

Natural Language Processing (NLP)

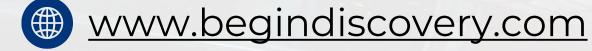
N Grams & Intro of Embedding

Equipping You with Research Depth and Industry Skills – Data Science Oriented By:

Dr. Zohair Ahmed



www.youtube.com/@ZohairAl





Introduction to Text Representation

- **Text Representation**: The process of converting text into numerical form for machine learning tasks.
- Importance in NLP: Text must be represented as vectors to be processed by algorithms.
- Common Methods:
 - Bag of Words (BoW)
 - TF-IDF
 - Word Embeddings
 - n-grams

What is an n-gram?

- Definition: A sequence of n consecutive words in a text.
- Example:
 - Unigrams (1-gram): "I", "am", "learning"
 - Bigrams (2-grams): "I am", "am learning"
 - Trigrams (3-grams): "I am learning"
- Why n-grams?: Captures context and word relationships beyond single words.

Bag of n-grams Overview

- Bag of n-grams: An extension of the Bag of Words (BoW) model.
 - Represents a text by counting occurrences of n-grams (not individual words).

Features:

- Can capture context between consecutive words.
- Works well for tasks where word order matters (e.g., text classification, language modeling).

How Bag of n-grams Works

- Step 1: Split the text into sentences and then into n-grams.
 - Example: "I am learning NLP."
 - Bigrams: ("I am", "am learning", "learning NLP")
- Step 2: Count the frequency of each n-gram across the entire corpus.
- Step 3: Represent the text as a vector of n-gram frequencies.

Example of Bag of n-grams

- Text: "I love programming and I love learning."
- Unigrams: {"I", "love", "programming", "and", "learning"}
- Bigrams: {"I love", "love programming", "programming and", "and I", "I love", "love learning"}
- **Trigrams**: {"I love programming", "love programming and", "programming and I", "and I love", "I love learning"}

Advantages of Bag of n-grams

- Captures Context: Unlike BoW, it can model relationships between consecutive words.
- Improved Performance: Especially useful for tasks like language modeling, sentiment analysis, and text classification.
- Simplicity: Easy to implement and understand.

Challenges of Bag of n-grams

- Increased Dimensionality: The number of possible n-grams can grow exponentially with large corpora.
- Sparsity: Many n-grams may not appear frequently enough to be useful.
- **Ignoring Word Order**: Although n-grams capture some context, they still do not fully capture complex word relationships beyond n-gram length.

Applications of Bag of n-grams

- Text Classification: Categorizing text into predefined categories.
- Sentiment Analysis: Identifying the sentiment (positive/negative) of a piece of text.
- Machine Translation: Translating sentences based on learned n-grams.
- Speech Recognition: Modeling language patterns in spoken language.



Applications of Bag of n-grams

- Text Classification: Categorizing text into predefined categories.
- Sentiment Analysis: Identifying the sentiment (positive/negative) of a piece of text.
- Machine Translation: Translating sentences based on learned n-grams.
- Speech Recognition: Modeling language patterns in spoken language.



Word Embeddings



Limitations of Bag of Words & TF-IDF

Large Vector Sizes:

- Example: Vocabulary of 200,000 words, resulting in 100,000-size vectors.
- High Memory & Compute Usage.

Sparsity:

Most values are zeros, leading to inefficient representations.

• Example:

 Sentences like "I need help" vs. "I need assistance" might not have similar vector representations in TF-IDF or Bag of Words, despite being semantically similar.

What is Word Embedding?

- Word Embedding: A technique to represent words in dense vectors with a lower dimensionality.
- Key Advantage:
 - Similar words (like "good" and "great") have similar vectors.
 - Word embeddings have dense representations (fewer zeros).
- **Vector Size**: Typically around **300 dimensions**, which is much smaller than traditional models.

How Word Embedding Works

- Similar Words = Similar Vectors:
 - Example: "Good" and "Great" will have vectors like:
 - a. "Good": [3.1, 3.1, 4.4, 4.2]
 - b. "Great": [3.2, 3.1, 4.3, 4.1]
- Efficient: Smaller vectors, fewer zeros, and captures the meaning of words.

Word Embedding Techniques

- Popular Techniques:
 - Word2Vec
 - GloVe
 - FastText
- Word2Vec:
 - Uses Continuous Bag of Words (CBOW) and Skip-gram.
- Transformer-based Models:
 - BERT, GPT: Recent advancements that improve text representation further.

The Power of Word2Vec

- Word2Vec Example:
 - Arithmetic with Words:
 - a. King Man + Woman = Queen
 - b. Word2Vec can perform arithmetic with word vectors to capture relationships.
- Visualizing Word Relationships:
 - King and Queen share common attributes like authority and power, but differ by gender.

Dataset Variations in Word Embeddings

- Word2Vec: Train on different corpora to get domain-specific embeddings.
 - Example: Google News, Amazon Reviews, etc.
- GloVe: Can be trained on specific datasets like Twitter or Wikipedia.
 - Twitter dataset understands slang and abbreviations better.
- **BERT**: Can be fine-tuned on specific domains (e.g., BioBERT for biomedical data, FinBERT for financial data).

Other Advanced Techniques

- Elmo: Embeddings based on LSTM (Long Short-Term Memory).
- Transformer Models:
 - BERT, Albert, Roberta, etc.
 - Trained on specific datasets for better domain understanding.



Converting Text to Vectors

- Goal: Convert individual words, sentences, or entire documents into vectors.
 - Sentence Embedding: Converting a sentence into a single vector.
 - Document Embedding: Converting a document or article into a vector.
- Why: Machine learning models need numbers, and word embeddings allow text to be represented in a numerical format that captures meaning.

