1. Tokenization

- **What it is:** The first step in natural language processing (NLP) is to break down a piece of text into smaller units called tokens. These tokens can be words, individual characters, or sub-word units like n-grams.

- **Example:**

    - Sentence: "The quick brown fox jumps over the lazy dog."
    - Tokens: "The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"

2. Stemming

- **What it is:** A technique that reduces a word to its root or base form by removing suffixes (endings like "-ing", "-ed", "-s") and sometimes prefixes. It's a faster but less accurate method compared to lemmatization.

- **Example:**

    - "running" -> "run"
    - "better" -> "bet"
    - "studies" -> "studi"
- **Limitations:**

    - Can produce incorrect or meaningless stems (e.g., "studies" -> "studi" instead of "study").
    - Might not capture morphological variations correctly.

3. Lemmatization

- **What it is:** A more sophisticated technique that aims to transform a word to its dictionary form (lemma). It considers the grammatical context and uses morphological analysis to identify the correct base form.

- **Example:**

    - "better" -> "good"
    - "studies" -> "study"
    - "is" -> "be"
- **Advantages:**

    - It is more accurate than stemming in producing meaningful base forms.
    - Can handle irregular verbs and nouns correctly.

4. Porter's Algorithm

- **What it is:** A well-known stemming algorithm developed by Martin Porter. It's a rule-based approach that involves a series of steps to remove suffixes from English words.

- **Example:**

    - "s s f s" -> "s s"
    - "t f s" -> "t"
    - "s" -> "s'"

In Summary

Both stemming and lemmatization are crucial in NLP for tasks like text analysis, search engines, and information retrieval. They help reduce the dimensionality of text data and improve the accuracy of NLP models by normalizing words to their base forms. Lemmatization generally provides more accurate results but is computationally more expensive than stemming.

Sample Text:

This paragraph discusses the benefits of a particular type of analysis. It highlights that this analysis can uncover hidden patterns or features that are difficult to spot when looking at individual components. This analysis allows us to create a clearer and more understandable picture of a complex biological process.

1. Sentence token

- **Tokenization:** In the context of natural language processing (NLP), tokenization is the process of breaking down text into smaller units called tokens. These tokens can be words, individual characters, or even sub-word units like n-grams.
- **Sentence token:** In this case, a "sentence token" likely refers to tokenizing the text at the sentence level. This means dividing the paragraph into individual sentences.

2. word

- This simply refers to the most common type of token in NLP - individual words.

3. Porter's

- **Porter's Algorithm:** This is a well-known stemming algorithm. Stemming is a technique used to reduce words to their base or root form by removing suffixes (endings like "-ing", "-ed", "-s", etc.).

1. Stop Words

- **Definition:** Stop words are the most common words in a language that typically don't carry much significant meaning for text analysis.
- **Examples:** "a", "the", "is", "are", "and", "in", "of", "to"
- **Reason for Removal:** Removing stop words can help improve the efficiency and accuracy of text analysis techniques by focusing on the more important words (content words) that convey the actual meaning of the text.

2. Bag of Words

- **Definition:** A simple way to represent text data. It's a collection of all the unique words (or tokens) present in a set of documents, along with their frequencies.

- **Example:**

    - Document 1: "He is a good boy."
    - Document 2: "She is a good girl."
    - Document 3: "Boy's and girl's are good."
    - Bag of Words:
        - Good: 3
        - Boy: 2
        - Girl: 2
        - He: 1
        - She: 1
        - is: 3
        - a: 2
- **Key Characteristics:**

    - Order of words is ignored.
    - Only word frequencies are considered.

3. Removing Stop Words

- **Step 1:** Identify and remove stop words from each document.

    - Document 1: "He is a good boy." -> "Good boy"
    - Document 2: "She is a good girl." -> "Good girl"
    - Document 3: "Boy's and girl's are good." -> "Boy girl good"
- **Step 2:** Create a new Bag of Words based on the processed documents.

    - Good: 3
    - Boy: 2
    - Girl: 2

In Summary:

The image illustrates the concept of stop words and how they are removed before creating a Bag of Words representation for text analysis. By removing stop words, we can focus on the more informative words in the text and improve the performance of NLP models.

TF-IDF

- **Purpose:** TF-IDF is a numerical statistic that reflects how important a word is to a document in a collection of documents.
- **Significance:** It helps to identify the most relevant words within a document and across a collection of documents.

Components

1. **Term Frequency (TF)**
    - **Definition:** Measures how frequently a term appears within a single document.

Calculation:
TF(term, document) = (Number of times the term appears in the document) / (Total number of terms in the document)

    -
2. **Inverse Document Frequency (IDF)**
    - **Definition:** Measures how rare or common a term is across the entire document collection.

Calculation:
IDF(term, document collection) = log((Total number of documents) / (Number of documents containing the term))

    -

TF-IDF Calculation

Formula:
TF-IDF(term, document, document collection) = TF(term, document) * IDF(term, document collection)

-

Example in the Image

- **Documents:**
    - Sen1: "He is a good boy."
    - Sen2: "She is a good girl."
    - Sen3: "Boy's and girl's are good."
- **Calculations:**
    - **TF for "Good" in Sen1:**
        - TF(Good, Sen1) = 1 / 4 = 0.25
    - **IDF for "Good":**
        - IDF(Good) = log(3 / 3) = log(1) = 0
    - **TF-IDF for "Good" in Sen1:**
        - TF-IDF(Good, Sen1) = 0.25 * 0 = 0
    - **Similarly, other TF-IDF values are calculated for "Boy" and "Girl" in each sentence.**

Interpretation

- **High TF-IDF:** Indicates that the term is important within the specific document but relatively rare across the entire collection.
- **Low TF-IDF:** Suggests that the term is either common in the document or frequently appears across the entire collection, making it less distinctive.

Applications

- **Information Retrieval:** Ranking search results based on relevance.
- **Text Summarization:** Identifying key sentences or phrases.
- **Topic Modeling:** Discovering underlying themes in a collection of documents.

Example -

1. Original Sentences:

- These are the initial sentences given:
    - D1: "Sudip does not dance"
    - D2: "Sudip love doing"
    - D3: "Sudip stops doing"

2. After Lemmatization:

- **Lemmatization:** This step aims to reduce words to their dictionary form (lemma).
    - "doing" in D2 is lemmatized to "dance" as it's the base form of the verb.
    - The other words remain unchanged as they are already in their base form.
- D1: "Sudip does not dance"
- D2: "Sudip love dance"
- D3: "Sudip stop dance"

3. Apply Stop Words:

- **Stop Words:** These are common words like "a", "the", "is", "are", etc., that often don't carry significant meaning for text analysis.
- In this example, "not" and "does" are likely considered stop words and are removed.
- D1: "Sudip do dance"
- D2: "Sudip love dance"
- D3: "Sudip stop dance"

4. Table

- This table summarizes the word frequencies and Term Frequency (TF) for each word in each sentence.
    - **Frequency:** The total number of times a word appears in all the sentences.
    - **TF (Term Frequency):** The number of times a word appears in a specific sentence divided by the total number of words in that sentence.

Example:

- "Sudip" appears in all three sentences, so its frequency is 3.
- In sentence 1, "Sudip" appears once out of three words, so its TF in sentence 1 is 1/3.

In Summary:

The image demonstrates a basic text processing pipeline, including lemmatization and stop word removal. These steps are often used in Natural Language Processing (NLP) tasks to prepare text data for further analysis, such as creating a Bag of Words representation or calculating TF-IDF.

Text Vectorization

- This refers to the process of converting text data into numerical vectors that can be used as input for machine learning models. The image lists several common text vectorization techniques:

    1. **Bag of Words:**

        - Represents text as a collection of words, ignoring their order.
        - Each unique word is assigned a position in the vector, and the value at that position represents the frequency of the word in the text.
    2. **TF-IDF:**

        - Stands for Term Frequency-Inverse Document Frequency.
        - Weights words based on their importance within a document and across a collection of documents.
        - Words that are frequent in a document but rare across the collection are given higher weights.
    3. **Word2Vec:**

        - Represents words as dense vectors (embeddings) in a continuous vector space.
        - Words with similar meanings are placed closer together in this space.
    4. **One-hot Encoding:**

        - Creates a sparse vector where each position corresponds to a unique word in the vocabulary.
        - The vector has a value of 1 at the position corresponding to the word and 0 elsewhere.

Example

- The image provides three sentences:

    - "My name is Raz Bond"
    - "Bond is Dangerous"
    - "Raz is coming"
- **One-hot Encoding Example:**

    - The words "My", "name", "is", "Raz", "Bond", "Dangerous", and "coming" are assigned unique positions in the vector space.
    - In the sentence "Bond is coming", only the positions corresponding to "Bond", "is", and "coming" would have a value of 1, while all other positions would be 0.

In Summary

The image provides a brief overview of deep learning and several text vectorization techniques commonly used in NLP. These techniques are crucial for converting human language into a format that can be processed by machine learning algorithms.

Cosine Similarity Explained

Cosine Similarity is a metric used to measure how similar two vectors are. It's particularly useful in Natural Language Processing (NLP) tasks where we represent text data as vectors. By comparing the cosine similarity between sentence vectors, we can estimate how similar or relevant the sentences are to each other.

Formula Breakdown:

The formula for Cosine Similarity is:

$\cos(\theta) = A \cdot B / ||A|| \, ||B||$

where:

- `cos(θ)` (cosine theta): This represents the cosine similarity score between vectors A and B. It ranges from -1 to 1.
- `A · B` (dot product): This is the dot product of vectors A and B. It captures the overall direction and magnitude of the vectors.
- `||A||` (magnitude of A): This is the length or magnitude of vector A.
- `||B||` (magnitude of B): This is the length or magnitude of vector B.

Intuitive Explanation:

Imagine two vectors like arrows in a high-dimensional space. The cosine similarity essentially calculates the cosine of the angle (θ) between these two arrows.

- **High Cosine Similarity (close to 1):** When the angle (θ) between the vectors is small, the cosine value approaches 1. This indicates that the vectors are pointing in similar directions, signifying high similarity between the sentences they represent.
- **Low Cosine Similarity (close to 0):** If the angle (θ) is large (closer to 90 degrees), the cosine value approaches 0. This suggests the vectors are nearly perpendicular, indicating low similarity between the sentences.
- **Negative Cosine Similarity (between -1 and 0):** A negative cosine similarity implies the vectors point in opposite directions. This can indicate contrasting or opposite meanings between the sentences.

Example in the Image:

The image provides an example using the sentences "Hello world!" and "Hello!".

- **Vector Representation:** These sentences are likely converted into one-hot encoded vectors, where each word has a unique position and a value of 1 if the word exists and 0 otherwise.
- **Cosine Similarity Calculation:** The dot product and magnitudes of the vectors are calculated, resulting in a cosine similarity of 0.
- **Interpretation:** A cosine similarity of 0 indicates the sentences have no semantic similarity, which aligns with our intuition.

Applications of Cosine Similarity in NLP

- **Document Retrieval:** Ranking documents based on their relevance to a query.
- **Text Summarization:** Identifying key sentences that capture the essence of a document.
- **Recommendation Systems:** Recommending similar items or content based on user preferences.
- **Chatbots:** Understanding user intent and responding with relevant information.

By understanding Cosine Similarity, you can gain valuable insights into the relationships between textual data and leverage it for various NLP tasks.

Example: Comparing Product Descriptions

Let's say we have two product descriptions:

Product A: "Comfortable running shoes for men with excellent cushioning and breathable mesh upper."

Product B: "Men's athletic shoes designed for running, featuring a breathable mesh upper and superior cushioning."

1. Vector Representation:

We represent each product description as a vector, where each dimension corresponds to a word in the vocabulary. For simplicity, let's consider the following words:

- "comfortable"
- "running"
- "shoes"
- "men"
- "cushioning"
- "breathable"

- "mesh"
- "upper"
- "athletic"
- "designed"
- "superior"

Now, we create vectors for each product:

- **Product A:** [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0]
- **Product B:** [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

(Note: This is a simplified representation. In real-world scenarios, we would use techniques like TF-IDF to weigh words based on their importance.)

2. Calculate Cosine Similarity:

- **Dot Product (A · B):** (1 * 0) + (1 * 1) + (1 * 1) + (1 * 1) + (1 * 1) + (1 * 1) + (1 * 1) + (1 * 1) + (0 * 1) + (0 * 1) + (0 * 1) = 7
- **Magnitude of A (||A||):** $\sqrt{(1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 0^2 + 0^2 + 0^2)} = \sqrt{8} = 2.83$
- **Magnitude of B (||B||):** $\sqrt{(0^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2)} = \sqrt{10} = 3.16$
- **Cosine Similarity:** 7 / (2.83 * 3.16) ≈ 0.78

3. Interpretation:

A cosine similarity of 0.78 suggests a high degree of similarity between the two product descriptions. This indicates that they describe similar products, which is expected given the overlapping keywords.

Key Points:

- Cosine similarity is a measure of similarity between vectors, often used to compare text documents.
- It calculates the cosine of the angle between two vectors.
- A value closer to 1 indicates higher similarity, while values closer to 0 indicate lower similarity.
- Cosine similarity is widely used in information retrieval, recommendation systems, and other NLP tasks.

Language Model

- A language model is a statistical model that predicts the probability of a sequence of words.
- It can be used for various tasks like:
    - **Text Generation:** Generating human-like text, such as stories, poems, or code.
    - **Speech Recognition:** Transcribing spoken language into written text.
    - **Machine Translation:** Translating text from one language to another.
    - **Spell Checking:** Identifying and correcting spelling errors.

n-grams

- An n-gram is a contiguous sequence of n-words in a given piece of text.
- Examples:
    - **Unigram:** "I", "am", "king", "of", "the" (single words)
    - **Bigram:** "I am", "am king", "king of", "of the" (pairs of words)
    - **Trigram:** "I am king", "am king of", "king of the" (sequences of three words)

n-gram Probabilities

- **Unigram Probability:** The probability of a single word occurring.
    - `P(w) = count(w) / total`
    - where `count(w)` is the number of times word "w" appears in the text and `total` is the total number of words in the text.
- **Bigram Probability:** The probability of a word occurring given the previous word.
    - `P(w | w-1) = count(w, w-1) / count(w-1)`
    - where `count(w, w-1)` is the number of times the sequence "w-1 w" appears in the text, and `count(w-1)` is the number of times the word "w-1" appears.
- **Trigram Probability:** The probability of a word occurring given the previous two words.
    - `P(w | w-1, w-2) = count(w, w-1, w-2) / count(w-1, w-2)`

Using n-grams for Language Modeling

- n-gram models can be used to predict the next word in a sequence.
- For example, given a sentence "I am on the", a trigram model can predict the next word by calculating the probabilities of different words following "on the" based on their frequencies in the training data.

Limitations of n-grams:

- **Data Sparsity:** Higher order n-grams (like trigrams) require a large amount of training data to estimate probabilities accurately.

- **Limited Context:** n-grams only consider a limited window of previous words, which might not be sufficient to capture long-range dependencies in language.

In Summary:

The image provides a basic introduction to language models, n-grams, and their use in predicting word probabilities. n-gram models, while simple, serve as a foundational concept for more complex language models like recurrent neural networks (RNNs) and transformers, which can capture longer dependencies and generate more fluent and coherent text.

1. Sentences and Vocabulary:

- You have a set of four sentences.
- The "Unique Vocabulary" lists all the distinct words present in these sentences.

2. Trigram Probability:

- **Definition:**
    - A trigram is a sequence of three words.
    - Trigram probability is the probability of a word occurring given the two preceding words.
- **Formula:**
    - `P(w<sub>i</sub> | w<sub>i-1</sub>, w<sub>i-2</sub>) = count(w<sub>i-2</sub>, w<sub>i-1</sub>, w<sub>i</sub>) / count(w<sub>i-2</sub>, w<sub>i-1</sub>)`
    - where:
        - `w<sub>i</sub>` is the current word
        - `w<sub>i-1</sub>` is the previous word
        - `w<sub>i-2</sub>` is the word before the previous word
        - `count(w<sub>i-2</sub>, w<sub>i-1</sub>, w<sub>i</sub>)` is the count of the trigram (sequence of three words)
        - `count(w<sub>i-2</sub>, w<sub>i-1</sub>)` is the count of the bigram (sequence of two words)
- **Example:**

    - `P(bought | The, Girl)` = `count(The, Girl, bought) / count(The, Girl)`

3. Basic Probability Concepts:

- **Conditional Probability:**
    - `P(B|A)`: Probability of event B happening given that event A has already happened.
    - `P(B|A) = P(A, B) / P(A)`
- **Joint Probability:**
    - `P(A, B)`: Probability of both events A and B happening.
    - `P(A, B) = P(A) * P(B|A)`
- **Chain Rule (for multiple events):**
    - `P(A, B, C, D) = P(A) * P(B|A) * P(C|A, B) * P(D|A, B, C)`

4. Language Modeling and Smoothing:

- **Language Modeling:** The task of predicting the next word in a sequence.
- **Underflow Problem:** When dealing with n-grams (like trigrams), the counts for many combinations of words will be zero, leading to zero probabilities. This can cause issues in calculations.
- **Laplace Smoothing (Add-one Smoothing):** A technique to address the underflow problem. It adds 1 to the count of each possible n-gram, preventing zero probabilities and improving model robustness.

5. Log Probabilities:

- **Log-likelihood:** Using logarithms of probabilities can improve numerical stability and prevent underflow issues when dealing with very small probabilities.
- The example shows the calculation of the log-likelihood of a sequence of words: " I like college".

In Summary:

The images present foundational concepts in probability and natural language processing, including:

- **n-gram probabilities**
- **Basic probability rules (conditional probability, joint probability, chain rule)**
- **Techniques for handling zero probabilities in language models (Laplace smoothing)**
- **Log-likelihood calculation**

These concepts are crucial for building and understanding various language models, such as those used for text generation, machine translation, and more.