**TF-IDF (term frequency-inverse document frequency)** is a statistical measure that evaluates how relevant a word is to a document in a collection of documents.

This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents.

It has many uses, most importantly in automated text analysis, and is very useful for scoring words in machine learning algorithms for  (NLP).

TF-IDF was invented for document search and information retrieval. It works by increasing proportionally to the number of times a word appears in a document, but is offset by the number of documents that contain the word. So, words that are common in every document, such as this, what, and if, rank low even though they may appear many times, since they don’t mean much to that document in particular.

However, if the word Bug appears many times in a document, while not appearing many times in others, it probably means that it’s very relevant. For example, if what we’re doing is trying to find out which topics some NPS responses belong to, the word Bug would probably end up being tied to the topic Reliability, since most responses containing that word would be about that topic.

How is TF-IDF calculated?

TF-IDF for a word in a document is calculated by multiplying two different metrics:

* The **term frequency** of a word in a document. There are several ways of calculating this frequency, with the simplest being a raw count of instances a word appears in a document. Then, there are ways to adjust the frequency, by length of a document, or by the raw frequency of the most frequent word in a document.
* The **inverse document frequency** of the word across a set of documents. This means, how common or rare a word is in the entire document set. The closer it is to 0, the more common a word is. This metric can be calculated by taking the total number of documents, dividing it by the number of documents that contain a word, and calculating the logarithm.
* So, if the word is very common and appears in many documents, this number will approach 0. Otherwise, it will approach 1.

Multiplying these two numbers results in the TF-IDF score of a word in a document. The higher the score, the more relevant that word is in that particular document.



## Applications of TF-IDF

Determining how relevant a word is to a document, or TD-IDF, is useful in many ways, for example:

* **Information retrieval**

TF-IDF was invented for document search and can be used to deliver results that are most relevant to what you’re searching for. Imagine you have a search engine and somebody looks for LeBron. The results will be displayed in order of relevance. That’s to say the most relevant sports articles will be ranked higher because TF-IDF gives the word LeBron a higher score.

It’s likely that every search engine you have ever encountered uses TF-IDF scores in its algorithm.

* **Keyword Extraction**

TF-IDF is also useful for extracting keywords from text. How? The highest scoring words of a document are the most relevant to that document, and therefore they can be considered *keywords* for that document. Pretty straightforward.

Final words

It’s useful to understand how TF-IDF works so that you can gain a better understanding of how machine learning algorithms function. While machine learning algorithms traditionally work better with numbers, TF-IDF algorithms help them decipher words by allocating them a numerical value or vector. This has been revolutionary for machine learning, especially in fields related to NLP such as text analysis.

In text analysis with machine learning, TF-IDF algorithms help sort data into categories, as well as extract keywords. This means that simple, monotonous tasks, like tagging support tickets or rows of feedback and inputting data can be done in seconds.

import pandas as pd

from sklearn.feature\_extraction.text import TfidfVectorizer

import requests

# Replace with actual URLs for resume and job application

resume\_url = "https://www.academia.edu/14730001/Bill\_Gates\_Biography\_in\_full\_William\_Henry\_Gates\_III\_1955"

job\_application\_url = "https://www.comsats.edu.pk/nefiles/eligibilitycriteria\_nonfacultyposition.pdf"

# Fetch text data from URLs

def fetch\_text\_from\_url(url):

    try:

        response = requests.get(url)

        if response.status\_code == 200:

            return response.text

        else:

            print(f"Error fetching data from {url}")

            return None

    except Exception as e:

        print(f"Error fetching data from {url}: {e}")

        return None

# Get resume and job application text

resume\_text = fetch\_text\_from\_url(resume\_url)

job\_application\_text = fetch\_text\_from\_url(job\_application\_url)

if resume\_text and job\_application\_text:

    # Concatenate all texts

    all\_texts = [resume\_text, job\_application\_text]

    # Initialize the TF-IDF vectorizer

    vectorizer = TfidfVectorizer(stop\_words='english')

    # Fit and transform all texts

    tfidf\_matrix = vectorizer.fit\_transform(all\_texts)

    # Calculate cosine similarity matrix

    cosine\_similarity = (tfidf\_matrix \* tfidf\_matrix.T).toarray()

    print(cosine\_similarity)

    # Accuracy and F-score

    threshold = 0.5

    # Assuming the documents are similar

    predicted\_similarity = (cosine\_similarity[0, 1] > threshold).astype(int)

    actual\_similarity = 1

    print(predicted\_similarity)

else:

    print("Error fetching data from URLs. Please check the URLs.")

[[1. 0.02204341]

[0.02204341 1. ]]

0