, accent removal).

Lemmatization, and Stemming

Stemming: Reduces a word to its root (e.g., "running" → "run").

Lemmatization: Converts a word to its base form, considering context (e.g., "was" → "be").

Comparison:

- **Stemming:** faster but less precise.
- **Lemmatization:** more accurate.

Tokenization

Whitespace Tokenization:
Splits text at spaces.

Example: "I can't play outside because it's raining."→ ["I", "can't", "play", "outside", "because", "it's", "raining."]

Limitations: Treats punctuation as part of tokens, can't handle contractions like "don't".

Tokenization

Word-based Tokenization:
Divides text into words while applying additional rules to handle punctuation, contractions, etc.

Example: "I can't play outside because it's raining." → ["I", "ca", "n't", "play", "outside", "because", "it", "'s", "raining", "."]

Limitations: Slower due to additional rules.

Tokenization

Character-based Tokenization:
Treats each character as a token.

Example: "can’t" → ["c", "a", "n", " ‘ ", "t"]

Use cases: Rarely used alone; mostly for simple or low-resource setups.

Tokenization

Tokenization

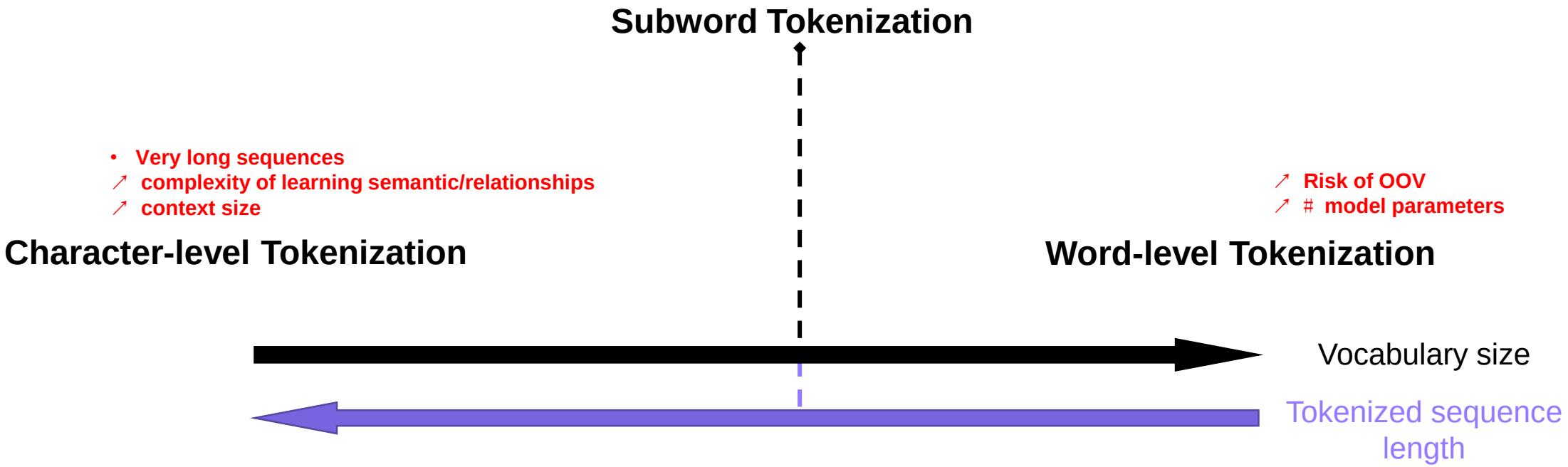
- Very long sequences
↗ complexity of learning semantic/relationships
↗ context size

Character-level Tokenization

↗ Risk of OOV
↗ # model parameters

Word-level Tokenization

Tokenization



Tokenization

Subword Tokenization:

Words are divided into smaller units called subwords.

- Common(to be formalized) occurring words should be in the vocabulary
- Rare(to be formalized) words should be split into frequent subwords

Example:

- Rare word: "unbelievable" → ["un", "believ", "able"]
- Common word: "cat" → ["cat"] (remains intact).

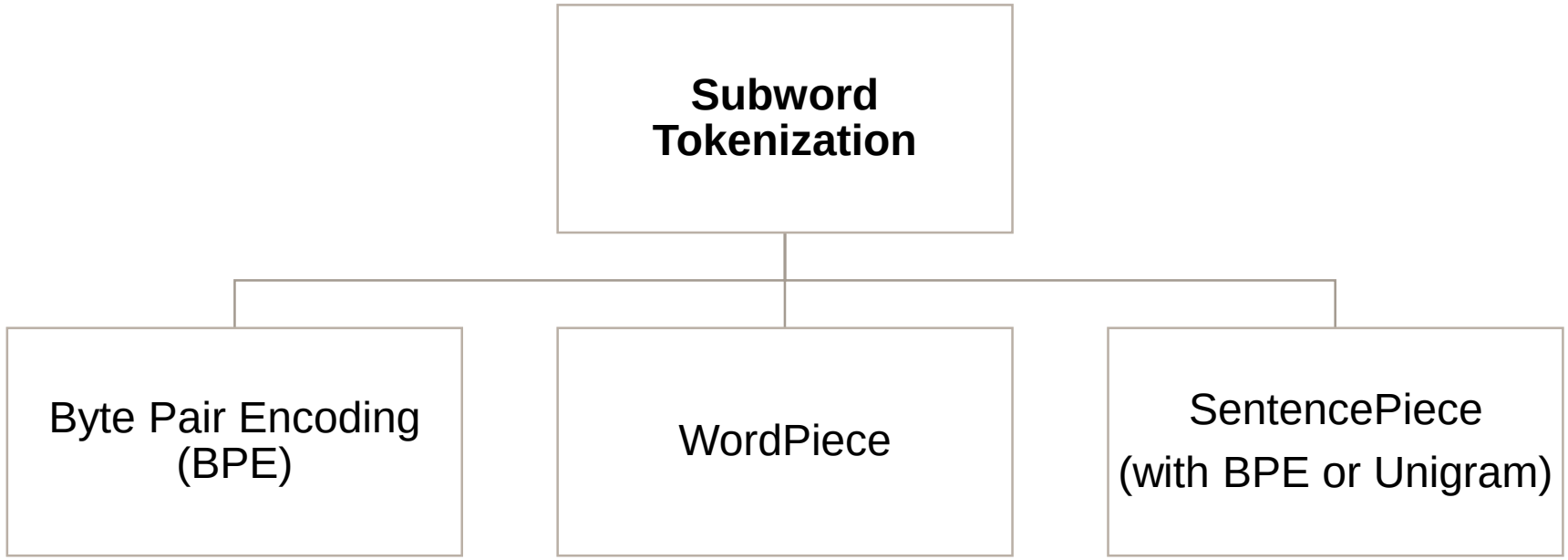
A fixed vocabulary (e.g., 30k-50k tokens) is created before model training.
Includes frequent subwords, characters, and special symbols (e.g., [CLS], [SEP]).

Tokenization

Subword Tokenization:

Words are divided into smaller units called subwords.

- Common(to be formalized) occurring words should be in the vocabulary
- Rare(to be formalized) words should be split into frequent subwords



Subword Tokenization Techniques

1) Byte Pair Encoding (BPE)
Compression algorithm adapted to NLP => build vocab by merge token pairs based on frequency
Used in models like GPT and GPT-2.

Procedure:

1. Initialization
   - Start with the training corpus (space splitted).
     Example: **[("car", 5), ("cable", 3), ("tablet", 1), ("watch", 2), ("chair", 5), ("mouse", 1)]**
   - Build initial token-level vocabulary **V = [‘a’, ‘b’, ‘c’, ‘e’, ‘h’, ‘i’, ‘l’, ‘m’, ‘o’, ‘r’, ‘s’ ,’t’, ‘u’, ‘w’]**

2. Segment based on the current vocabulary: **[(‘c’,’a’,’r’ , 5), (‘c’,’a’,’b’,’l’,’e’, 3), (‘t’,’a’,’b’,’l’,’e’,’t’, 1), (‘w’,’a’,’t’,’c’,’h’, 2), (‘c’,’h’,’a’,’i’,’r’, 5), (‘m’,’o’,’u’,’s’,’e’, 1)]**

3. Merge the most frequent pairs of tokens to form new tokens and update vocab.
   Example: “c” + “a” : **8 occ** ; “c”+ “h”: **7 occ** ; **…**
   - Update the vocab: **V ← [‘a’, ‘b’, ‘c’, ‘e’, ‘h’, ‘i’, ‘l’, ‘m’, ‘o’, ‘r’, ‘s’ ,’t’, ‘u’, ‘w’, *‘ca’*]** (size=15)

Subword Tokenization Techniques

1) Byte Pair Encoding (BPE)
Compression algorithm adapted to NLP => build vocab by merge token pairs based on frequency
Used in models like GPT and GPT-2.

Procedure:

4. Segment based on the current vocabulary: **[('ca','r' , 5), ('ca','b','l','e', 3), ('t','a','b','l','e','t', 1), ('w','a','t','c','h', 2), ('c','h','a','i','r', 5), ('m','o','u','s','e', 1)]**

5. Merge the most frequent pairs of tokens to form new tokens.
   Example: "c"+ "h": 7 occ ; **…**
   Update the vocab: **V ← ['a', 'b', 'c', 'e', 'h', 'i', 'l', 'm', 'o', 'r', 's' ,'t', 'u', 'w', *'ca', 'ch'*]** (size=16)

6. Continue until reaching a fixed vocabulary size (e.g., 50k tokens).

Subword Tokenization Techniques

1) Byte Pair Encoding (BPE)
Compression algorithm adapted to NLP => build vocab by merge token pairs based on frequency
Used in models like GPT and GPT-2.

Tokenization:

Longest matching token:

Vocabulary =[‘a’, ‘b’, ‘c’, ‘e’, ‘h’, ‘i’, ‘l’, ‘m’, ‘o’, ‘r’, ‘s’ ,’t’, ‘u’, ‘w’, *‘ca’, ‘ch’*]

Example: “cash” → [“ca”, “s”, “h”]

Advantages:

- Reduces vocabulary size while retaining coverage.

Subword Tokenization Techniques

2) WordPiece

Used in models like BERT.

- Similar to BEP
- Aim to maximize the likelihood of the training data

Procedure

1. Initialization:
   - Start with the training corpus (space splitted). Example: **[("car", 5), ("cable", 3), ("tablet", 1), ("watch", 2), ("chair", 5), ("mouse", 1)]**
   - Build initial token-level vocabulary. Use ## to indicate that a token continues a previous word.
     Example: [“car”] → [“c", “##a“, “##r“]

2. Merge pair of tokens into subwords based on **highest statistical likelihood score** (rather than frequency like BPE)
   Example: Score(“##a”, “##r”) = $\frac{number\ of\ pair\ ("##a," "##r")}{(number\ of\ \#\#a) \times (\text{number of } \#\#r)}$
3. Update vocabulary and repeat until reaching a fixed vocabulary size (e.g., 50k tokens).

Tokenization

Example: [“car”] → [“c", “##ar“]

Subword Tokenization Techniques

3) SentencePiece (Unigram)
Used in models like T5 and GPT-3.

How it works:

1. Processes raw text without requiring initial segmentation (no splitting by spaces).
   Example: “I can’t play.” → “I_can’t_play”
1. Tokenize a sentence using Unigram (minimizing the number of tokens while maximizing the overall probability) or BPE

Example:
“I_can’t_play” → ["▁I", "▁can", "’t", "▁play"] (▁ denotes the start of a new word.)

P("▁I")=0.3	**P(“’t")=0.1**
P("▁can’t")=0.04	**P(“_play")=0.12**
P("▁ca")=0.002	P(“_pl")=0.003
P("▁can")=0.5	P(“ay")=0.01

Advantages:
Flexible for different scripts or languages.

Subword Tokenization Techniques

	BPE	WordPiece	SentencePiece
Principle	Merges the most frequent subword pairs	Merges the most likely subword pairs	Probabilistic model based on a global vocabulary (Unigram)
Spaces	Prior segmentation required	Prior segmentation required	Treats spaces as characters
Indicators	No	Uses indicators (e.g., ##)	Prefix _ for new words
Advantages	Simple and fast	Better at capturing context	Flexible, suitable for any language
Disadvantages	Ignores global context	Less suitable for unsegmented corpora	More complex to train
Current use	GPT-2, RoBERTa	BERT, DistilBERT	T5, GPT-3

Padding, and Truncation

Input sequences must have the same length to form a consistent batch.
Models cannot handle sequences of varying lengths directly.

1. Padding

Adds special tokens ([PAD]) to shorter sequences to match the length of the longest sequence in a batch.
Attention masking to ignore padding tokens.

Example:
Sequence 1: ["I", "love", "studying", "transformers", "."]
Sequence 2: ["AI", "is", "fun"]
After Padding:
Sequence 1: ["I", "love", "studying", "transformers ", "."]
Sequence 2: ["AI", "is", "fun“, “[PAD]”, “[PAD]”]

Padding, and Truncation

Input sequences must have the same length to form a consistent batch.
Models cannot handle sequences of varying lengths directly.

2. Truncation

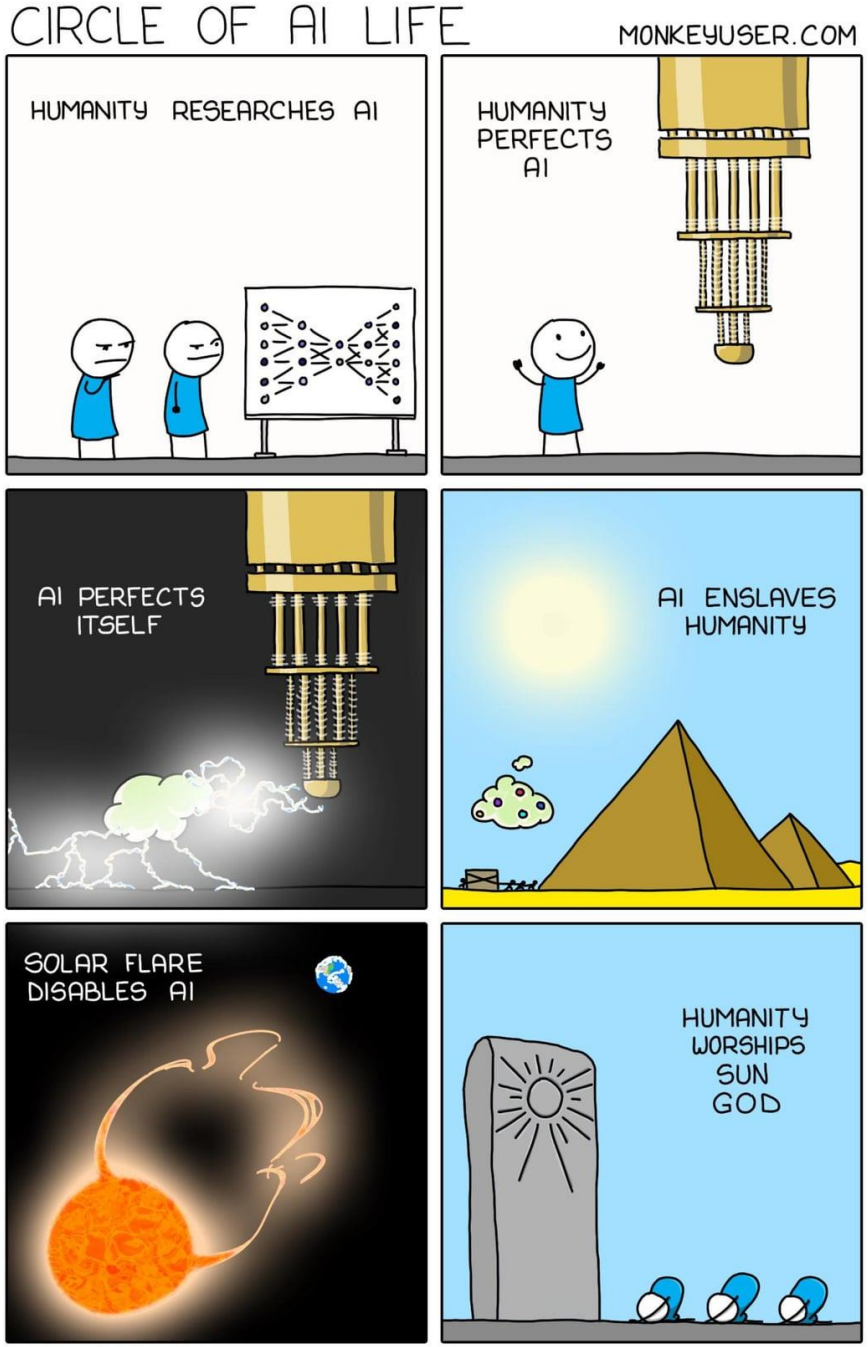
Cuts sequences that exceed a predefined maximum length.

Example:
Sequence 1: ["I", "love", "studying", "transformers", "."]
Sequence 2: ["AI", "is", "fun"]
Max Length: 4
After Truncation:
Sequence 1: ["I", "love", "studying", "transformers"]
Sequence 2: ["AI", "is", "fun"]

Padding, and Truncation

In practice, we use a mix of both techniques:

Example:
Sequence 1: ["I", "love", "studying", "transformers", "."]
Sequence 2: ["AI", "is", "fun"]
Max Length: 4
After Truncation & Padding:
Sequence 1: ["I", "love", "studying", "transformers"]
Sequence 2: ["AI", "is", "fun“, “[PAD]”]



Questions so far ?

Vectorization Techniques

Text Vectorization

What Is Text Vectorization?
The process of converting **text into numerical representations** so it can be used by machine learning models.

Key Idea: Words or documents are transformed into **vectors** (arrays of numbers) that encode their features.

Example:
Sentence Tokenized: [“I”, “love”, “NLP”, “and”, “I”, “love”, “AI”]

Bag of Words:
- Vocabulary: [“I”, “love”, “like”, “NLP”, “and”, “AI”, “ML”] (built from a corpus)
- Vector: [2,2,0,1,1,1,0]

Count-based Techniques

Bag of Words (BoW)

Bag of Words is a simple and widely-used text representation technique. It represents a **document** (such as a sentence) as a set of words (a "bag"), ignoring grammar, word order, and context.

Key Features

1. **Vocabulary Creation**: Extracts all unique words from the text corpus to form a vocabulary.
2. **Vector Representation**: Each document is represented as a vector where:
    - Each dimension corresponds to a word in the vocabulary.
    - Values indicate word frequency.

Example

Corpus: [
Doc 1:"I like NLP and NLP tools",
Doc 2: "I love NLP and AI“
]

Vocabulary: [I, like, NLP, love, and, tools, AI]

Vectors: Doc 1: [1, 1, 2, 0, 1, 1, 0]
Doc 2: [1, 0, 1, 1, 1, 0, 1]

Term Frequency - Inverse Document Frequency (TF-IDF)

TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a corpus (collection of documents).

Key Features

1. **Term Frequency (TF):** Measures how often a term appears in a document.
2. **Inverse Document Frequency (IDF)**: Reduces the weight of common words appearing in many documents.
3. **TF-IDF Weight**: Final weight of a term in a document is the product

$$TF(t, d) = \frac{\text{Number of times term t appears in document d}}{\text{Total number of terms in document d}}$$

$$IDF(t, D) = \log \frac{\text{Total number of documents in corpus D}}{\text{Number of documents containing term t}}$$

$$TF - IDF(t, d, D) = TF(t, d) \times IDF(t, D)$$

Example
Corpus: ["I like NLP", "I love NLP and AI"]

TF ('NLP', Doc 1): 1/3
IDF ('NLP'): log(2/2) = log(1) = 0
TF-IDF ('NLP', Doc 1): 1/3×0=0

N-Grams

n-Grams are contiguous sequences of n words (or characters) from a given text.
An n-gram model captures word sequences and their probabilities, providing insight into the structure and context of text.

Key Features

1. What is n? n=1: Unigram (single words).
   n=2: Bigram (pairs of words).
   n=3: Trigram (triplets of words).
   Larger n: Captures longer context but increases sparsity.
2. **How it works:** Extract all n-word sequences from text.

Example

Text: "I love NLP models."
Bigrams: ["I love", "love NLP", "NLP models"].

Vectors: …

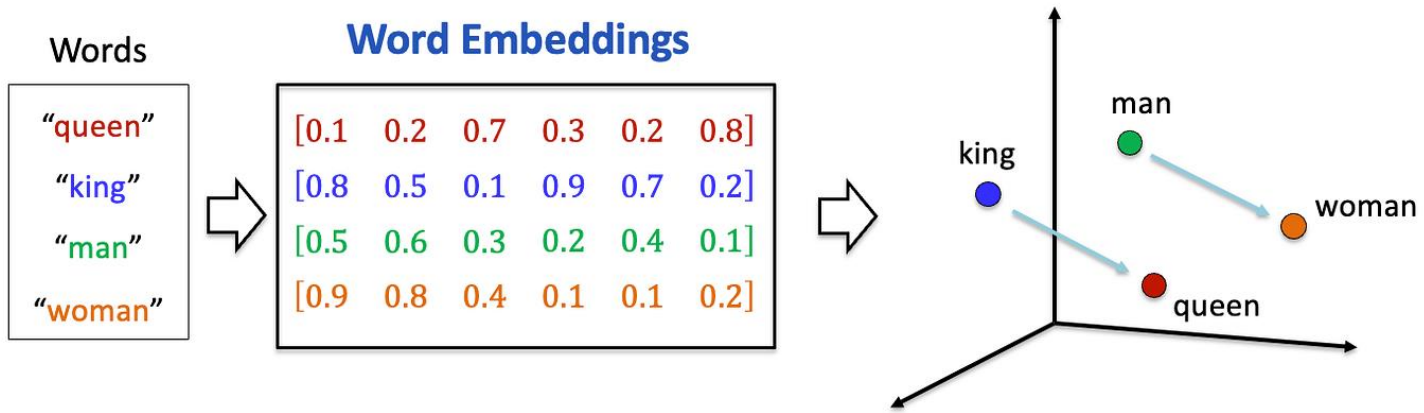
Comparaison

	BoW	TF-IDF	N-Grams
Advantages	• Easy to understand and implement. • Effective for small datasets or simple models.	• Highlights **important terms** specific to a document. • Reduces the impact of **common words** like "the" or "is."	• Adds **context** compared to unigrams. • Flexible for text analysis tasks.
Limitations	• **No context**: Ignores word order and semantics. • **High dimensionality**: Vocabulary size grows with corpus size. • **Sparse representation**: Many zeros in the vector.	• **No context**: Similar to Bag of Words, it ignores word order and semantics. • **High dimensionality**: Large vocabularies result in sparse vectors.	• **Dimensionality grows**: More unique n-grams with higher n. • **Data sparsity**: Larger n values need more data for meaningful results. • **No long-range dependencies**: Limited context beyond n.

Word Embeddings

Embeddings

Word embeddings are a way of representing words as **dense vectors in a multi-dimensional space**, where the **distance** and direction between vectors reflect the **similarity** and **relationships** among the corresponding words.



Embeddings

General Procedure:

- Tokenization and mapping to IDs:

Tokens are converted into numeric IDs based on the vocabulary.

Example: ["un", "believ", "able"] → [1234, 5678, 910].

- Vectorization with embedding model:

These IDs are transformed into dense vectors via an embedding layer before being processed by the LLM.

Example: [1234, 5678, 910] → [[0.2,0.45,1.23,1.01], [2.54,0.03,0.42,2.3], [0.11,0.4,0.1,2.54]]

Examples of embedding models:

- Word2Vec
- GloVe
- SBERT

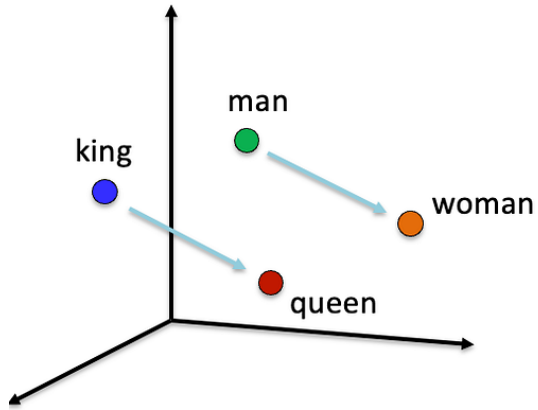
Word2Vec

Word2Vec is a neural network-based model that learns vector representations (embeddings) of words from a text corpus. It maps words to a continuous vector space where similar words are closer together based on their context.

Two Training approaches: Continuous Bag of Words (CBOW) or Skip-gram.

Word Representations:

- Each word is represented as a fixed-size vector (e.g., 100-300 dimensions).
- Captures **semantic similarity** and **relationships** (e.g., "king" - "man" + "woman" ≈ "queen").



Continuous Bag of Words (CBOW)

CBOW is a training architecture in Word2Vec that predicts a target word based on the context words surrounding it. It uses a shallow neural network to generate word embeddings.

Key Features

- **Input Representation**: One-hot encoding of context words.
- **Hidden Layer**: Projects words into a lower-dimensional vector space.
- **Output Layer**: Predicts the target word using a softmax function.

How CBOW Works

Input:

Context words surrounding a target word.

Example: "I __ NLP models" → Input = ["I", "NLP"]

Output:

Predicts the **missing word** ("love" in this case).

Training:

- The model learns embeddings by maximizing the probability of the target word given its context.
- Objective function: $\max \prod_{w \in \text{Vocabulary}} P(w|\text{context})$

Skip-gram

Skip-gram is a training architecture in Word2Vec that predicts context words given a target word.
It uses a shallow neural network to generate word embeddings.

Key Features

- **Input Representation**: One-hot encoding of the target word.
- **Hidden Layer**: Projects the target word into a lower-dimensional vector space.
- **Output Layer**: Predicts multiple context words using a softmax function.

How Skip-gram Works

Input:

A single target word.

Example: Input = "love."

Output:

Predicts surrounding **context words** (e.g., "I" and "NLP").

Training:

- The model learns embeddings by maximizing the probability of context words given the target word.
- Objective function: $\max \prod_{c \in \text{Context}} P(c|\text{target})$

Word2Vec – Architecture Training Choice

Criteria	CBOW	Skip-gram
Objective	Predict a target word from its surrounding context words.	Predict context words from a target word.
Efficiency	Faster to train because it aggregates multiple contexts.	Slower as it generates multiple predictions per target word.
Performance on Rare Words	Less effective, as it gives more importance to frequent words.	Better at learning representations for rare words.
Data Size Requirement	Requires a larger corpus for good performance.	Works well even with a smaller corpus.

GloVe (Global Vectors)

GloVe is a word embedding model that learns vector representations by analyzing the global word **co-occurrence** statistics from a text corpus.
Unlike Word2Vec, GloVe focuses on both local and global context for richer semantic embeddings.

Word Representations:

- Each word is represented as a fixed-size vector (e.g., 100-300 dimensions).
- Captures **semantic similarity** and **relationships** (e.g., "king" - "man" + "woman" ≈ "queen").

Global Context:

Captures word meaning by considering overall word relationships across the corpus.

How GloVe Works ?

Word Co-occurrence Matrix:

Builds a matrix where each cell represents how often two words co-occur in the corpus.

Optimization Objective:

Learns word embeddings w_i and w_j by factorizing the co-occurrence matrix into lower-dimensional vectors.

Objective function minimizes the reconstruction error:

$$J = \sum_{i,j} f(X_{ij})(\vec{w}_i^\top \vec{w}_j + b_i + b_j - \log(X_{ij}))^2$$

$$X = \begin{array}{c} \\ I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{array} \begin{array}{c} \begin{array}{cccccccc} I & like & enjoy & deep & learning & NLP & flying & . \end{array} \\ \begin{bmatrix} 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Example of co-occurrence matrix

X_{ij}: Co-occurrence count between words i and j.

$f(X_{ij})$: Weighting function to control the influence of rare vs. frequent pairs.

Contextual Representations

Contextual Techniques

Overview:

Embeddings based on Transformers (e.g. BERT).
Each word’s representation depends on its context.

Pretrained Models:

Use large corpora for training, enabling rich contextual understanding.

Comparison Example:

Sentence: "The bank is on the river."
Word2Vec: One vector for "bank."
BERT: Different vectors for "bank" depending on context.

How It Works?

Covered in CM5 Transformers

Questions ?

Exercices

In blue: statements

In black: correction

Exercice

Create the vocabulary for the corpus "banana bandanaer banner" **using Byte Pair Encoding (BPE) approach. Target vocab size is 10.**

Exercice

Create the vocabulary for the corpus "banana bandanaer banner" **using Byte Pair Encoding (BPE) approach. Target vocab size is 10.**

Step1: Initialize Tokenization

banana -> b a n a n a
bandanaer -> b a n d a n a e r
banner -> b a n n e r

Vocabulary = [‘b’,’a’,’n’,’d’,’r’,’e’]
Size = 6

Step2: Count Pair Frequencies

(b, a) -> 3
(a, n) -> 5
(n, a) -> 3
(n, d) -> 1
(d, a) -> 1
(a,e) -> 1
(e, r) -> 2
(n, n) -> 1
(n, e) -> 1

Exercice

Create the vocabulary for the corpus "banana bandanaer banner" **using Byte Pair Encoding (BPE) approach. Target vocab size is 10.**

Step3: Merge the Most Frequent Pair

banana -> b an an a
bandanaer -> b an d an a e r
banner -> b an n e r

Vocabulary = ['b','a','n','d','r','e','an']
Size = 7

Step4: Count Pair Frequencies

(b, an) -> 3
(an, an) -> 1
(an, a) -> 2
(an, d) -> 1
(d, an) -> 1
(a, e) -> 1
(e, r) -> 2
(an, n) -> 1
(n, e) -> 1

Exercice

Create the vocabulary for the corpus "banana bandanaer banner" **using Byte Pair Encoding (BPE) approach. Target vocab size is 10.**

Step 5: Merge the Most Frequent Pair

banana -> ban an a
bandanaer -> ban d an a e r
banner -> ban n e r

Vocabulary = [‘b’,’a’,’n’,’d’,’r’,’e’,’an’,‘ban’]
Size = 8

Step6: Count New Pair Frequencies

(ban, an) -> 1
(an, a) -> 2
(ban, d) -> 1
(d, an) -> 1
(a, e) -> 1
(e,r) -> 2
(ban, n) -> 1
(n,e) -> 1

Exercice

Create the vocabulary for the corpus "banana bandanaer banner" **using Byte Pair Encoding (BPE) approach. Target vocab size is 10.**

Step 7: Merge the Most Frequent Pair

banana -> ban ana
bandanaer -> ban d ana e r
banner -> ban n e r

Vocabulary = [‘b’,’a’,’n’,’d’,’r’,’e’,’an’,’ban’,‘ana’]
Size = 9

Step8: Count New Pair Frequencies

(ban, ana) -> 1
(ban, d) -> 1
(d, ana) -> 1
(ana, e) -> 1
(e,r) -> 2
(ban, n) -> 1
(n,e) -> 1

Exercice

Create the vocabulary for the corpus "banana bandanaer banner" **using Byte Pair Encoding (BPE) approach. Target vocab size is 10.**

Step 9: Merge the Most Frequent Pair → **Step10: Vocabulary size 10** ✅

banana -> ban ana
bandanaer -> ban d ana er
banner -> ban n er

Vocabulary = [‘b’,’a’,’n’,’d’,’r’,’e’,’an’,’ban’,‘ana’,‘er’]
Size = 10

Exercice – Text Cleaning

- **Remove punctuation and emoji.**
- **Lowercase**
- **Use stopwords: [“a”, “in”, “the”, “to”]**
- **Use Stemming**

Corpus

Document 1:
"A dog runs in the park every morning."

Document 2:
"Every dog loves to run."

Document 3:
"I have 3 very nice dogs, I love them. 😀"

Exercice – Text Cleaning

- **Remove punctuation and emoji.**
- **Lowercase**
- **Use stopwords: [“a”, “in”, “the”, “to”]**
- **Use Stemming**

Cleaned corpus

Document 1:
"dog run park every morning"

Document 2:
“every dog love run"

Document 3:
“i have 3 very nice dog i love them”

Exercice – Tokenization

Tokenize documents with Whitespace Tokenization

Cleaned corpus

Document 1:
"dog run park every morning"

Document 2:
“every dog love run"

Document 3:
“i have 3 very nice dog i love them”

Exercice – Tokenization

Tokenize documents with Whitespace Tokenization

Tokenized corpus

Document 1:
["dog”, “run”, “park”, “every”, “morning”]

Document 2:
[“every”, “dog”, “love”, “run”]

Document 3:
[“i”, “have”, “3”, “very”, “nice”, “dog”, “i”, “love”, “them”]

Exercice – Bag of Words (BoW)

Create Bag of Words representation for each document

Tokenized corpus

Document 1:
["dog", “run”, “park”, “every”, “morning”]

Document 2:
[“every”, “dog”, “love”, “run”]

Document 3:
[“i”, “have”, “3”, “very”, “nice”, “dog”, “i”, “love”, “them”]

Exercice – Bag of Words (BoW)

Create Bag of Words representation for each document

Vocabulary: ["dog”, “run”, “park”, “every”, “morning”, “love”, “i”, “have”, “3”, “very”, “nice”, “them”]

Vectorized documents:

Document 1:
[1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]

Document 2:
[1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0]

Document 3:
[1, 0, 0, 0, 0, 1, 1, 2, 1, 1, 1, 1]

Exercice – TF-IDF

Calculate de TF-IDF value for “dog” and “run” for each document.

Tokenized documents:

Document 1:
["dog”, “run”, “park”, “every”, “morning”]

Document 2:
[“every”, “dog”, “love”, “run”]

Document 3:
[“i”, “have”, “3”, “very”, “nice”, “dog”, “i”, “love”, “them”]

$$\mathrm{TF}(t) = \frac{\text{Count of term } t \text{ in the document}}{\text{Total terms in the document}}$$

$$\mathrm{IDF}(t) = \log\left(\frac{\text{Number of documents}}{\text{Number of documents containing } t}\right)$$

$$\text{TF-IDF}(t) = \mathrm{TF}(t) \times \mathrm{IDF}(t)$$

Exercice – TF-IDF

Calculate de TF-IDF value for “dog” and “run” for each document.

Tokenized documents

$$\mathrm{TF}(t) = \frac{\text{Count of term } t \text{ in the document}}{\text{Total terms in the document}}$$

Document 1:
["dog”, “run”, “park”, “every”, “morning”] TF(dog, Doc1) = 1/5

Document 2:
[“every”, “dog”, “love”, “run”] TF(dog, Doc2) = 1/4

Document 3:
[“i”, “have”, “3”, “very”, “nice”, “dog”, “i”, “love”, “them”] TF(dog, Doc3) = 1/9

Exercice – TF-IDF

Calculate de TF-IDF value for “dog” and “run” for each document.

Tokenized documents

Document 1:
["dog”, “run”, “park”, “every”, “morning”]

Document 2:
[“every”, “dog”, “love”, “run”]

Document 3:
[“i”, “have”, “3”, “very”, “nice”, “dog”, “i”, “love”, “them”]

$$\mathrm{IDF}(t) = \log\left(\frac{\text{Number of documents}}{\text{Number of documents containing } t}\right)$$

IDF(dog) = log(3/3) = 0

Exercice – TF-IDF

Calculate de TF-IDF value for “dog” and “run” for each document.

Tokenized documents

$$\text{TF-IDF}(t) = \text{TF}(t) \times \text{IDF}(t)$$

Document 1:
["dog”, “run”, “park”, “every”, “morning”]

TF-IDF(dog, Doc1) = 1/5 x 0 = 0

Document 2:
[“every”, “dog”, “love”, “run”]

TF-IDF(dog, Doc1) = 1/4 x 0 = 0

Document 3:
[“i”, “have”, “3”, “very”, “nice”, “dog”, “i”, “love”, “them”]

TF-IDF(dog, Doc1) = 1/9 x 0 = 0

Exercice – TF-IDF

Calculate de TF-IDF value for “dog” and “run” for each document.

Tokenized documents

$$\mathrm{TF}(t) = \frac{\text{Count of term } t \text{ in the document}}{\text{Total terms in the document}}$$

Document 1:
["dog”, “run”, “park”, “every”, “morning”] TF(run , Doc1) = 1/5

Document 2:
[“every”, “dog”, “love”, “run”] TF(run , Doc2) = 1/4

Document 3:
[“i”, “have”, “3”, “very”, “nice”, “dog”, “i”, “love”, “them”] TF(run , Doc3) = 0/9

Exercice – TF-IDF

Calculate de TF-IDF value for “dog” and “run” for each document.

Tokenized documents

Document 1:
["dog”, “run”, “park”, “every”, “morning”]

Document 2:
[“every”, “dog”, “love”, “run”]

Document 3:
[“i”, “have”, “3”, “very”, “nice”, “dog”, “i”, “love”, “them”]

$$\mathrm{IDF}(t) = \log\left(\frac{\text{Number of documents}}{\text{Number of documents containing } t}\right)$$

IDF(run) = log(3/2) = 0.176

Exercice – TF-IDF

Calculate de TF-IDF value for “dog” and “run” for each document.

Tokenized documents

$$\text{TF-IDF}(t) = \text{TF}(t) \times \text{IDF}(t)$$

Document 1:
["dog”, “run”, “park”, “every”, “morning”]

TF-IDF(run, Doc1) = 1/5 x 0.176 = 0.0352

Document 2:
[“every”, “dog”, “love”, “run”]

TF-IDF(run, Doc2) = 1/4 x 0.176 = 0.44

Document 3:
[“i”, “have”, “3”, “very”, “nice”, “dog”, “i”, “love”, “them”]

TF-IDF(run, Doc3) = 0/9 x 0.176 = 0

Exercice – N-Grams

Create N-grams representation for each document with n = 2

Cleaned corpus

Document 1:
"dog run park every morning"

Document 2:
“every dog love run"

Document 3:
“i have 3 very nice dog i love them”

Exercice – N-Grams

Create N-grams representation for each document with n = 2

Step 1: Tokenization

Tokenized documents

Document 1:
["dog run”, “run park”, “park every”, “every morning”]

Document 2:
[“every dog”, “dog love”, “love run”]

Document 3:
[“i have”, “have 3”, “3 very”, “very nice”, “nice dog”, “dog i”, “i love”, “love them”]

Exercice – N-Grams

Create N-grams representation for each document with n = 2

Step 2: Vectorization

Vocabulary: ["dog run”, “run park”, “park every”, “every morning”, “every dog”, “dog love”, “love run”, “i have”, “have 3”, “3 very”, “very nice”, “nice dog”, “dog i”, “i love”, “love them”]

Vectorized documents

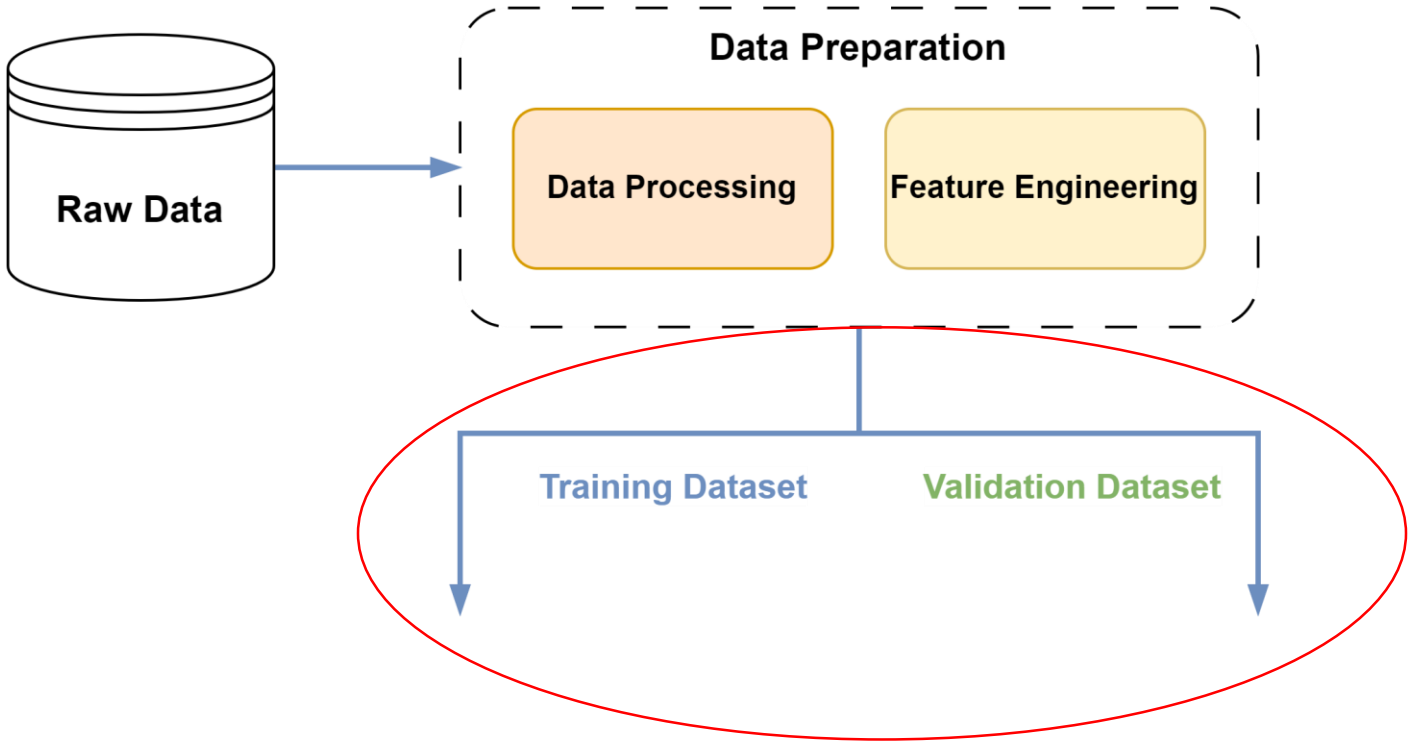
Document 1:
[1,1,1,1,0,0,0,0,0,0,0,0,0,0,0]

Document 2:
[0,0,0,0,1,1,1,0,0,0,0,0,0,0,0]

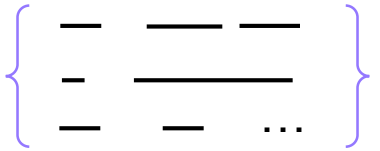
Document 3:
[0,0,0,0,0,0,0,1,1,1,1,1,1,1,1]

Questions ?

ML Workflow (Supervised)



Reminder

Corpus	Document	Token	Vocabulary
			
Collection of documents	Collection of tokens	Collection of characters	Unique tokens

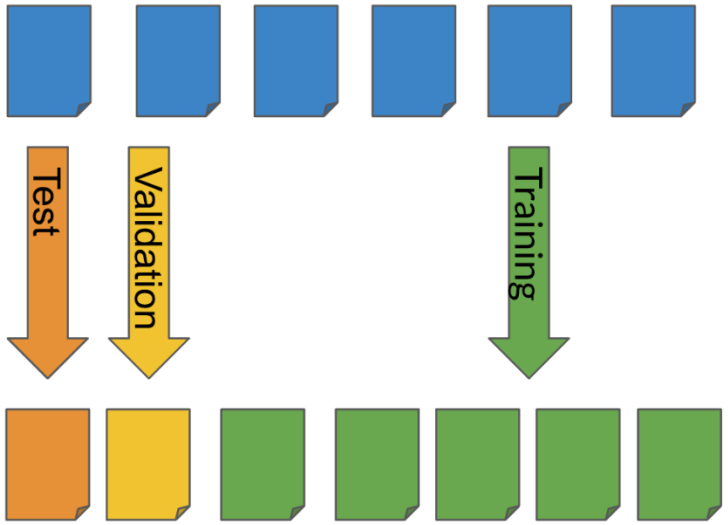
Train

Validation

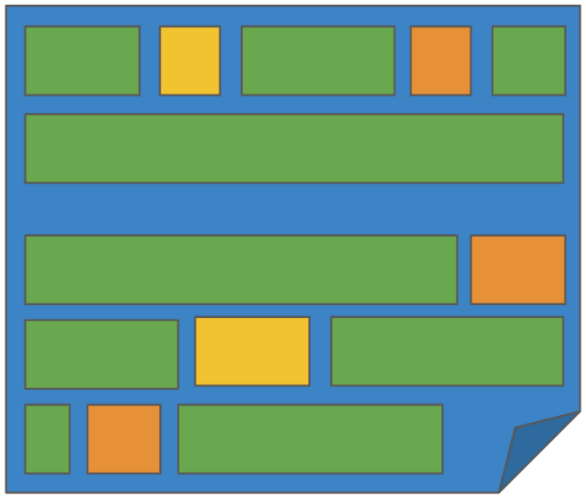
Test

ML Models for NLP: Splitting the data

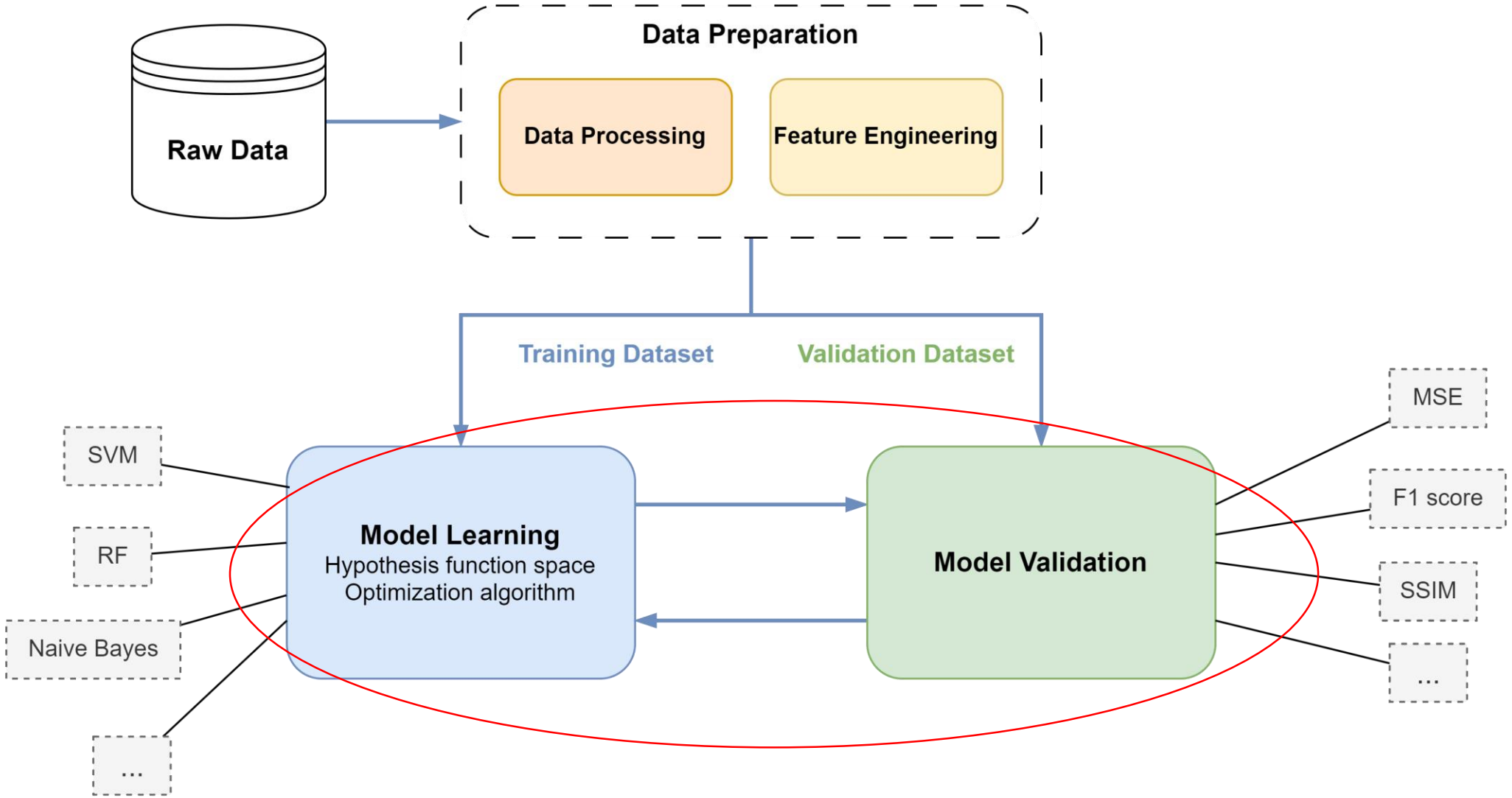
- Continuous text



- Random short sequences



ML Workflow (Supervised)



ML Models for NLP: Classification

- *Input:*
  - a document d
  - a fixed set of classes $C = \{c_1, c_2, \ldots, c_K\}$
  - A training set of n hand-labeled documents $(\mathrm{d}_1, \mathrm{c}_1), \ldots, (d_n, c_n)$
- *Output:*
  - a learned classifier $\hat{f}: d \rightarrow c \in C$

ML Models for NLP: Classification

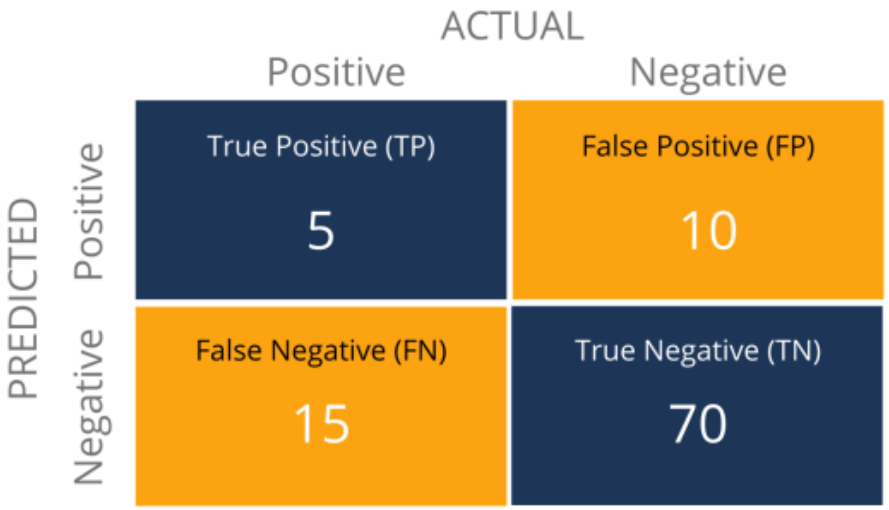
- *Input:*
  - a document d
  - a fixed set of classes $C = \{c_1, c_2, \ldots, c_K\}$
  - A training set of n hand-labeled documents $(\mathrm{d}_1, \mathrm{c}_1), \ldots, (d_n, c_n)$
- *Output:*
  - a learned classifier $\hat{f}: d \rightarrow c \in C$
  - Actually $\hat{f}: d \rightarrow (\pi_1, \ldots, \pi_k) \in [0,1]^K$

ML Models for NLP: Classification

- Any kind of classifier can be used:
  - Naïve Bayes
  - Logistic regression
  - Support-vector machines
  - Random Forest
  - k-Nearest Neighbors
  - …

Metrics for Classification:

- Usual classification metrics can be used:

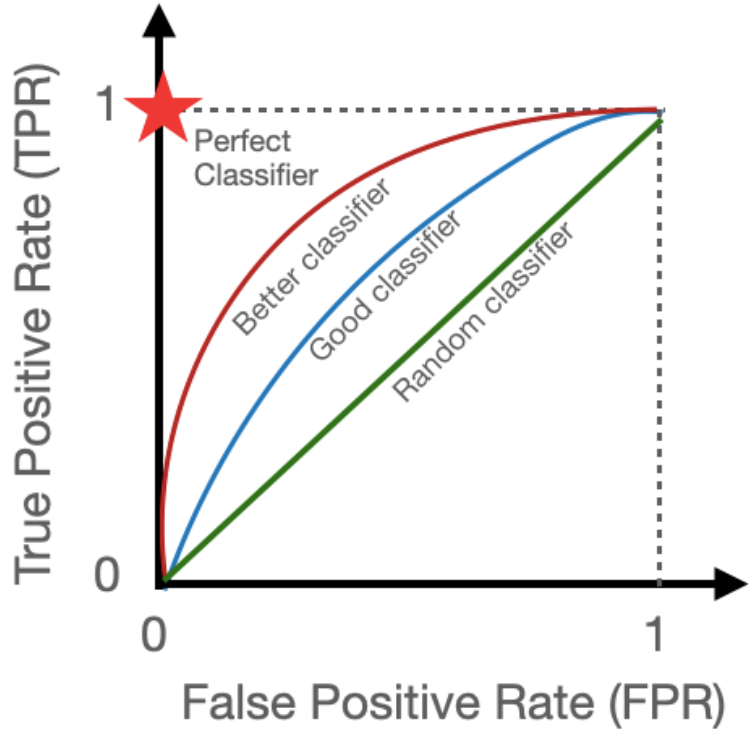


$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = 0.75$$

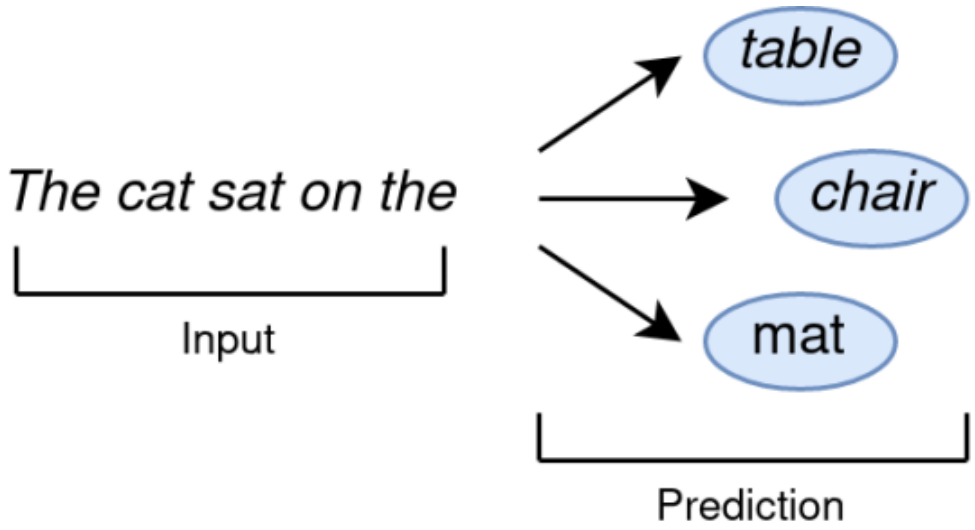
$$\text{Recall} = \frac{TP}{TP + FN} = 0.25$$

$$\text{Precision} = \frac{TP}{TP + FP} = 0.33$$

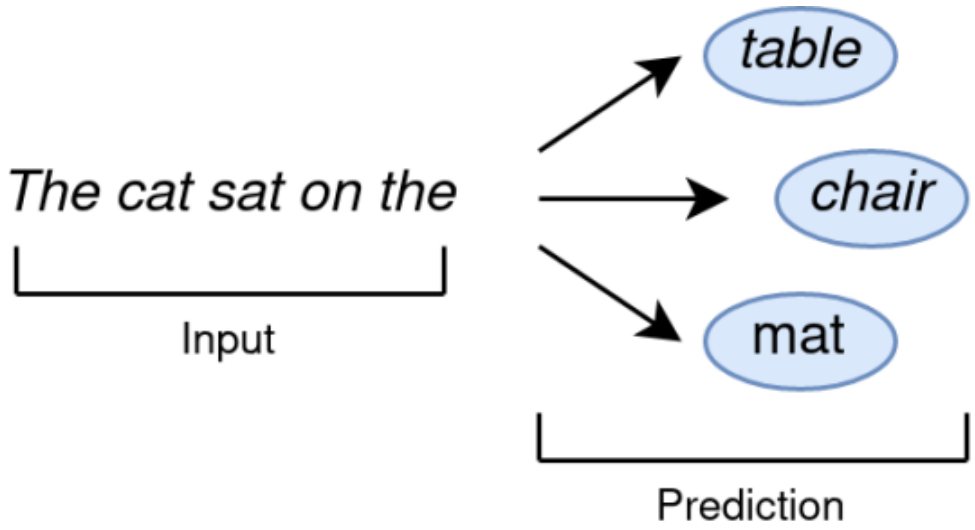
$$\text{F1 Score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} = 0.28$$

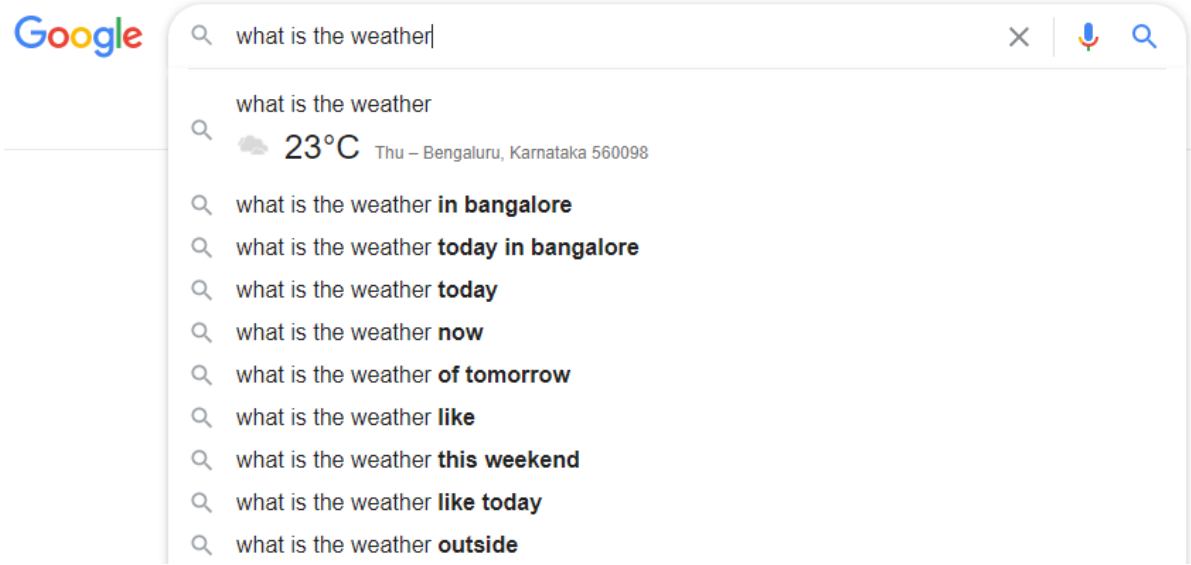


ML Models for NLP: Language Models

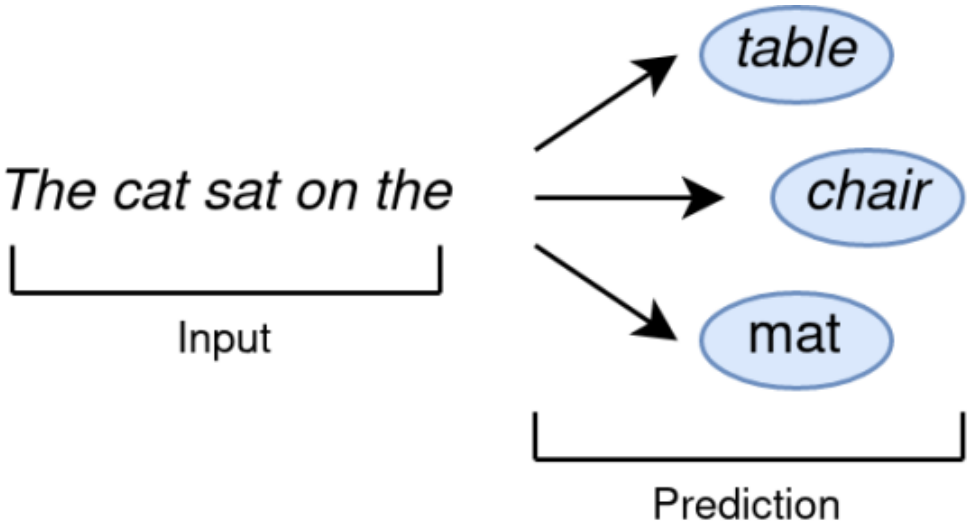


ML Models for NLP: Language Models





ML Models for NLP: Language Models



$P(\text{table} \mid \text{The cat sat on the})$

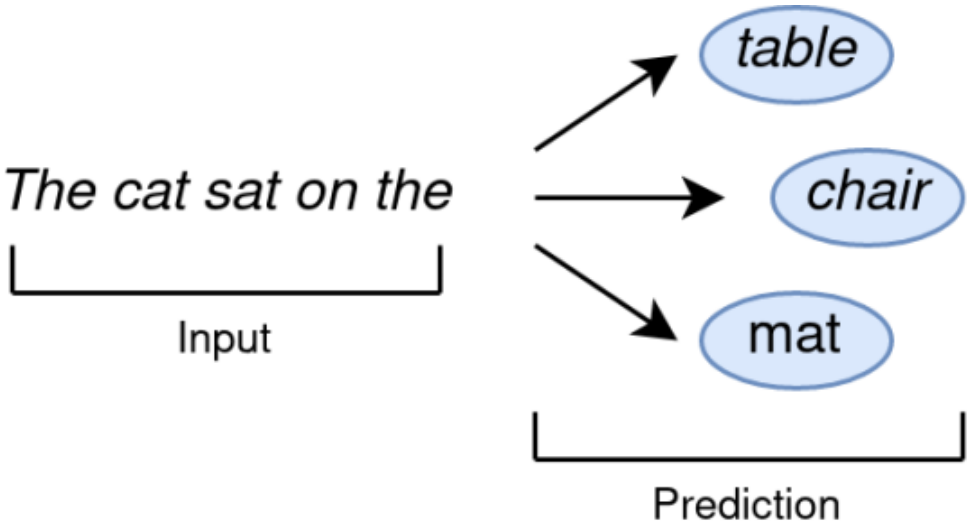
$P(\text{chair} \mid \text{The cat sat on the})$

$P(\text{mat} \mid \text{The cat sat on the})$

...

$$P(\text{table} \mid \text{The cat sat on the}) = ?$$

ML Models for NLP: Language Models



$P(\text{table} \mid \text{The cat sat on the})$

$P(\text{chair} \mid \text{The cat sat on the})$

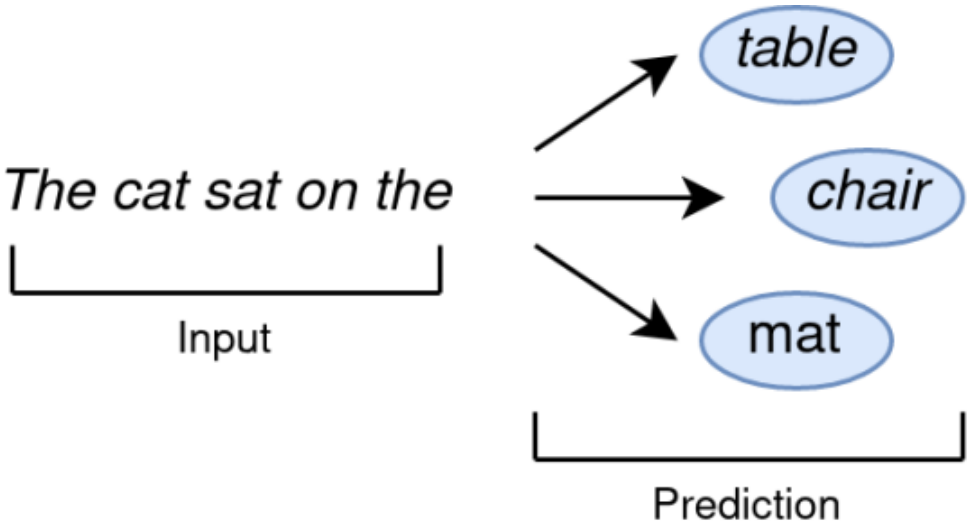
$P(\text{mat} \mid \text{The cat sat on the})$

...

$$P(\text{table} \mid \text{The cat sat on the}) = ?$$

To be estimated from data

ML Models for NLP: Language Models



$P(\text{table} \mid \text{The cat sat on the})$

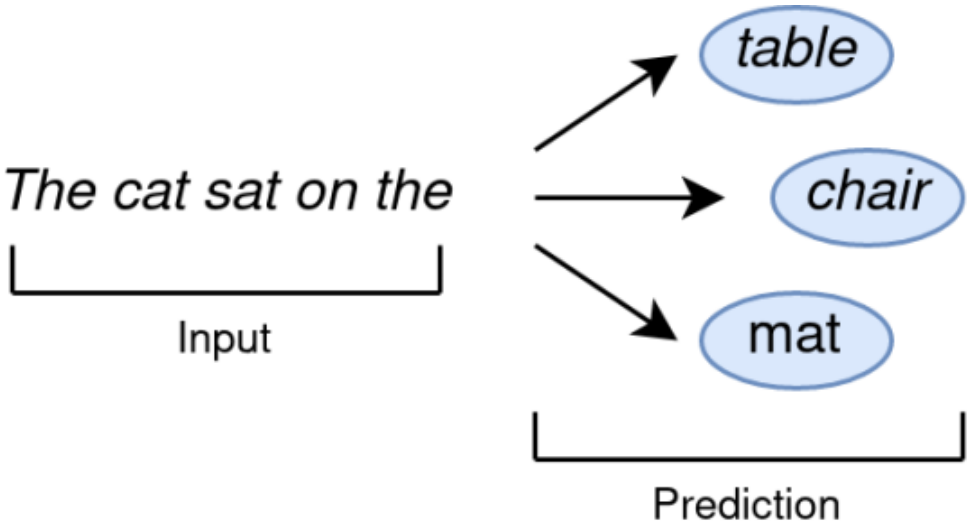
$P(\text{chair} \mid \text{The cat sat on the})$

$P(\text{mat} \mid \text{The cat sat on the})$

...

$$P(\text{table} \mid \text{The cat sat on the}) = \frac{\#\,(The\ cat\ sat\ on\ the\ table)}{\#\,(The\ cat\ sat\ on\ the)}$$

ML Models for NLP: Language Models



$P(\text{table} \mid \text{The cat sat on the})$

$P(\text{chair} \mid \text{The cat sat on the})$

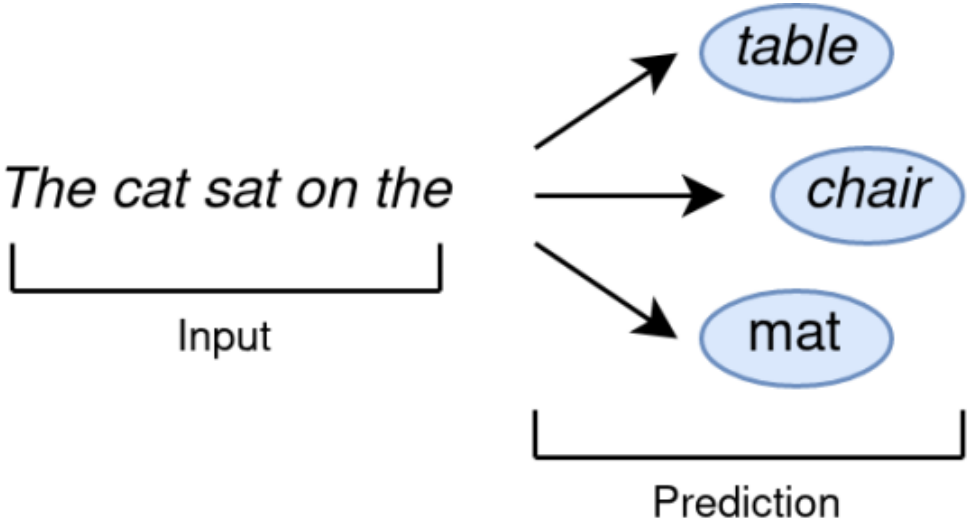
$P(\text{mat} \mid \text{The cat sat on the})$

...

$$P(\text{table} \mid \text{The cat sat on the}) = \frac{\#\,(The\ cat\ sat\ on\ the\ table)}{\#\,(The\ cat\ sat\ on\ the)}$$

We'll hardly see enough data for estimating these

ML Models for NLP: Language Models



$P(\text{table} \mid \text{The cat sat on the})$

$P(\text{chair} \mid \text{The cat sat on the})$

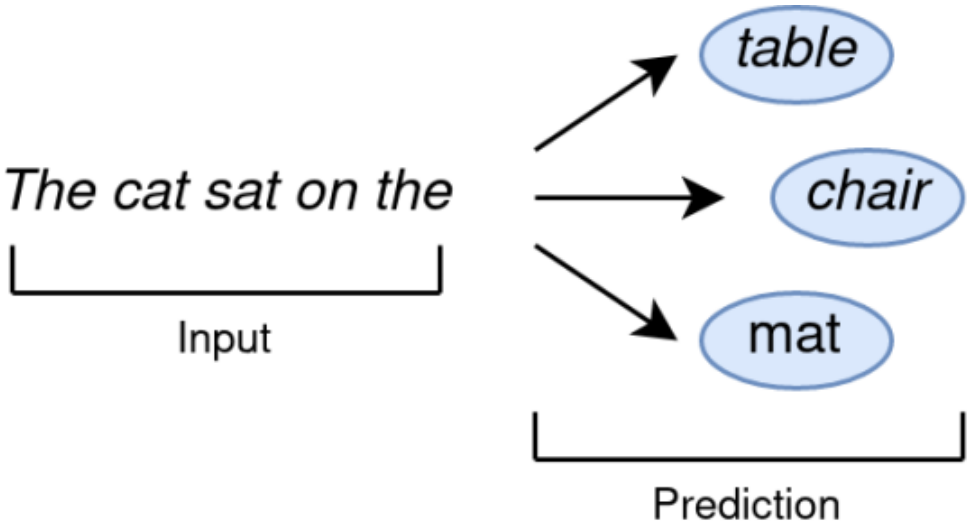
$P(\text{mat} \mid \text{The cat sat on the})$

...

$$P(\text{table} \mid \text{The cat sat on the}) \approx \begin{cases} P(table)\ or \\ P(table \mid the)\ or \\ P(table \mid on\ the)\ or \\ P(table \mid sat\ on\ the)\ or \\ \dots \\ P(w_i \mid w_1 w_2 \dots w_{i-1}) \end{cases}$$

Markov Assumption

ML Models for NLP: Language Models



$P(\text{table} \mid \text{The cat sat on the})$

$P(\text{chair} \mid \text{The cat sat on the})$

$P(\text{mat} \mid \text{The cat sat on the})$

...

$$P(\text{table} \mid \text{The cat sat on the}) \approx \begin{cases} P(table)\ or \\ P(table \mid the)\ or \\ P(table \mid on\ the)\ or \\ P(table \mid sat\ on\ the)\ or \\ \dots \\ P(w_i \mid w_1 w_2 \dots w_{i-1}) \end{cases}$$

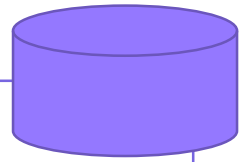
Markov Assumption

Increasing context and complexity

ML Models for NLP: Language Models

$$P(w_i \mid w_{i-1}) = \frac{\#\,(w_{i-1}, w_i)}{\#\,(w_{i-1})}$$

Training corpus



<s> I am a Student </s>
<s> Student I am </s>
<s> I do like Machine Learning </s>

$P(I \mid < s >) =$

$P(Student \mid < s >) =$

$P(am \mid I) =$

$P(</s > \mid Student) =$

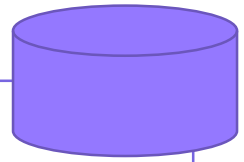
$P(Student \mid am) =$

$P(do \mid I) =$

ML Models for NLP: Language Models

$$P(w_i \mid w_{i-1}) = \frac{\#\,(w_{i-1}, w_i)}{\#\,(w_{i-1})}$$

Training corpus



<s> I am a Student </s>
<s> Student I am </s>
<s> I do like Machine Learning </s>

$$P(I \mid <s>) = \frac{2}{3} \qquad P(Student \mid <s>) = \frac{1}{3} \qquad P(am \mid I) = \frac{2}{3}$$

$$P(</s> \mid Student) = \frac{1}{2} \qquad P(Student \mid am) = \frac{1}{2} \qquad P(do \mid I) = \frac{1}{3}$$

Metrics for text generation

- **Perplexity**: Measures how well the model predicts a (test) sequence (used for language models)
- **BLEU (Bilingual Evaluation Understudy)**: Measures n-gram overlap between generated and reference text (used for machine translation).
- **ROUGE (Recall-Oriented Understudy for Gisting Evaluation)**: Focuses on content overlap (used for summarization).
- …

NLP Workflow

01. Data Collection

Open Source database
Web scrapping
Crowdsourcing
Social Media data
...

02. Data Cleaning

Unicode normalization
Regular Expression
OCR
Deduplication
...

03. Pre-Processing

Tokenization
Lowercasing
StopWord removal
Stemming
...

04. Feature Eng.

Bag of Words
TF-IDF
N-grams
Word Embedding
...

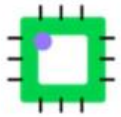
Feedback loop

07. Deployment

Export
Input pipeline
Scaling
Monitoring
...

06. Evaluation

F1-score
MAP
ROUGE
BLEU
...



05. Model Building

Naive Bayes
SVM
RNN, LSTM
Transformers
...

FRENCH INSTITUTES OF TECHNOLOGY

When you penalize your Natural Language Generation model for large sentence lengths