TF-IDF Explained And Easy Examples In Python To Get Started

Neri Van Otten · Follow
8 min read · Nov 28, 2022

1

Tf-idf is a way to measure the importance of a word. It is one of the ten most commonly used natural language processing techniques. This comprehensive guide covers tf-idf, why you should use it, and some typical applications. We also cover its advantages, disadvantages, and some tools to implement tf-idf.

The goal of this article is to get you to understand the technique so you can start using it immediately in your projects.

Tf-idf looks for valuable words in a document and a large corpus of documents.

What is TF IDF?

Finding essential words in a text is one of the most common use cases in information retrieval and text mining, and a common way of doing this is using tf-idf. Tf-idf stands for term frequency-inverse document frequency. This measure assesses a word's significance within a collection of documents. Therefore, a unique word that only appears a few times in a set of documents will be more critical and assigned a higher weight than frequently occurring words. Common English words like "a," "it," and "this" will often appear and, therefore, have a lower tf-idf weight.

> TFIDF is a simple measure of a word's importance within a set of documents.

Search engines frequently use variations of the tf-idf weighting schemes as their leading scoring and ranking tool when determining how relevant a document is to a user query.

Tf-idf is also commonly used to filter out stop-words effectively, and this has various use cases in text classification and summarization.

Term Frequency

Let's say we want to order a collection of English text documents based on which one is more pertinent to the question "the red car." We start by simply removing any documents that don't contain all three words-"the," "red," and "car." This leaves many documents. We could count the number of times each term appears in each document to separate them further. The frequency at which a word appears in a document is referred to as "term frequency." (Adjustments are frequently made when the length of documents varies significantly.)

> The weight of a term that occurs in a document is simply proportional to the term's frequency.

Inverse Document Frequency

The term "the" is so widely used that the word "frequency" will often incorrectly emphasise documents that happen to use it more often. In the meantime, the more significant terms "red" and "car." will be undervalued. Moreover, unlike the less popular words "red" and "car," the word "the" is not a good keyword. It can't be used to distinguish between relevant and irrelevant documents. As a result, an inverse document frequency factor is used. This increases the weight of infrequent terms and decreases the importance of frequently occurring words in the document set.

Term-specificity, called Inverse Document Frequency (IDF), is an essential measure of a word's importance.

> The specificity of a term can be quantified as an inverse function of the number of documents in which it occurs.

What we get when we put them together: TF-IDF

Then tf-idf is calculated as follows:

> TF-IDF = term frequency * inverse document frequency

The tf-idf weights have a tendency to filter out common terms and give a high score to unique words.

Check out Wikipedia for a more mathematical definition and justification.

Why is TF-IDF used in machine learning?

The most significant problem faced by natural language processing is that machine learning models tend to only deal with numerical values. This is a problem, as numbers can't just represent natural language, or they would lose meaning. Therefore, we must vectorize the text to convert it into numbers. This is a crucial step in machine learning, and the outcomes of various vectorization algorithms will vary greatly. Hence, choosing one that produces the desired product for your problem is vital.

The tf-idf score converts words into numbers that can be fed to algorithms like Naive Bayes and Support Vector Machines, significantly improving the results of more straightforward techniques like word counts.

Does this work? In its simplest form, a word vector represents a document as a list of numbers. A number is used to represent each possible word in the text. By taking a document's text and turning it into one of these vectors, the text's content is somehow represented by the vectors' numbers. Then, with the help of tf-idf, we can quantify the relevance of each word in a document by associating it with a number. As a result, similar vectors will exist in documents that contain identical, pertinent words, which is what a machine learning algorithm seeks to do.

What are the Applications of TF-IDF

Finding relevant words in documents is helpful in many ways.

Information retrieval

Tf-idf is critical in search and ranking applications. Tf-idf provides results that are most pertinent to your search. Consider your search engine as someone searching for "the red car." The outcomes will be presented in relevant order. In other words, the most pertinent articles about red cars will be ranked higher because the words "red" and "car" receive a higher score from tf-idf. Due to it's importance, every search engine you have used probably incorporates tf-idf scores into its algorithm.

Tf-idf is most commonly used in information retrieval.

Keyword Extraction

Tf-idf can be used to extract keywords from the text as well. The words that received the highest scores were the most pertinent to the document, making them suitable for use as keywords. This is useful for applications like word cloud formations and quick summaries of large bodies of text.

Keyword extractio n quickly let's you see what a document is about

Advantages and disadvantages of using TF-IDF

Advantages of TF-IDF

The simplicity and ease of use of tf-idf are its most significant benefits. As a result, it is easy to compute, inexpensive to run, and a clear starting point for similarity calculations.

Disadvantages of using TF-IDF

It should be noted that tf-idf cannot assist in carrying semantic meaning. It weighs the words and considers them when determining their importance, but it cannot always infer the context of the phrase or determine their significance in that way.

Tf-idf disregards word order, so compound nouns like "New York" will not be regarded as a "single unit." This applies to situations where the order makes a significant difference, such as negation with "friendly" vs "not friendly."

"New_York" or "not-friendly" are two ways to treat the phrase as a single unit in both situations using dashes and underscores.

Because tf-idf can experience the curse of dimensionality, it can also experience memory inefficiency. The vocabulary size is equal to the length of the tf-idf vectors. This might not be a problem in some classification contexts, but in others, like clustering, it can become cumbersome as the number of documents rises. Therefore, it might be necessary to look into alternatives (BERT, Word2Vec).

What tools are used to implement TFIDF?

Scikit Learn

```python
from sklearn.feature_extraction.text import TfidfVectorizer

data = ["I love natural language processing",
        "Creating word vectors",
        "Is my jam!"]

# fit and tranform your data
vectorizer = TfidfVectorizer()
vectorized_data = vectorizer.fit_transform(data)o
```

Using Python, it's straightforward to transform your data to a tf-idf vector in just a few lines of code.

For more details, see the documentation

NLTK

Another lovely Python package is NLTK; it has straightforward implementations of many basic natural language processing tools, including tf-idf. Although they have a tf-idf implementation, you should use the Scikit Learn implementation above. This implementation has been optimised for better memory performance and will be faster on your data.

Spacy

```
# Note: This requires these setup steps:
#   pip install tmtoolkit[recommended]
#   python -m tmtoolkit setup en

from tmtoolkit.bow.bow_stats import tfidf

data = ["I love natural language processing",
        "Creating word vectors",
        "Is my jam!"]

vectorized_data = tfidf(data)
```

Spacy is another great toolkit in Python, with plenty of natural language processing tools. You would need to download the toolkit, but after that, using the implementation is just a single line of code.

For more on this method, see the documentation.

Key takeaways

- Tf-idf is a helpful tool for finding important words in a document or a collection of documents.
- Tf-idf allows text to be turned into numerical vectorizes, which is crucial for many machine learning algorithms that only work with numerical

input. It’s a vital pre-processing step in any natural language processing pipeline.

- The primary use case of tf-idf is in information retrieval and keyword extraction. Information retrieval lets us rank documents according to the relevance of a given search term and is therefore used by search engines to retrieve relevant web pages. Keyword extraction lets us quickly find important words in a large document set.
- The main advantage of tf-idf is its simplicity. It is easy to implement and fast to use. Great to get started with and to give you immediate results.
- The main disadvantage is that it can’t infer context and that it’s hard to determine what a word or phrase is. As a result, terms such as “New York” will be split into two terms “New” and “York” and this is no longer useful for any further analysis.
- Python is a great tool for natural language processing (NLP). The main packages for a tf-idf implementation are Scikit Learn, NLTK and Spacy.

Final Words

Once you have understood and implemented a tf-idf solution, moving on to more complicated vectorization methods can be useful, depending on your used case.

Given the pitfalls of tf-idf, we at Spot Intelligence still use it rather frequently in our pipelines. Accuracy is often not the main concern when processing large volumes of text. Finding and combining large data sets can be process intensive, which is often too slow with some other vectorizers. Especially those that require a lot of memory to train. If you focus on good feature engineering, you can make sure that you capture bigrams and trigrams in your tf-idf algorithm.

Do you use tf-idf in your projects, or do you have another preferred vectorization technique? Let us know in the comments.

Originally published at https://spotintelligence.com on November 28, 2022.

Python Naturallanguageprocessing Data Science Machine Learning

Artificial Intelligence

Written by Neri Van Otten

Follow

50 Followers · 4 Following

Machine learning engineer. 12+ years of experience. Natural Language Processing (NLP) expert. Founder of Spot Intelligence https://spotintelligence.com

No responses yet

What are your thoughts?

Respond