

Understanding TF-IDF in NLP: A Comprehensive Guide



Pradeep · Follow

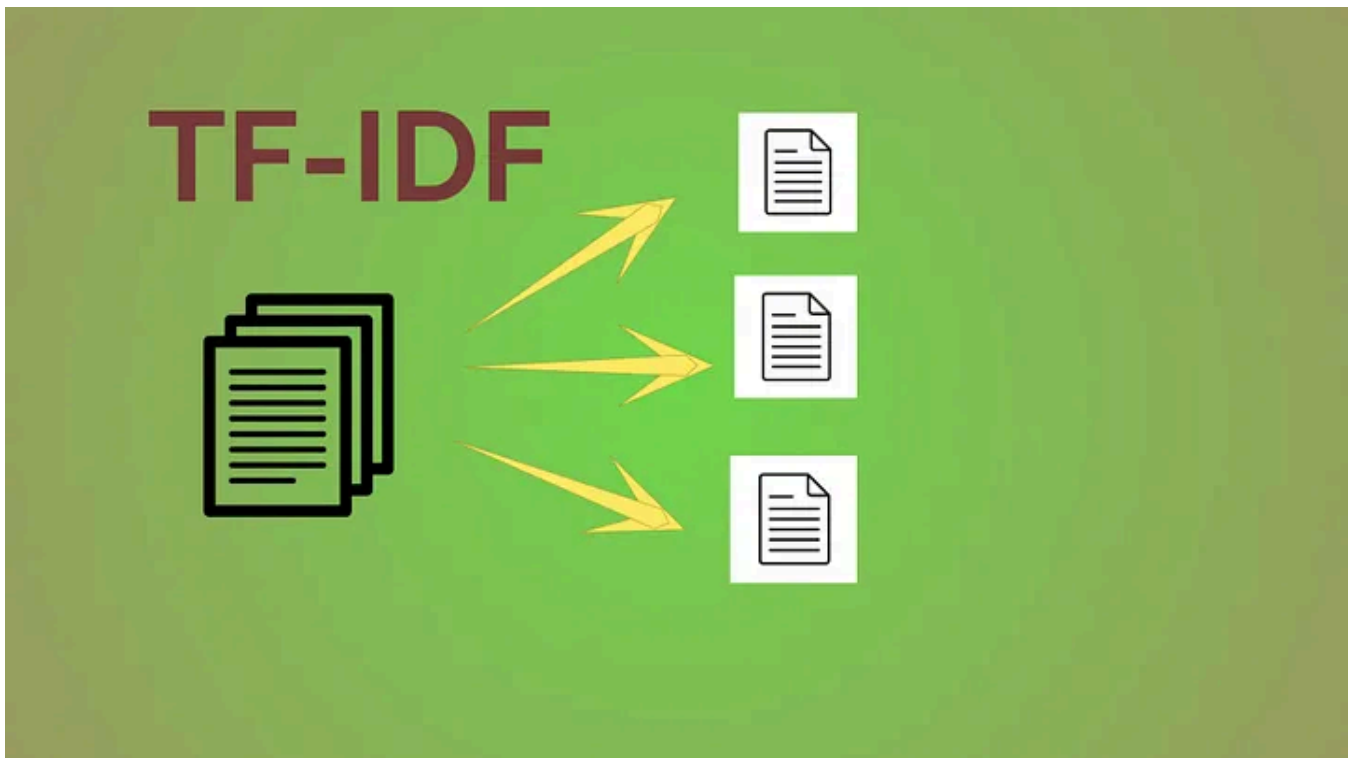
8 min read · Mar 21, 2023



Listen



Share



Natural Language Processing (NLP) is an area of computer science that focuses on the interaction between human language and computers. One of the fundamental tasks of NLP is to extract relevant information from large volumes of unstructured data. In this article, we will explore one of the most popular techniques used in NLP called TF-IDF.

What is TF-IDF?

TF-IDF is a numerical statistic that reflects the importance of a word in a document. It is commonly used in NLP to represent the relevance of a term to a document or a

corpus of documents. The TF-IDF algorithm takes into account two main factors: the frequency of a word in a document (TF) and the frequency of the word across all documents in the corpus (IDF).

The term frequency (TF) is a measure of how frequently a term appears in a document. It is calculated by dividing the number of times a term appears in a document by the total number of words in the document. The resulting value is a number between 0 and 1.

The inverse document frequency (IDF) is a measure of how important a term is across all documents in the corpus. It is calculated by taking the logarithm of the total number of documents in the corpus divided by the number of documents in which the term appears. The resulting value is a number greater than or equal to 0.

The TF-IDF score is calculated by multiplying the term frequency and the inverse document frequency. The higher the TF-IDF score, the more important the term is in the document.

$$\text{TF-IDF} = \text{TF} * \text{IDF}$$

$$\text{TF-IDF} = \text{TF} * \log(N/\text{DF})$$

Where:

- . TF is the term frequency of a word in a document
- . N is the total number of documents in the corpus
- . DF is the document frequency of a word in the corpus (i.e., the number of documents that contain the word)

How does TF-IDF work?

TF-IDF is a commonly used technique in Natural Language Processing (NLP) to evaluate the importance of a word in a document or corpus. It works by assigning weights to words based on their frequency and rarity. Let's walk through an example to better understand how TF-IDF works. Suppose we have a corpus of five documents

Doc1: The quick brown fox jumps over the lazy dog

Doc2: The lazy dog likes to sleep all day

Doc3: The brown fox prefers to eat cheese

Doc4: The red fox jumps over the brown fox

Doc5: The brown dog chases the fox

Now, let's say we want to calculate the TF-IDF scores for the word "fox" in each of these documents.

Step 1: Calculate the term frequency (TF)

The term frequency (TF) is the number of times the word appears in the document.

We can calculate the TF for the word "fox" in each document as follows:

TF = (Number of times word appears in the document) / (Total number of words in the document)

Doc1: 1 / 9

Doc2: 0 / 8

Doc3: 1 / 7

Doc4: 2 / 8

Doc5: 1 / 6

Step 2: Calculate the document frequency (DF):

Open in app ↗

Sign up

Sign in



Search



DF = 4 (Doc1, Doc3, Doc4 and Doc5)

Step 3: Calculate the inverse document frequency (IDF):

The inverse document frequency (IDF) is a measure of how rare the word is across the corpus. It is calculated as the logarithm of the total number of documents in the corpus divided by the document frequency. In our case, we have:

IDF = $\log(5/4)$ = 0.2231

Step 4: Calculate the TF-IDF score:

The TF-IDF score for the word "fox" in each document can now be calculated using

the following formula:

$$\text{TF-IDF} = \text{TF} * \text{IDF}$$

$$\text{Doc1: } 1/9 * 0.2231 = 0.0247$$

$$\text{Doc2: } 0/8 * 0.2231 = 0$$

$$\text{Doc3: } 1/7 * 0.2231 = 0.0318$$

$$\text{Doc4: } 2/8 * 0.2231 = 0.0557$$

$$\text{Doc5: } 1/6 * 0.2231 = 0.0372$$

Therefore, the TF-IDF score for the word “fox” is highest in Doc4 indicating that this word is relatively important in this document compared to the rest of the corpus. On the other hand, the TF-IDF score is zero in Doc2, indicating that the word “fox” is not relevant in this document.

Code Implementation

Here's how you can implement TF-IDF in Python using the scikit-learn. The algorithm works as follows:

1. Preprocessing: The text data is preprocessed by removing stop words, punctuation, and other non-alphanumeric characters.
2. Tokenization: The text is tokenized into individual words.
3. Instantiate TfidfVectorizer and fit the corpus
4. Transform that corpus to get the representation

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import re
from sklearn.feature_extraction.text import TfidfVectorizer

# Sample corpus of documents
corpus = ['The quick brown fox jumps over the lazy dog.',
          'The lazy dog likes to sleep all day.',
          'The brown fox prefers to eat cheese.',
          'The red fox jumps over the brown fox.',
          'The brown dog chases the fox'
        ]

# Define a function to preprocess the text
```

```
def preprocess_text(text):
    # Remove punctuation and other non-alphanumeric characters
    text = re.sub('[^a-zA-Z]', ' ', text)
    # Tokenize the text into words
    words = word_tokenize(text.lower())
    # Remove stop words
    words = [word for word in words if word not in stopwords.words('english')]
    # Join the words back into a string
    return ' '.join(words)

# Preprocess the corpus
corpus = [preprocess_text(doc) for doc in corpus]
print('Corpus: \n{}'.format(corpus))

# Create a TfidfVectorizer object and fit it to the preprocessed corpus
vectorizer = TfidfVectorizer()
vectorizer.fit(corpus)

# Transform the preprocessed corpus into a TF-IDF matrix
tf_idf_matrix = vectorizer.transform(corpus)

# Get list of feature names that correspond to the columns in the TF-IDF matrix
print("Feature Names:\n", vectorizer.get_feature_names_out())

# Print the resulting matrix
print("TF-IDF Matrix:\n", tf_idf_matrix.toarray())
```

Output:

```
Corpus:
['quick brown fox jumps lazy dog', 'lazy dog likes sleep day', 'brown fox prefe
Feature Names:
['brown', 'chases', 'cheese', 'day', 'dog', 'eat', 'fox', 'jumps', 'lazy', 'li
TF-IDF Matrix:
[[0.30620672 0.          0.          0.          0.36399815 0.
  0.30620672 0.43850426 0.43850426 0.          0.          0.54351473
  0.          0.          ]
 [0.          0.          0.          0.49389914 0.33077001 0.
  0.          0.          0.39847472 0.49389914 0.          0.
  0.          0.49389914]
 [0.29550385 0.          0.52451722 0.          0.          0.52451722
  0.29550385 0.          0.          0.          0.52451722 0.
  0.          0.          ]
 [0.31309104 0.          0.          0.          0.          0.
  0.62618207 0.44836297 0.          0.          0.          0.
  0.55573434 0.          ]
 [0.39032474 0.69282362 0.          0.          0.46399205 0.
```

```
0.39032474 0.      0.      0.      0.      0.
0.      0.      ]]
```

Advantages of TF-IDF

Some of the advantages of using TF-IDF include:

1. Measures relevance: TF-IDF measures the importance of a term in a document, based on the frequency of the term in the document and the inverse document frequency (IDF) of the term across the entire corpus. This helps to identify which terms are most relevant to a particular document.
2. Handles large text corpora: TF-IDF is scalable and can be used with large text corpora, making it suitable for processing and analyzing large amounts of text data.
3. Handles stop words: TF-IDF automatically down-weights common words that occur frequently in the text corpus (stop words) that do not carry much meaning or importance, making it a more accurate measure of term importance.
4. Can be used for various applications: TF-IDF can be used for various natural language processing tasks, such as text classification, information retrieval, and document clustering.
5. Interpretable: The scores generated by TF-IDF are easy to interpret and understand, as they represent the importance of a term in a document relative to its importance across the entire corpus.
6. Works well with different languages: TF-IDF can be used with different languages and character encodings, making it a versatile technique for processing multilingual text data.

Limitations of TF-IDF

Here are a few limitations of TF-IDF:

1. Ignores the context: TF-IDF only considers the frequency of each term in a document, and does not take into account the context in which the term appears. This can lead to incorrect interpretations of the meaning of the document.

2. Assumes independence: TF-IDF assumes that the terms in a document are independent of each other. However, this is often not the case in natural language, where words are often related to each other in complex ways.
3. Vocabulary size: The vocabulary size can become very large when working with large datasets, which can lead to high-dimensional feature spaces and difficulty in interpreting the results.
4. No concept of word order: TF-IDF treats all words as equally important, regardless of their order or position in the document. This can be problematic for certain applications, such as sentiment analysis, where word order can be crucial for determining the sentiment of a document.
5. Limited to term frequency: TF-IDF only considers the frequency of each term in a document and does not take into account other important features, such as the length of the document or the position of the term within the document.
6. Sensitivity to stopwords: TF-IDF can be sensitive to stop words, which are common words that do not carry much meaning, but appear frequently in documents. Removing stop words from the document can help to address this issue.

Overall, while TF-IDF is a useful technique for text analysis, it is important to keep in mind its limitations and to use it in conjunction with other techniques to get a more complete picture of the meaning of a document or a corpus

Applications of TF-IDF

Here are some of the main applications of TF-IDF:

1. Search engines: TF-IDF is used in search engines to rank documents based on their relevance to a query. The TF-IDF score of a document is used to measure how well the document matches the search query.
2. Text classification: TF-IDF is used in text classification to identify the most important features in a document. The TF-IDF score of each term in the document is used to measure its relevance to the class.
3. Information extraction: TF-IDF is used in information extraction to identify the most important entities and concepts in a document. The TF-IDF score of each term is used to measure its importance in the document.

4. **Keyword extraction:** TF-IDF is used in keyword extraction to identify the most important keywords in a document. The TF-IDF score of each term is used to measure its importance in the document.
5. **Recommender systems:** TF-IDF is used in recommender systems to recommend items to users based on their preferences. The TF-IDF score of each item is used to measure its relevance to the user's preferences.
6. **Sentiment analysis:** TF-IDF is used in sentiment analysis to identify the most important words in a document that contribute to the sentiment. The TF-IDF score of each word is used to measure its importance in the document.

Overall, TF-IDF is a versatile and widely used technique that can be applied to a variety of natural language processing tasks.

Conclusion

TF-IDF is a powerful technique in NLP that enables us to evaluate the importance of words in a document or a corpus. By assigning weights to words based on their frequency and rarity, we can extract meaningful information from unstructured text data. TF-IDF has several applications in NLP, including information retrieval, text classification, and keyword extraction. As NLP continues to evolve, TF-IDF remains a fundamental technique for processing natural language data.

If you enjoyed the article, **please consider clapping** for it using the clap button at the bottom of the page. Feel free to **leave a comment** below sharing your thoughts, feedback, or any additional tips and tricks.

Thank you for reading this post! I hope you found it informative and helpful. If you enjoyed it and would like to see more, be sure to **follow and subscribe me**. I would greatly appreciate it if you could show your *support by clapping for it and leaving a comment*. I look forward to sharing more with you in the future. Thanks again!

NLP

Deep Learning

Artificial Intelligence

Machine Learning

Technology

[Follow](#)


Written by Pradeep

369 Followers

Experienced Data Scientist | Loves to learn and share content regarding Machine learning, AI and Data | Join Medium: <https://tinyurl.com/ym9zsfzr>

More from Pradeep



 Pradeep

Deploy Python Application in AWS Lambda: Step-By-Step Guide

If you're working with AWS Lambda, you're probably already aware of its benefits in terms of cost, scalability, and efficiency. However, if...

🌟 · 8 min read · Mar 17, 2023

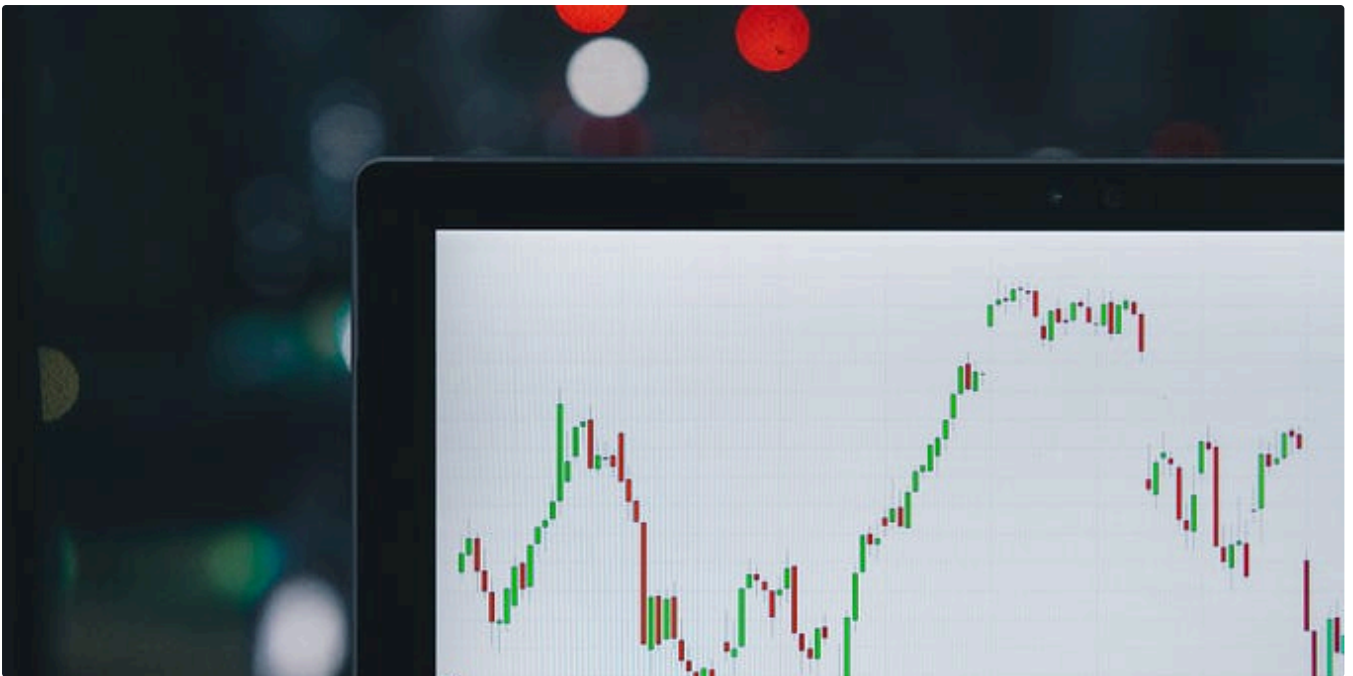


244



1





P Pradeep

Percentage Change in Stock Price Pandas

Calculating percentage change is very helpful in scenarios such as stock price, sales of an article, consumption of the commodities and...

★ · 4 min read · Nov 24, 2022



147



1

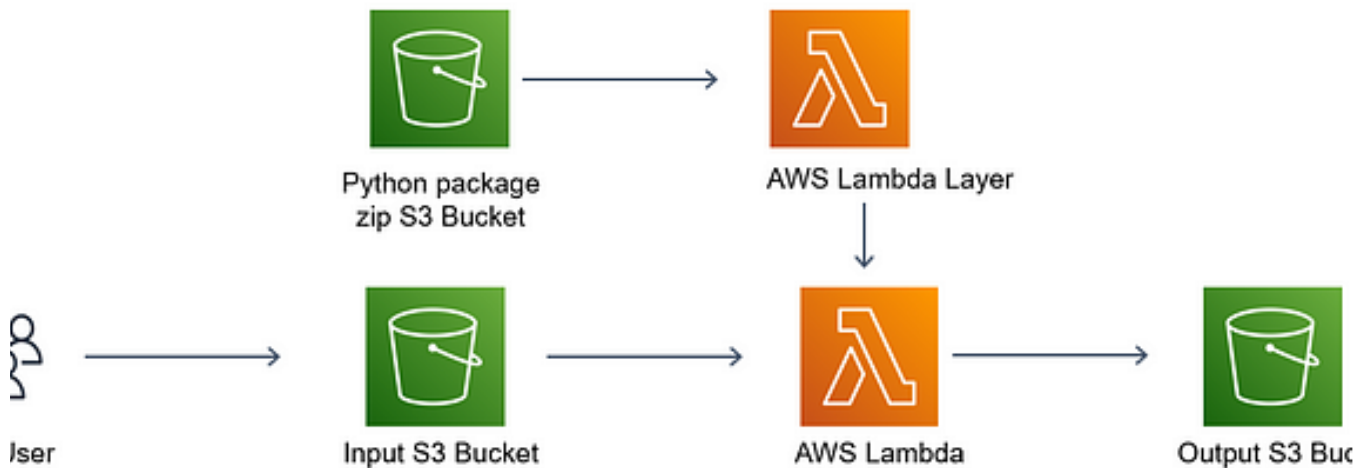


P Pradeep

6 Simple Ways to convert your S3 Bucket into Athena Database

Are you struggling to convert your S3 data into actionable insights? Look no further than Amazon Athena. Athena is a serverless query...

★ · 9 min read · Apr 26, 2023



P Pradeep

Automating Your S3 File Uploads with AWS Lambda: A Simple Guide Part 2

This is 2nd part of the Automating Your S3 File Uploads with AWS Lambda. I highly recommend that you read Part 1 first. Part 1 lays the...

★ · 7 min read · Feb 21, 2023



See all from Pradeep

Recommended from Medium

Sent 2 : good girl
Sent 3 : boy girl good

$$TF = \frac{\text{No of rep of words in a sentence}}{\text{No of words in a sentence}}$$
$$IDF = \log / \underline{\text{No of sentences}}$$



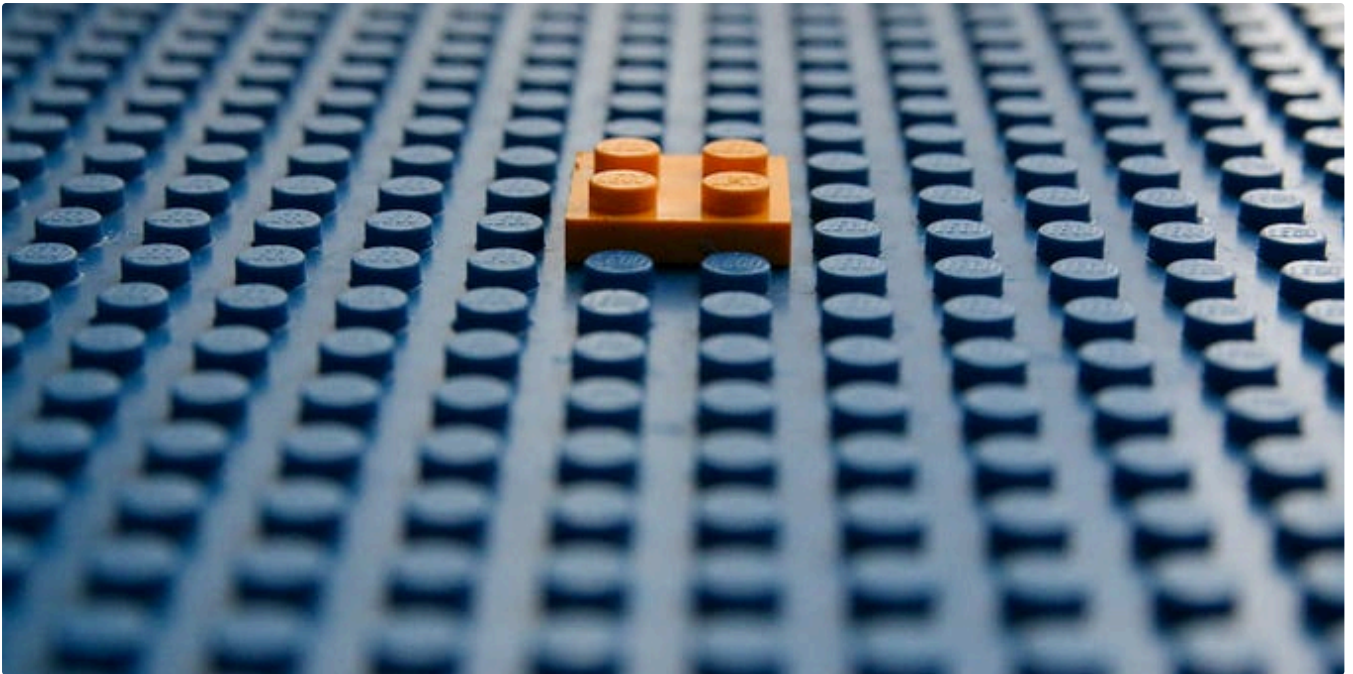
Abhishek Jain

TF-IDF in NLP (Term Frequency Inverse Document Frequency)

In the realm of Natural Language Processing (NLP), TF-IDF (Term Frequency-Inverse Document Frequency) is a powerful technique used to...

3 min read · Feb 4, 2024





Awaldeep Singh

Understanding the Essentials: NLP Text Preprocessing Steps!

Introduction

8 min read · Dec 30, 2023



3



Lists



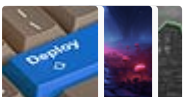
AI Regulation

6 stories · 447 saves



Natural Language Processing

1447 stories · 951 saves



Predictive Modeling w/ Python


20 stories · 1184 saves



ChatGPT prompts

47 stories · 1545 saves



 Eulene

Sentiment Analysis of Amazon Reviews Using Natural Language Processing

11 min read · Feb 8, 2024



22



1



The Google logo, consisting of the word 'Google' in its signature multi-colored font (blue, red, yellow, blue, green, red).

The word 'BERT' in a bold, red, sans-serif font.

 Rayyan Shaikh

Mastering BERT: A Comprehensive Guide from Beginner to Advanced in Natural Language Processing...

Introduction: A Guide to Unlocking BERT: From Beginner to Expert

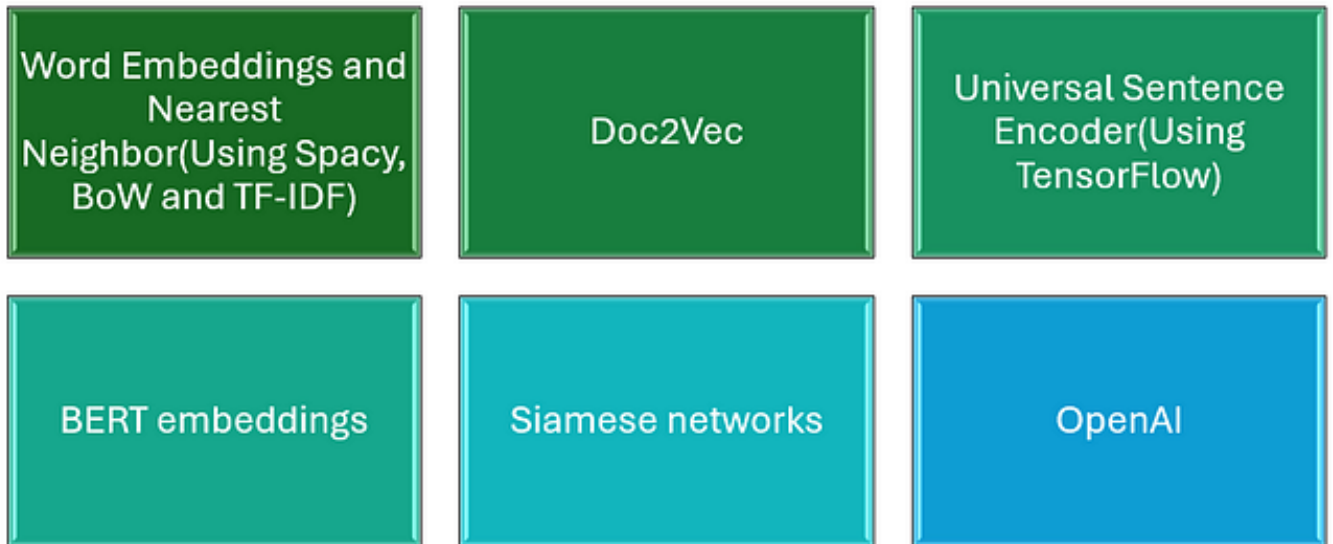
19 min read · Aug 26, 2023



1.9K



15



Aneesha B Soman

Exploring Diverse Techniques for Sentence Similarity

Sentence similarity refers to the degree of similarity or closeness between two sentences in terms of their meaning or semantic content...

11 min read · Mar 2, 2024



14





Miftahul Ulyana Hutabarat

Comparing Text Documents Using TF-IDF and Cosine Similarity in Python

How can we know if two text documents are similar? Humans can see the differences, but how can computers know if two text documents are...

6 min read · Dec 17, 2023



36



1



See more recommendations