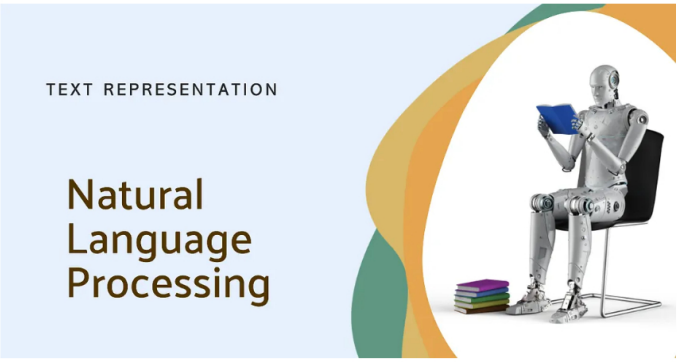
From Traditional to Modern: A Comprehensive Guide to Text Representation Techniques in NLP

Susovan Dey Follow 6 min read · Apr 27, 2023

Natural Language Processing (NLP) is a rapidly growing field that focuses on enabling machines to understand and process human language. Text representation is a crucial aspect of NLP that involves converting raw text data into machine-readable form.

In this article, we will explore the different text representation techniques, starting from traditional approaches such as bag-of-words and n-grams to modern techniques like word embeddings.

By the end of this article, you will have a fair understanding of the different text representation techniques along with their strength and weaknesses.



Topics to cover:

1. Bag of words
2. N-gram
3. TF-IDF
4. Word embedding
5. Sentence embedding
6. Document embedding

. . .

Bag of words (BoW):

This is the simplest way to convert unstructured text data into a structured numeric format that can be processed by machine learning algorithms. Each word in the text is considered a feature, and the number of times a particular word appears in the text is used to represent the importance of that word in the text. Disregarding grammar and word order but keeping track of the frequency of each word.

Example :

Example :

Let us consider 3 sentences :

1. The cat in the hat
2. The dog in the house
3. The Bird in the Sky

Text	dog	cat	bird	in	house	sky	the	hat
The cat in the hat	0	1	0	1	0	0	2	1
The dog in the house	1	0	0	1	1	0	2	0
The bird in the sky	0	0	1	1	0	1	2	0

Below given code displays the above result:

```
from sklearn.feature_extraction.text import CountVectorizer
# Sample sentences
sentences = ["The cat in the hat",
"The dog in the house", "The bird in the sky"]
# Create a CountVectorizer object
vectorizer = CountVectorizer()
# Use the fit_transform method to transform the sentences into a bag of words
bow = vectorizer.fit_transform(sentences)
# Print the vocabulary (features) of the bag of words
print(vectorizer.get_feature_names())
# Print the bag of words
print(bow.toarray())
```

. . .

N-gram:

An N-gram is a traditional text representation technique that involves breaking down the text into contiguous sequences of n-words. A **uni-gram** gives all the words in a sentence. A **Bi-gram** gives sets of two consecutive words and similarly, a **Tri-gram** gives sets of consecutive 3 words, and so on.

Top highlight

Example: *The dog in the house*

Uni-gram: “The”, “dog”, “in”, “the”, “house”

Bi-gram: “The dog”, “dog in”, “in the”, “the house”

Tri-gram: “The dog in”, “dog in the”, “in the house”

. . .

TF-IDF:

TF-IDF stands for Term Frequency-Inverse Document Frequency. This is better than BoW since it interprets the importance of a word in a document. The idea behind TF-IDF is to weigh words based on how often they appear in a document (the term frequency) and how common they are across all documents (the inverse document frequency).

The formula for calculating the TF-IDF score of a word in a document is:

$$\textbf{TF-IDF} = \text{Term frequency in document} \times \log\left(\frac{\text{Total number of documents}}{\text{Number of documents containing the term}}\right)$$

Text	bird	cat	flying	in	jumped	roared	sky	the	tiger	white
The cat jumped	0	0.5844	0	0	0.5844	0	0	0.3452	0	0
The white tiger roared	0	0	0	0	0	0.5464	0	0.3227	0.5464	0.5464
Bird flying in the sky	0.5046	0	0.5046	0.3838	0	0	0.5046	0.2980	0	0

Below given code displays the above result:

```
from sklearn.feature_extraction.text import TfidfVectorizer
# Example documents
docs = ["The cat jumped",
        "The white tiger roared",
        "Bird flying in the sky"]
# Create a TfidfVectorizer object
vectorizer = TfidfVectorizer()
# Use the fit_transform method to transform the documents into a TF-IDF matrix
tfidf = vectorizer.fit_transform(docs)
# Print the vocabulary (features) of the TF-IDF matrix
print(vectorizer.get_feature_names())
# Print the TF-IDF matrix
print(tfidf.toarray())
```

. . .

Word embedding:

Word embedding represents each word as a dense vector of real numbers, such that the similar or closely related words are nearer to each other in the vector space. This is achieved by training a neural network model on a large corpus of text, where each word is represented as a unique input and the network learns to predict the surrounding words in the text. The semantic meaning of the word is captured using this. The dimension of these words can range from a few hundred (Glove, Wod2vec)to thousands (Language models).

Below given is the code :

```
from gensim.models import Word2Vec

# Define the corpus (list of sentences)
corpus = ["The cat jumped",
        "The white tiger roared",
        "Bird flying in the sky"]
corpus=[sent.split(" ") for sent in corpus]
# Train the Word2Vec model on the corpus
model = Word2Vec(corpus, size=50, window=5, min_count=1, workers=2)

# Get the vector representation of a word
vector = model.wv["cat"]
# Get the top-N most similar words to a given word
similar_words = model.wv.most_similar("cat", topn=5)
```

```
In [12]: model.wv["cat"]
Out[12]:
array([-0.00336673, -0.00485849,  0.00374757,  0.0056207 ,  0.00660873,
       -0.00343384, -0.00605804,  0.00362263,  0.00145669, -0.00143821,
       -0.006509  ,  0.00906889, -0.00309272, -0.00014993, -0.00434327,
        0.00044223,  0.00602196,  0.00677635,  0.00373609, -0.00151189,
        0.00617319, -0.00421809, -0.00789614, -0.00071401,  0.00340425,
       -0.00876774, -0.00124566,  0.00885817,  0.00105338,  0.00504051,
       -0.00410418,  0.0034758 ,  0.00379805,  0.00112287, -0.006118  ,
        0.00111341,  0.00970529, -0.00811307,  0.00604032, -0.00573371,
        0.00565568, -0.00209512,  0.00414546, -0.00195942, -0.00651728,
        0.00585019,  0.00229643,  0.00985541,  0.0095931 , -0.00381472],
      dtype=float32)
```

> *Refer document for details : https://jalammar.github.io/illustrated-word2vec/*

. . .

Sentence Embedding:

It is similar to that of word embedding, the only difference is in place of a

It is similar to that of word embedding, the only difference is in place of a word, a sentence is represented as a numerical vector in a high-dimensional space. The goal of sentence embedding is to capture the meaning and semantic relationships between words in a sentence, as well as the context in which the sentence is used.

Below given is the code:

```
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
# Define a list of sentences to be embedded
sentences = ["The cat jumped",
        "The white tiger roared",
        "Bird flying in the sky"]
# Convert the sentences to TaggedDocuments
tagged_data = [TaggedDocument(words=sentence.split(), tags=[str(i)]) for i, sent
# Train a Doc2Vec model on the TaggedDocuments
model = Doc2Vec(tagged_data, vector_size=50, min_count=1, epochs=10)
# Get the embedding for the first sentence
embedding = model.infer_vector("The white tiger roared".split())
# Print the resulting embedding
print(embedding)
```

```
In [20]: print(embedding)
[-4.0764785e-03 -8.8679567e-03 -4.5506926e-03 -7.6859985e-03
 -8.3008297e-03  8.6368062e-03 -7.6332176e-03  2.6817858e-04
 -9.1235163e-03  6.9689350e-03  2.9975823e-03  1.6072097e-03
  9.3724513e-03 -4.2577544e-03 -1.8217788e-03  8.2548093e-03
 -9.1526285e-03 -5.4314965e-03 -4.3570669e-03  1.4407694e-03
  4.9085249e-03  5.4269154e-03 -6.5111271e-03  5.2013310e-03
  2.3442961e-03  7.5654141e-03  3.9288271e-03 -4.7466788e-03
 -9.9232113e-03 -3.9864173e-03 -9.5671192e-03  6.1138640e-03
  9.7588692e-03  4.0948028e-03  2.8471416e-03  7.5217204e-06
  9.9995798e-03  3.5529889e-03 -3.4659787e-04  6.5274914e-03
  6.4515667e-03 -7.1320734e-03 -3.2238946e-03  1.2980208e-03
  2.8378714e-03  8.7456107e-03  8.8058459e-03 -7.9882704e-04
 -8.3044851e-03  7.7766660e-03]
```

. . .

Document Embedding:

Document embedding refers to the process of representing an entire document, such as a paragraph, article, or book, as a single vector. It captures not only the meaning and context of individual sentences but also the relationships and coherence between sentences within the document. The code used is the same as sentence embedding but has multiple sentences within one document. Below given is the code :

```
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
# Define a list of documents to be embedded
documents = ["This is the first document. It has multiple sentences.",
             "This is the second document. It is shorter than the first.",
             "This is the third document. It is longer than the first two and ha
# Convert the documents to TaggedDocuments
tagged_data = [TaggedDocument(words=document.split(), tags=[str(i)]) for i, docu
# Train a Doc2Vec model on the TaggedDocuments
model = Doc2Vec(tagged_data, vector_size=50, min_count=1, epochs=10)
# Get the embedding for the first document
embedding = model.docvecs[0]
# Print the resulting embedding
print(embedding)
```

```
...: print(embedding)
[ 0.00913262  0.00859622  0.00951773 -0.00606533 -0.00949697  0.00165433
 -0.00691366 -0.00613473 -0.00382259  0.0068699  -0.00508502  0.00675393
  0.00913705  0.00340099  0.00026987 -0.00096869  0.00797543 -0.00048496
  0.0030355  -0.00912543 -0.0015756   0.00799678  0.00676449 -0.00406684
  0.00258873  0.00309777 -0.00219947  0.00910104  0.00796557 -0.00378933
  0.00948659 -0.0021706  -0.00565562  0.00954839  0.00737489 -0.00876492
  0.00958963 -0.00297451 -0.00494447  0.00159239  0.00876829 -0.00754943
  0.00664403 -0.00615669  0.00758254  0.00901006  0.00381718 -0.00624672
  0.00849412  0.00663318]
```

In this blog post, we discussed several techniques used for text representation, including Bag of Words, N-grams, TF-IDF, Word Embedding, Sentence Embedding, and Document Embedding. Each technique has its own advantages and disadvantages, and the task at hand determines which is best. Choosing optimal performance in NLP depends on selecting the appropriate technique for a given task.

Thank you for taking the time to read the content.Clap 👏 if you have enjoyed the content.

Follow me, Susovan Dey, for stories on different implementations of AI and NLP.

NLP | Bag Of Words | Word Embeddings | Sentence Embedding | Text Representation

16

Written by Susovan Dey

25 followers · 20 following

Data scientist with a passion for automating tasks & analyzing data to uncover insights. Love exploring new technologies & expanding my expertise.Hi, Impact

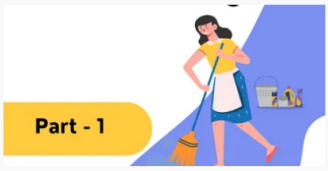
Follow

No responses yet

Abdullh Al Masum

What are your thoughts?

More from Susovan Dey

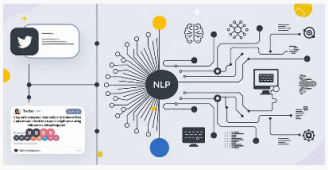


Susovan Dey

Text Cleaning: The Secret Weapon for Smarter NLP Models

Natural Language Processing (NLP) has become a critical component in every sector...

Jul 7, 2023 · 12



Susovan Dey

From Tweets to Truth: Training a Model to Classify Disaster-Relate...

Twitter has become a major go-to platform for real-time updates, especially during time...

Aug 19, 2025 · 1

Susovan Dey

Automate your Instagram Posts Using Python

Want to automate your Instagram posts? Then you have come to the right place, my...

Jun 6, 2021 · 27

Susovan Dey

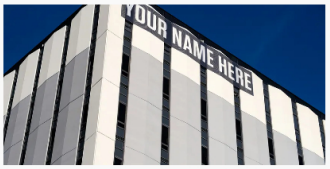
Beyond Keywords: Semantic Search for Smarter Information...

This article explores semantic search, highlighting its superiority over keyword-...

Oct 25, 2023

See all from Susovan Dey

Recommended from Medium

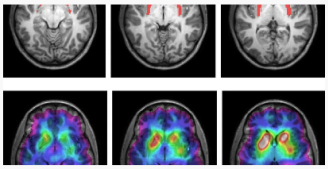


DamenC

Fine-Tuning BERT for Named Entity Recognition: A Step-by-Step Guide

Named Entity Recognition (NER) is a fundamental task in Natural Language...

Oct 3, 2025

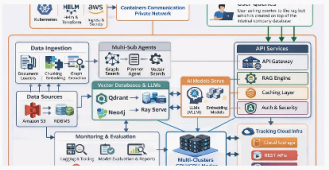


In Write A Catalyst by Dr. Patricia Schmidt

As a Neuroscientist, I Quit These 5 Morning Habits That Destroy You...

Most people do #1 within 10 minutes of waking (and it sabotages your entire day)

Jan 15 13.4K 240



In Level Up Coding by Fareed Khan

Building a Scalable, Production-Grade Agentic RAG Pipeline

Autoscaling, Evaluation, AI Compute Workflows and more

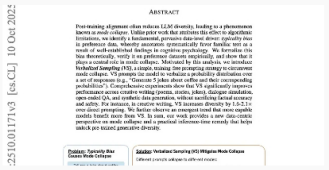
Jan 1 1.2K 13

Will Lockett

The AI Bubble Is About To Burst, But The Next Bubble Is Already...

Techbros are preparing their latest bandwagon.

Sep 15, 2025 21K 903

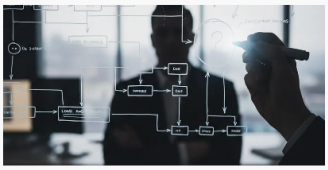


In Generative AI by Adham Khaled

Stanford Just Killed Prompt Engineering With 8 Words (And I...

ChatGPT keeps giving you the same boring response? This new technique unlocks 2×...

Oct 20, 2025 23K 588



In Beyond Localhost by The Speedcraft Lab

I Thought I Knew System Design Until I Met a Google L7 Interviewer

A single whiteboard question revealed the gap between knowing patterns and actually...

Dec 22, 2025 4.4K 81

See more recommendations