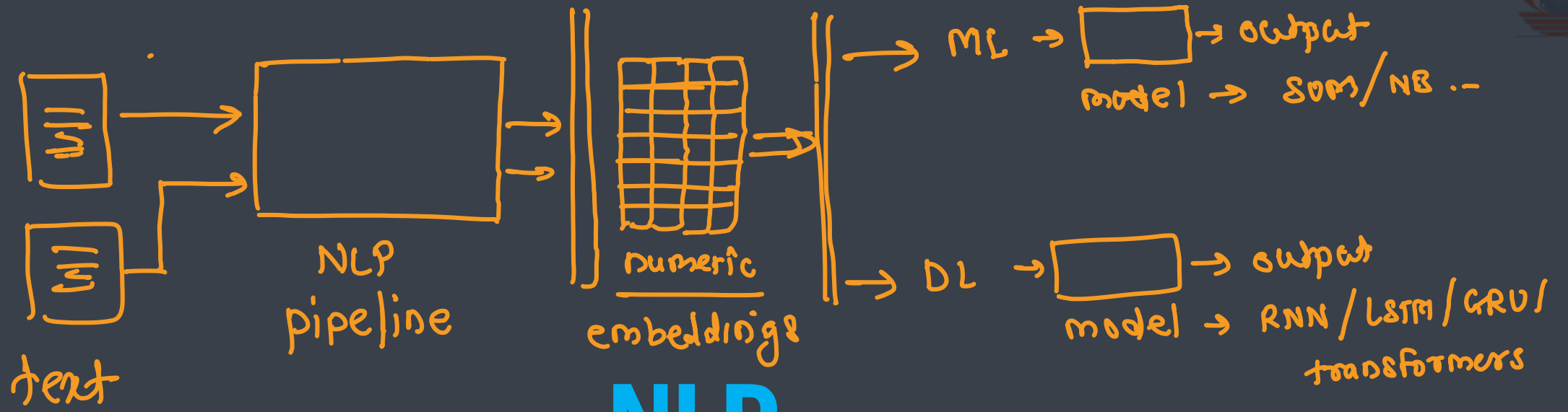
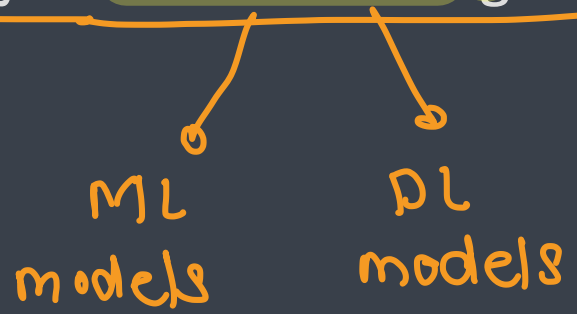

text
NLP
pipeline
numeric
embeddings
ML
model
output
SVM/NB ...
DL
model
output
RNN / LSTM / GRU /
transformers

NLP

What is NLP?

- Natural Language Processing (NLP) is a field of artificial intelligence (AI) that focuses on the interaction between computers and humans through natural language. → English - other language
- Its goal is to enable computers to understand, interpret, and respond to human language in a way that is both meaningful and useful. → context → output → sentiment, QA etc.
- The ultimate objective of NLP is to read, decipher, understand, and make sense of the human languages in a manner that is valuable → machine can understand
- Most NLP techniques rely on machine learning to derive meaning from human languages



Uses cased of NLP

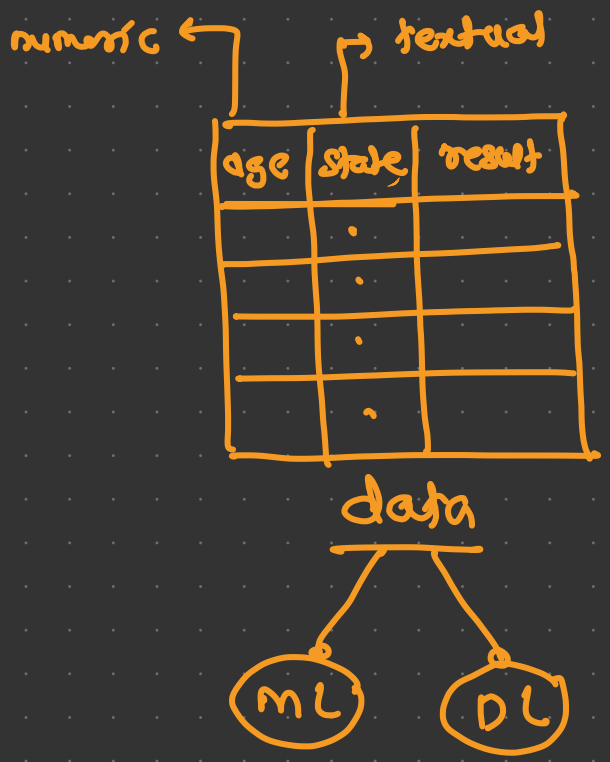
→ sentiment analysis – [+ve, -ve, neutral]

- **Text Analysis:** Understanding and extracting information from text data, including sentiment analysis, topic modeling, and keyword extraction.
- **Machine Translation:** Automatically translating text from one language to another (e.g., Google Translate) → DL model → transformers → LSTM / GPU
- **Speech Recognition:** Converting spoken language into written text, enabling applications like virtual assistants (e.g., Siri, Alexa)
- **Chatbots and Conversational Agents:** Designing systems that can engage in dialogue with users, providing responses to queries and performing tasks → similar to humans
- **Information Retrieval:** Enhancing search engines to retrieve relevant information based on user queries
- **Text Generation:** Creating coherent and contextually relevant text, as seen in applications like summarization and creative writing → Gen AI → GAN
- **Named Entity Recognition (NER):** Identifying and classifying key entities (like people, organizations, and locations) within a text → basis of Event Extraction

p summarization, event Extraction

Libraries for NLP

- Natural Language Toolkit (NLTK)
  - NLTK is one of the top frameworks for creating Python applications that can operate on human language data
  - Sentence identification, tokenization, lemmatization, stemming, parsing, chunking, and POS tagging are just a few of the text processing functions that it has
  - Over 50 corpora and lexical resources can be accessed through NLTK's user-friendly interfaces
- spaCy
  - Python's spaCy is an open-source NLP package
  - It allows you to create applications that process massive amounts of text because it is specifically intended for use in production environments
  - It can be used to build information extraction or natural language processing systems
  - It has word vectors and pre-trained statistical models, and can accommodate more than 49 languages for tokenization
- TextBlob
  - TextBlob provides very convenient APIs for standard NLP tasks, including POS tagging, noun phrase extraction, sentiment analysis, classification, language translation, word inflection, parsing, n-grams, and WordNet integration
  - The objects it creates can be thought of as Python strings with NLP training



Non-NLP data

corpus → data used in NLP, a huge text

↳ documents → paragraphs/sentences of huge text

```
data = """ Welcome to NLP.
NLP is the most imp task in ML.
"""
```

textual

NLP data

NLP Pipeline : collection of stages executed sequentially

(unstructured) text → pipeline → structured data / numeric data

1. **Text Preprocessing** — data cleansing ✓ — token = word — sentence tokenization → word tokenization
   - **Tokenization:** Splitting text into individual words or phrases (tokens).
   - **Lowercasing:** Converting all text to lowercase to maintain consistency.
   - **Removing Punctuation and Special Characters:** Cleaning the text to remove any non-essential elements.
   - **Stop Word Removal:** Eliminating common words (like "the," "is," "and") that may not contribute significant meaning
   - **Lemmatization:** Converting the words to their base forms → stemming
   - **NER:** Finding named entity representation → person name, organization name, date

   Apple, Apples → apple (base form)

   data = ["...", "...", "..."...]

2. **Text Representation** → conversion of words into numeric format.
   - **Word Embeddings:** Converting tokens into numerical representations (e.g., using Word2Vec, GloVe, or BERT) that capture semantic meanings → context
   - **Bag of Words (BoW):** Representing text as a vector of word counts → frequently appearing → CountVectorizer
   - **Term Frequency-Inverse Document Frequency (TF-IDF):** Weighting the importance of words based on their frequency in a document relative to their frequency across a collection of documents

NLP Pipeline

Selection of features
Extraction of features

3. Feature Engineering

- Creating additional features from the text that might help in modeling, such as:
- N-grams (combinations of n words)
- Part-of-speech tags (Pos tag) → verb, noun, pro-noun etc.
- Named entities - (NER)

noun
apples - apple

playing - play
verb

4. Modeling

ML
DL

- Training a Machine Learning Model: Using labeled data to train models for tasks like classification, regression, or clustering → grouping similar words
- Using Pre-trained Models: Leveraging existing models (like BERT or GPT) for specific tasks, often fine-tuning them on domain-specific data → transfer learning → langChain → Ollama

5. Post-Processing

- Interpreting Model Outputs: Converting raw predictions into actionable insights (e.g., converting sentiment scores into categories). → building newer model
- Visualizing Results: Creating visual representations of the data and findings. → WordCloud

NLP Pipeline

6. **Evaluation** → classification
   - **Metrics:** Assessing model performance using metrics such as accuracy, precision, recall, F1-score, etc.
   - **Cross-Validation:** Ensuring the model's robustness by testing it on different subsets of data.

7. **Deployment** → deployment of application in production env. [making it accessible to end user]
   - Integrating the model into applications or systems to make it accessible for real-world use, such as chatbots or recommendation systems → cloud, native app, mobile app

8. **Monitoring and Maintenance**
   - Continuously monitoring the model's performance and updating it as needed based on new data or changing requirements

Text Preprocessing

Conversion of corpus (documents) into tokens

Text Preprocessing

corpus → document ⇒ corpus = ["...", "...", "..."]

raw

Text Document

document → sentences

sentences → words

noun, verb, adjective

brings word to base form

- Sentence Segmentation
- Tokenization
- Parts of Speech Tagging
- Lemmatization
  - stemming

unwanted words / noise

dependency graph

person name, org name etc.

- Stop Words
- Dependency Parsing
- Noun Phrases
- Named Entity Relationship

Data Structure representing parsed text

structured

Sentence Segmentation

- Sentence tokenization is the process of splitting a piece of text into individual sentences
- This is an important step in natural language processing (NLP) and text analysis, as it allows for a more structured understanding of text by isolating distinct units of meaning → sentences
- The process typically involves identifying sentence boundaries, which can be defined by punctuation marks like periods, exclamation points, and question marks
- For example, the text "Hello! How are you? I hope you are well." would be tokenized into the sentences: ["Hello!", "How are you?", "I hope you are well."]
- **Techniques**
  - **Rule-Based Approaches:** Simple methods that rely on predefined rules (e.g., looking for punctuation).
  - **Machine Learning:** More advanced methods that use trained models to predict sentence boundaries based on context and patterns in the text. meaning
  - **Hybrid Approaches:** Combining both rule-based and machine learning techniques for better accuracy.

Word Tokenization

- Word tokenization is the process of breaking down a text into individual words, or token
- This is a crucial step in natural language processing (NLP) and text analysis, as it allows for the handling of text data in a structured way
- The process typically involves splitting a string of text based on whitespace and punctuation
- For example, the sentence "Hello, world!" would be tokenized into the tokens: ["Hello", "world"]
- **Types of Tokenization**
  - **Word Tokenization:** Splitting text into individual words. It often considers punctuation and may handle contractions (e.g., "don't" → "do", "n't").
  - **Subword Tokenization:** Dividing words into smaller units or subwords, which can help handle rare or complex words. Techniques like Byte Pair Encoding (BPE) are commonly used.
  - **Character Tokenization:** Breaking text down to individual characters, which can be useful in specific applications like language modelling.

Predicting Parts of Speech for Each Token

lexical analysis → checking syntax

sl = "the boy goes to school" ⇒ the school goes to boy

- Part-of-Speech (POS) tagging is the process of assigning a part of speech to each word in a sentence, based on its definition and context →
- The parts of speech include categories such as nouns, verbs, adjectives, adverbs, pronouns, prepositions, conjunctions, and interjections
- **Lexical Analysis**: Each word in a sentence is analyzed to determine its most likely part of speech based on its context. → structure
- **Contextual Clues**: The meaning of a word can change depending on its surrounding words, so context plays a crucial role in accurate tagging

- **Tagging Techniques**
  - **Rule-Based Tagging:** Uses hand-crafted rules and dictionaries to assign POS tags. While simple, this approach can be limited by the rules' comprehensiveness. → dictionary lookup
  - **Statistical Tagging:** Involves using probabilistic models, such as Hidden Markov Models (HMMs), which use training data to learn the likelihood of certain tags following others. → probabilities
  - **Machine Learning Approaches:** Modern techniques utilize machine learning algorithms, such as Conditional Random Fields (CRFs) or deep learning models (e.g., LSTMs), to achieve higher accuracy.

Text Lemmatization

- Lemmatization is the process of reducing a word to its base or root form, known as a "lemma."
- Unlike stemming, which simply truncates words to their root form (often resulting in non-words), lemmatization considers the morphological analysis of the words, ensuring that the root form is a valid word in the language
- **Dictionary Lookup:** Lemmatization typically involves looking up words in a dictionary or a morphological analysis database to find their base forms.
- **Part of Speech Tagging:** It often requires knowing the part of speech of the word, as the lemma can differ based on its grammatical role. For example, "better" as an adjective lemmatizes to "good," while as a verb it may stay as "better."
- **Lemmatization vs. Stemming**
  - **Lemmatization:** Produces meaningful base forms and takes into account the context and grammar of the word.
  - **Stemming:** Usually involves simpler, rule-based reductions and may not always result in a valid word (e.g., "running" might stem to "run," but "better" might stem to "better" or "bett").

reviews	type
this res is good	1
this res is bad	0
service is awesome	1
⋮	⋮

BoW

this is a good restaurant

⇓

"this" "is" "a" "good" "rest"

tokens

[good: 5 , bad: 4, ..]

⇓

good:	→	1
bad:	→	0

Identifying Stop Words

- Stop words are common words in a language that are often filtered out or removed during natural language processing (NLP) and text analysis → they do not contribute in getting meaning of sentence
- These words typically include prepositions, conjunctions, articles, and other functional words that do not carry significant meaning by themselves. Examples of stop words in English include:
  - **Articles**: "a," "an," "the"
  - **Prepositions**: "in," "on," "at," "by"
  - **Conjunctions**: "and," "or," "but"
  - **Pronouns**: "he," "she," "it," "they"

Dependency Parsing → dependency tree → semantic nature of text

- Dependency parsing is a process in natural language processing (NLP) that involves analyzing the grammatical structure of a sentence to establish relationships between words
- This technique identifies how words depend on each other within a sentence, creating a tree-like structure that shows the syntactic relationships
- **Approaches** ← traditional
  - **Rule-Based Methods:** Use a set of predefined grammatical rules to identify dependencies.
  - **Statistical Methods:** Utilize probabilistic models trained on annotated corpora to learn and predict dependencies based on patterns in the data.
  - **Neural Networks:** Modern approaches often employ deep learning techniques, such as recurrent neural networks (RNNs) or transformer models, to achieve state-of-the-art results.

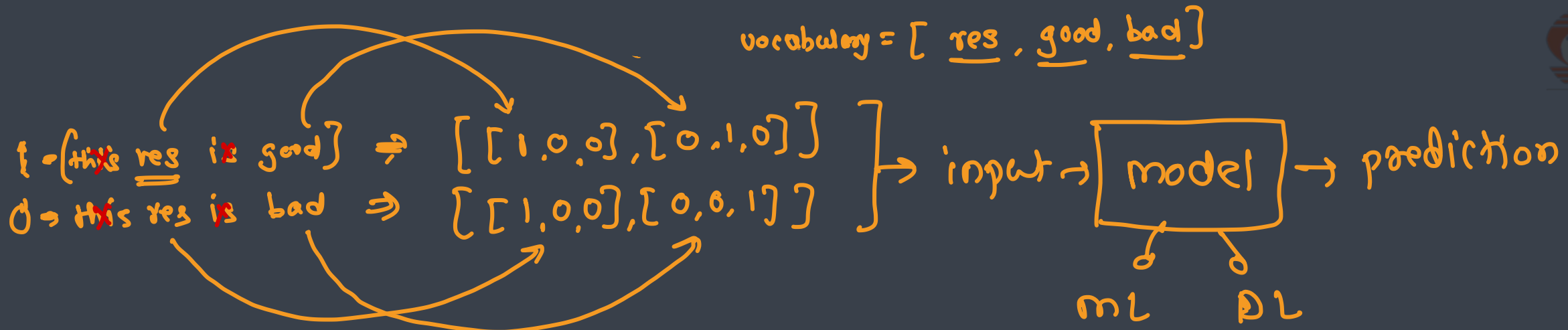
models to perform action

Finding Noun Phrases

- A noun phrase (NP) is a group of words that functions in a sentence as a noun
- It typically includes a noun (or pronoun) and its modifiers, which can be adjectives, determiners, or other phrases that provide additional information about the noun
- Noun phrases can serve various roles in a sentence, such as the subject, object, or complement
- **Structure**
  - **Head Noun:** The main noun in the phrase (e.g., "dog" in "the big dog").
  - **Modifiers:** Words or phrases that provide more detail about the noun, such as adjectives, articles, or prepositional phrases (e.g., "the big" and "in the park" in "the big dog in the park").
- **Examples**
  - Simple NP: "The cat"
  - NP with Adjectives: "The fluffy white cat"
  - NP with a Prepositional Phrase: "The cat on the roof"
  - NP with a Relative Clause: "The cat that I saw yesterday"

Named Entity Recognition (NER)

- Named Entity Recognition (NER) is a subtask of natural language processing (NLP) that involves identifying and classifying named entities in text into predefined categories
- These entities can include names of people, organizations, locations, dates, quantities, monetary values, and other specific identifiers
- **Common Categories**
  - **People:** Names of individuals (e.g., "Albert Einstein")
  - **Organizations:** Names of companies, institutions, or groups (e.g., "Google", "United Nations")
  - **Locations:** Geographical entities (e.g., "New York", "Mount Everest")
  - **Dates:** Specific dates or time expressions (e.g., "January 1, 2023")
  - **Miscellaneous:** Other entities such as products, events, and artworks.
- **Approaches** ↪ dictionary lookup
  - **Rule-Based Methods:** Use handcrafted rules and regular expressions to identify entities based on specific patterns.
  - **Statistical Models:** Employ probabilistic models like Hidden Markov Models (HMMs) or Conditional Random Fields (CRFs) to recognize and classify entities based on context.
  - **Deep Learning:** Utilize neural network architectures, such as BiLSTMs and transformers (e.g., BERT), to achieve state-of-the-art performance in entity recognition.



Text Representation

conversion of textual data to numeric format

Text Representation

- Text representation in natural language processing (NLP) refers to the methods and techniques used to convert textual data into numerical formats that can be processed by machine learning algorithms
- Effective text representation is crucial for various NLP tasks, such as classification, translation, and sentiment analysis
- Here are some of the key methods and concepts involved in text representation
  - **Bag of Words (BoW)**
  - Term Frequency-Inverse Document Frequency (TF-IDF)
  - Word Embeddings
  - Contextualized Word Embeddings
  - **Sentence and Document Embeddings**
  - **Feature Engineering**
  - Dimensionality Reduction

- one Hot Encoding

One-Hot Encoding → binary vectors

- Represents each word as a binary vector with a length equal to the size of the vocabulary
- Each position corresponds to a specific word, with a 1 in the position of the word and 0s elsewhere
- E.g.
  - For example, if your vocabulary is ["cat", "dog", "fish"], the one-hot encodings would be:
  - "cat" → [1, 0, 0]
  - "dog" → [0, 1, 0]
  - "fish" → [0, 0, 1]

I love cats → [I, cats, dogs, love, white]

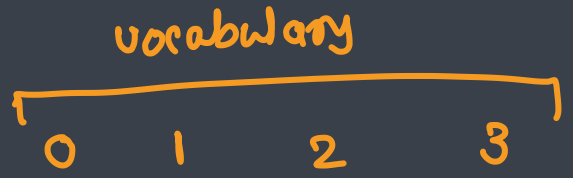
[[1 0 0 0 0], [0 0 0 1 0], [0 1 0 0 0]]

[1 0 0 0 0] → I

- **Advantages**
  - **Simplicity:** It's easy to understand and implement
  - **No Ordinality Assumption:** One-hot encoding does not impose any ordinal relationships among the words, which can be beneficial in many applications
- **Limitations**
  - **High Dimensionality:** If the vocabulary size is large, the resulting vectors can be very high-dimensional and sparse (mostly zeros), leading to inefficiency
  - **Lack of Semantic Information:** One-hot encoding does not capture any semantic relationships (↳ grammer) between words. For instance, "cat" and "dog" are treated as completely unrelated, even though they are both animals
  - **Increased Computational Load:** The high-dimensional representation can lead to increased computational costs for storage and processing

Bag of Words (BoW) → collection of unique words from corpus : CountVectorizer

- It is a popular model in natural language processing (NLP) used to represent text data.
- It simplifies the text into a collection of words without considering the order or grammar.
- **Steps to Create a Bag of Words Model:**
  - **Text Preprocessing:** This involves cleaning the text, which may include removing punctuation, converting to lowercase, and eliminating stop words (common words like "and," "the," etc.).
  - **Vocabulary Creation:** Compile a list of unique words (the vocabulary) from the entire text dataset.
    - **Vector Representation:** For each document (or sentence), create a vector that counts the occurrences of each word from the vocabulary. Each position in the vector corresponds to a word, and the value at that position represents its count in the document.
- **Example:**
  - Consider two sentences:
    - "I love cats."
    - "I love dogs."



document = review
review = "I love cats. cats are white..."
↳ [1, 1, 2, 0]

  - Assuming our vocabulary is ["I", "love", "cats", "dogs"], the BoW representation would be:
    - "I love cats." → [1, 1, 1, 0]
    - "I love dogs." → [1, 1, 0, 1]

- "food" "is" "not" "good" = ' food is good '

unigram = [food, not, good]

= [1 1 1] , [1 0 1]

bigram = [(food, not), (not good)]

= [1, 1) [0 0]

trigram = [(food not good)]

= [1] [0]

N-Gram

- In natural language processing (NLP), an **n-gram** is a contiguous sequence of n items (usually words or characters) from a given text or speech
- N-grams are used to analyze the structure and context of text data
- They help capture relationships and patterns that can be useful for various NLP tasks such as text classification, language modelling, and machine translation
- **Types of N-grams**
  - **Unigram**
    - An n-gram of size 1, which consists of individual words.
    - Example: From the sentence "I love cats," the unigrams are ["I", "love", "cats"]
  - **Bigram**
    - An n-gram of size 2, which consists of pairs of consecutive words.
    - Example: From the same sentence, the bigrams are ["I love", "love cats"]
  - **Trigram**
    - An n-gram of size 3, which consists of triplets of consecutive words
    - Example: ["I love cats"]
  - **Higher-order n-grams**
    - You can have n-grams of size 4, 5, or more, depending on your analysis needs.

N-grams

How are | you?
(x) (y)

How are you?
x y

- **Applications of N-grams**
  - **Text Prediction:** N-grams can be used to predict the next word in a sentence based on the previous words.
  - **Spam Detection:** N-grams help in identifying patterns in text that are characteristic of spam or non-spam content.
  - **Sentiment Analysis:** By analyzing n-grams, one can capture phrases that convey sentiment, improving the accuracy of sentiment classification models.
  - **Machine Translation:** N-grams help in understanding the context and relationships between words in different languages.
- **Limitations of N-grams**
  - **Sparsity:** As the size of n increases, the number of possible n-grams grows exponentially, leading to sparse representations, especially in large vocabularies.
  - **Contextuality:** N-grams can lose context, as they do not capture the meaning or relationship beyond their immediate sequence.

Term Frequency-Inverse Document Frequency (TF-IDF)

- It is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (corpus)
- It combines two components: **Term Frequency (TF)** and **Inverse Document Frequency (IDF)**
- TF-IDF is widely used in text mining and information retrieval to transform text into a meaningful representation for machine learning models
- **Applications of TF-IDF**
  - **Information Retrieval:** Enhances search engines by ranking documents based on their relevance to the search query.
  - **Text Classification:** Serves as a feature representation for machine learning algorithms.
  - **Topic Modelling:** Helps identify significant topics in a collection of documents.

Term Frequency-Inverse Document Frequency (TF-IDF)

- Components of TF-IDF
  - Term Frequency (TF)
    - Measures how frequently a term appears in a document
    - TF gives higher values to terms that appear more frequently in a document
    - $tf(t,d) = \frac{number\ of\ times\ term\ t\ appears\ in\ document\ d}{total\ number\ of\ terms\ in\ doument\ d}$
  - Inverse Document Frequency (IDF)
    - Measures how important a term is across the entire corpus
    - $IDF(t) = log(\frac{total\ number\ of\ documents}{number\ of\ documents\ containing\ term\ t})$
  - TF-IDF
    - The final TF-IDF score for a term t in a document d is calculated as:
      - TF-IDF (t, d) = TF (t, d) * IDF (t)

Word Embeddings

male = = king : 28598851 embedding queen = female

- Word embeddings are a type of word representation that allows words to be represented as dense vectors in a continuous vector space
- Unlike traditional methods such as one-hot encoding or Bag of Words, which produce high-dimensional and sparse vectors, word embeddings capture semantic meanings and relationships between words in a more efficient and meaningful way
- Key Characteristics of Word Embeddings
  - Dense Representation
    - Word embeddings typically reduce the dimensionality of the representation (e.g., from a vocabulary size of thousands to a vector of size 100-300)
  - Semantic Similarity
    - Words with similar meanings are represented by vectors that are close together in the embedding space. For example, "king" and "queen" might be close to each other, while "king" and "apple" would be far apart
  - Contextual Relationships
    - Word embeddings can capture various linguistic relationships. For example, the difference between the embeddings for "king" and "queen" can be similar to the difference between "man" and "woman."

Word Embeddings - Models

- **Word2Vec**
  - Developed by Google, Word2Vec can create word embeddings using two model architectures
    - **CBOW** predicts a target word based on its surrounding context words
    - **Skip-Gram** does the opposite, predicting the surrounding context words given a target word
- **GloVe (Global Vectors for Word Representation)**
  - Developed by Stanford, GloVe uses matrix factorization techniques on the word co-occurrence matrix to produce embeddings.
  - It leverages global word-word co-occurrence statistics from a corpus
- **FastText**:
  - Developed by Facebook, FastText represents words as bags of character n-grams
  - This allows it to capture subword information, making it effective for morphologically rich languages and handling out-of-vocabulary words
- **Contextualized Embeddings**
  - Models like ELMo, BERT, and GPT generate embeddings that are context-dependent, meaning the same word can have different embeddings based on its usage in a sentence

Word Embeddings - Applications

- **Text Classification**
  - Word embeddings serve as features for machine learning models to improve classification accuracy
- **Sentiment Analysis**
  - They help capture the nuanced meanings of words, aiding in determining sentiment
- **Machine Translation**
  - Embeddings assist in translating words and phrases across different languages
- **Information Retrieval**
  - They improve search algorithms by understanding semantic meanings

NLP

Text processing → tokenization, stop words .....

text representation → one hot, BoW, Ngram, TF-IDF, Word Embeddings

Modelling

→ ML : NB, SVM, KNN

→ DL : RNN → LSTM, GRU → Encoder, Decoder, bidirectional encoder, transformers ***

LLM