Search...

Understanding TF-IDF (Term Frequency-Inverse Document Frequency)

Last Updated : 07 Feb, 2025

TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical measure used in natural language processing and information retrieval **to evaluate the importance of a word in a document relative to a collection of documents (corpus).**

> *Unlike simple word frequency, TF-IDF balances common and rare words to highlight the most meaningful terms.*

How TF-IDF Works?

TF-IDF combines two components: Term Frequency (TF) and Inverse Document Frequency (IDF).

Term Frequency (TF): Measures how often a word appears in a document. A higher frequency suggests greater importance. If a term appears frequently in a document, it is likely relevant to the document's content. **Formula:**

$$\text{TF}(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

Term Frequency (TF)

Limitations of TF Alone:

- TF does not account for the global importance of a term across the entire corpus.
- Common words like **"the"** or **"and"** may have high TF scores but are not meaningful in distinguishing documents.

Inverse Document Frequency (IDF): Reduces the weight of common words across multiple documents while increasing the weight of rare words. If a term appears in fewer documents, it is more likely to be meaningful and specific.

Formula:

$$\text{IDF}(t, D) = \log \frac{\text{Total number of documents in corpus D}}{\text{Number of documents containing term t}}$$

Inverse Document Frequency (IDF)

- The logarithm is used to dampen the effect of very large or very small values, ensuring the IDF score scales appropriately.
- It also helps balance the impact of terms that appear in extremely few or extremely many documents.

Limitations of IDF Alone:

- IDF does not consider how often a term appears within a specific document.
- A term might be rare across the corpus (high IDF) but irrelevant in a specific document (low TF).

Converting Text into vectors with TF-IDF : Example

To better grasp how TF-IDF works, let's walk through a detailed example. Imagine we have a **corpus** (a collection of documents) with three documents:

1. **Document 1:** "The cat sat on the mat."
2. **Document 2:** "The dog played in the park."
3. **Document 3:** "Cats and dogs are great pets."

Our goal is to calculate the TF-IDF score for specific terms in these documents. Let's focus on the word **"cat"** and see how TF-IDF evaluates its importance.

Step 1: Calculate Term Frequency (TF)

For Document 1:

- The total number of terms in Document 1 is **6** (“the”, “cat”, “sat”, “on”, “the”, “mat”).
- So, TF(cat,Document 1) = 1/6

For Document 2:

- The word **“cat”** does **not appear**.
- So, TF(cat,Document 2)=0.

For Document 3:

- The word **“cat”** appears **1 time** (as “cats”).
- The total number of terms in Document 3 is **6** (“cats”, “and”, “dogs”, “are”, “great”, “pets”).
- So, TF(cat,Document 3)=1/6

  - *In Document 1 and Document 3, the word **“cat”** has the same TF score. This means it appears with the same relative frequency in both documents.*
  - *In Document 2, the TF score is 0 because the word **“cat”** does not appear.*

Step 2: Calculate Inverse Document Frequency (IDF)

- **Total number of documents in the corpus (D):** 3
- **Number of documents containing the term “cat”:** 2 (Document 1 and Document 3).

So, $IDF(cat, D) = log\frac{3}{2} \approx 0.176$

> *The IDF score for **“cat”** is relatively low. This indicates that the word **“cat”** is not very rare in the corpus—it appears in 2 out of 3 documents. If a term appeared in only 1 document, its IDF score would be higher, indicating greater uniqueness.*

The TF-IDF score for **“cat”** is 0.029 in Document 1 and Document 3, and 0 in Document 2 that **reflects both the frequency of the term in the document (TF) and its rarity across the corpus (IDF)**.

The TF-IDF score is the product of TF and IDF:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D)$$

For Document 1:

$$\text{TF-IDF}(\text{cat}, \text{Document 1}, D) = 0.167 \times 0.176 \approx 0.029$$

For Document 2:

$$\text{TF-IDF}(\text{cat}, \text{Document 2}, D) = 0 \times 0.176 = 0$$

For Document 3:

$$\text{TF-IDF}(\text{cat}, \text{Document 3}, D) = 0.167 \times 0.176 \approx 0.029$$

TF-IDF

> *A higher TF-IDF score means the term is more important in that specific document.*

Why is TF-IDF Useful in This Example?

1. Identifying Important Terms: TF-IDF helps us understand that **“cat”** is somewhat important in Document 1 and Document 3 but irrelevant in Document 2.

> *If we were building a search engine, this score would help rank Document 1 and Document 3 higher for a query like **“cat”**.*

2. Filtering Common Words: Words like **“the”** or **“and”** would have high TF scores but very low IDF scores because they appear in almost all documents. Their TF-IDF scores would be close to 0, indicating they are not meaningful.

3. Highlighting Unique Terms: If a term like **“mat”** appeared only in Document 1, it would have a higher IDF score, making its TF-IDF score more significant in

Implementing TF-IDF in Sklearn with Python

In python tf-idf values can be computed using *TfidfVectorizer()* method in *sklearn* module.

Syntax:

> *sklearn.feature_extraction.text.TfidfVectorizer(input)*
>
> ***Parameters:***
>
> - ***input:*** *It refers to parameter document passed, it can be a filename, file or content itself.*
>
> ***Attributes:***
>
> - ***vocabulary_:*** *It returns a dictionary of terms as keys and values as feature indices.*
> - ***idf_:*** *It returns the inverse document frequency vector of the document passed as a parameter.*
>
> ***Returns:***
>
> - ***fit_transform():*** *It returns an array of terms along with tf-idf values.*
> - ***get_feature_names():*** *It returns a list of feature names.*

Step-by-step Approach:

- Import modules.

```
# import required module
from sklearn.feature_extraction.text import TfidfVectorizer
```

- Collect strings from documents and create a corpus having a collection of strings from the documents *d0, d1,* and *d2*.

```
d2 = 'r2j'

# merge documents into a single corpus
string = [d0, d1, d2]
```

- Get tf-idf values from *fit_transform()* method.

```
# create object
tfidf = TfidfVectorizer()

# get tf-df values
result = tfidf.fit_transform(string)
```

- Display idf values of the words present in the corpus.

```
# get idf values
print('\nidf values:')
for ele1, ele2 in zip(tfidf.get_feature_names(), tfidf.idf_):
    print(ele1, ':', ele2)
```

Output:

```
idf values:
for : 1.6931471805599454
geeks : 1.2876820724517808
r2j : 1.6931471805599454
```

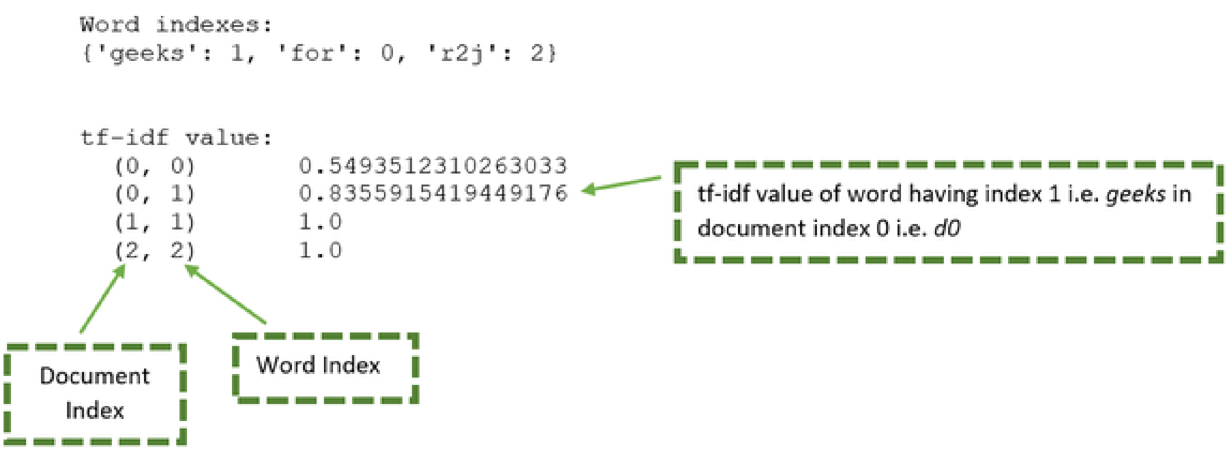
- Display tf-idf values along with indexing.

```
# get indexing
print('\nWord indexes:')
print(tfidf.vocabulary_)

# display tf-idf values
print('\ntf-idf value:')
print(result)

# in matrix form
print('\ntf-idf values in matrix form:')
print(result.toarray())
```

Output:

```
Word indexes:
{'geeks': 1, 'for': 0, 'r2j': 2}

tf-idf value:
  (0, 0)        0.5493512310263033
  (0, 1)        0.8355915419449176
  (1, 1)        1.0
  (2, 2)        1.0

tf-idf values in matrix form:
[[0.54935123 0.83559154 0.        ]
 [0.         1.         0.        ]
 [0.         0.         1.        ]]
```

The *result* variable consists of unique words as well as the tf-if values. It can be elaborated using the below image:



From the above image the below table can be generated:

Document	Word	Document Index	Word Index	tf-idf value
d0	for	0	0	0.549
d0	geeks	0	1	0.8355
d1	geeks	1	1	1.000
d2	r2j	2	2	1.000

Below are some examples which depict how to compute tf-idf values of words from a corpus:

Example 1: Below is the complete program based on the above approach:

```
# import required module
from sklearn.feature_extraction.text import TfidfVectorizer

# assign documents
d0 = 'Geeks for geeks'
d1 = 'Geeks'
d2 = 'r2j'

# merge documents into a single corpus
string = [d0, d1, d2]

# create object
tfidf = TfidfVectorizer()

# get tf-df values
result = tfidf.fit_transform(string)

# get idf values
print('\nidf values:')
for ele1, ele2 in zip(tfidf.get_feature_names(), tfidf.idf_):
    print(ele1, ':', ele2)

# get indexing
print('\nWord indexes:')
print(tfidf.vocabulary_)

# display tf-idf values
print('\ntf-idf value:')
print(result)

# in matrix form
print('\ntf-idf values in matrix form:')
print(result.toarray())
```

Output:

```
idf values:
for : 1.6931471805599454
geeks : 1.2876820724517808
r2j : 1.6931471805599454

Word indexes:
{'geeks': 1, 'for': 0, 'r2j': 2}

tf-idf value:
  (0, 0)        0.5493512310263033
  (0, 1)        0.8355915419449176
  (1, 1)        1.0
  (2, 2)        1.0

tf-idf values in matrix form:
[[0.54935123 0.83559154 0.        ]
 [0.         1.         0.        ]
 [0.         0.         1.        ]]
```

Example 2: Here, tf-idf values are computed from a corpus having unique values.

```python
# import required module
from sklearn.feature_extraction.text import TfidfVectorizer

# assign documents
d0 = 'geek1'
d1 = 'geek2'
d2 = 'geek3'
d3 = 'geek4'

# merge documents into a single corpus
string = [d0, d1, d2, d3]

# create object
tfidf = TfidfVectorizer()

# get tf-df values
result = tfidf.fit_transform(string)

# get indexing
print('\nWord indexes:')
print(tfidf.vocabulary_)

# display tf-idf values
print('\ntf-idf values:')
print(result)
```

Output:

```
Word indexes:
{'geek1': 0, 'geek2': 1, 'geek3': 2, 'geek4': 3}

tf-idf values:
  (0, 0)        1.0
  (1, 1)        1.0
  (2, 2)        1.0
  (3, 3)        1.0
```

Example 3: In this program, tf-idf values are computed from a corpus having similar documents.

```
# import required module
from sklearn.feature_extraction.text import TfidfVectorizer

# assign documents
d0 = 'Geeks for geeks!'
d1 = 'Geeks for geeks!'


# merge documents into a single corpus
string = [d0, d1]

# create object
tfidf = TfidfVectorizer()

# get tf-df values
result = tfidf.fit_transform(string)

# get indexing
print('\nWord indexes:')
print(tfidf.vocabulary_)

# display tf-idf values
print('\ntf-idf values:')
print(result)
```

Output:

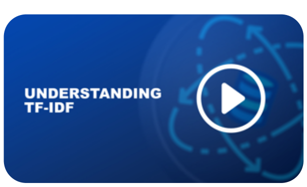
```
Word indexes:
{'geeks': 1, 'for': 0}

tf-idf values:
  (0, 0)        0.4472135954999579
  (0, 1)        0.8944271909999159
```

Example 4: Below is the program in which we try to calculate tf-idf value of a single word *geeks* is repeated multiple times in multiple documents.

```python
# import required module
from sklearn.feature_extraction.text import TfidfVectorizer

# assign corpus
string = ['Geeks geeks']*5

# create object
tfidf = TfidfVectorizer()

# get tf-df values
result = tfidf.fit_transform(string)

# get indexing
print('\nWord indexes:')
print(tfidf.vocabulary_)

# display tf-idf values
print('\ntf-idf values:')
print(result)
```

Output:

```
Word indexes:
{'geeks': 0}

tf-idf values:
  (0, 0)        1.0
  (1, 0)        1.0
  (2, 0)        1.0
  (3, 0)        1.0
  (4, 0)        1.0
```



Under
TF-IDI

TF-
IDF in
Action

Understanding TF-IDF

Visit Course

Comment More info Advertise with us

Next Article

Mathematical explanation of K-Nearest Neighbour

Similar Reads

Bag of word and Frequency count in text using sklearn

Text data is ubiquitous in today's digital world, from emails and social media posts to research articles and customer reviews. To analyze and derive insights from this textual information, it's essential to convert text...

3 min read

Understanding min_df and max_df in scikit CountVectorizer

In natural language processing (NLP), text preprocessing is a critical step that significantly impacts the performance of machine learning models. One common preprocessing step is converting text data into...

6 min read

NLP | Storing Conditional Frequency Distribution in Redis

The nltk.probability.ConditionalFreqDist class is a container for FreqDist instances, with one FreqDist per condition. It is used to count frequencies that are dependent on another condition, such as another word or a...

2 min read

Classification of text documents using sparse features in Python Scikit Learn

Classification is a type of machine learning algorithm in which the model is trained, so as to categorize or label

An Easy Approach to TF-IDF Using R

TF-IDF (Term Frequency-Inverse Document Frequency) is a fundamental technique in natural language processing and information retrieval for assessing the importance of a term within a document relative to a...

5 min read

Document Similarity Using LSA in R

Latent Semantic Analysis (LSA) is a powerful technique in natural language processing (NLP) that helps uncover the hidden relationships between words in a set of documents. Unlike basic word-matching...

5 min read

Using CountVectorizer to Extracting Features from Text

CountVectorizer is a great tool provided by the scikit-learn library in Python. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text. This is...

3 min read

Visualizing TF-IDF Scores: A Comprehensive Guide to Plotting a Document TF-IDF 2D…

For Natural Language Processing (NLP), Term Frequency-Inverse Document Frequency (TF-IDF) is a crucial technique used to evaluate the importance of a word in a document relative to a collection of documents...

4 min read

Latent Text Analysis (lsa Package) Using Whole Documents in R

Latent Text Analysis (LTA) is a technique used to discover the hidden (latent) structures within a set of documents. This approach is instrumental in natural language processing (NLP) for identifying patterns, topic...

8 min read

Create a Term-Document Matrix with a Dictionary of One or Two Words using R

A Term-Document Matrix (TDM), is a significant technique in text mining, used to represent the frequency of words across a collection of documents. Each row in the matrix corresponds to a unique phrase, and each...

4 min read

GeeksforGeeks
Sanchhaya Education Private Limited
Corporate & Communications Address:
A-143, 7th Floor, Sovereign Corporate