

Lesson Number- 11

TF-IDF

Objectives:

- Understand TF-IDF Concept
- Apply TF-IDF to Text Data and analyze the Role of TF-IDF in Search Ranking & Information Retrieval

Question?

 **Imagine you are building a search engine for movie reviews.**

A user searches for **"amazing action movie"**, and you need to rank the following reviews:

- 1 **"This action movie is amazing and full of thrill."**
- 2 **"Amazing cinematography, but the action scenes were average."**
- 3 **"The movie had action sequences, but it was not amazing."**

 **How do you decide which review is the most relevant?**


☒ Do you just count the number of times words like "amazing" and "action" appear?

☒ What if a common word like **"movie"** appears in every review—should it affect the ranking?

☒ Should **rare words be given more weight** than common words?

 **How can we ensure that words that truly matter for relevance (like "amazing" and "action") get more importance while ignoring generic words like "movie"?**

TF-IDF

 **TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical measure that evaluates how important a word is to a document within a collection (corpus) of documents.**

It balances **word frequency within a document (TF)** and **word rarity across all documents (IDF)** to prioritize meaningful words while reducing the impact of common ones.

Corpus

A corpus can have thousands of words, but the TF-IDF model **filters** them based on:

- Preprocessing (stopword removal, stemming, etc.)
- Minimum document frequency threshold (removing extremely rare words)
- Removing punctuation and special characters

Vocab

In **Natural Language Processing (NLP)**, **vocabulary (vocab)** refers to the **unique words** extracted from a dataset (corpus).

Document

In the dataset, each **document** represents a **single review/example/sample**.

IDF

IDF remains the same for a word across all reviews because IDF is calculated at the corpus level, not at the individual document level.

Question (Solve activity)

Why Should We Reduce the Impact of Common Words in Text Analysis?

Activity

We have 10,000 reviews, and the following words appear in the dataset:

- "the" appears in all 10,000 reviews.
- "food" appears in 5,000 reviews.
- "good" appears in 5,000 reviews.
- "bad" appears in 5,000 reviews.
- "terrible" appears in only 1 review.

Question1. What weight should we assign to "the" since it appears in all reviews? Hint: Common words like "the" add no meaning to sentiment analysis. If we assign equal importance to every word, words like "the" will dominate the analysis.

Introduce a formula to downweight frequent words and boost rare but meaningful words. total number of reviews where "the" is present is 10k, and total reviews are 10k. Make sure "terrible" gets a higher weight than "food" or "bad"?

Answer

- Total Reviews = 10,000
- Reviews containing "the" = 10,000
- Reviews containing "food" = 5,000
- Reviews containing "terrible" = 1

📌 Calculate the IDF for "the", "food", and "terrible".

✅ Answer:

Word	Total Reviews Containing Word	IDF Calculation	IDF Value
the	10,000	$\log(10,000 / 10,000)$	0.00 🚫
food	5,000	$\log(10,000 / 5,000)$	0.30
terrible	1	$\log(10,000 / 1)$	4.00 ✅

📌 Observations:

- "the" gets an IDF of 0 → It has no importance for distinguishing sentiment.
- "terrible" has a high IDF of 4 → It is rare and important.
- Common words like "food" get moderate importance.

Why use log

📌 Why Use Logarithm in IDF Calculation?

The Inverse Document Frequency (IDF) formula is:

$$IDF(w) = \log \left(\frac{\text{Total Documents}}{\text{Number of Documents Containing } w} \right)$$

where:

- **Total Documents** = The number of documents in the dataset.
- **Number of Documents Containing w** = The number of documents where the word appears.

The **logarithm** is used in IDF to **scale down large values and avoid extreme differences**. Let's break it down step by step.

1 Preventing Extremely High IDF Scores

- If a word appears in very few documents, its IDF score will be high.
- Without **log**, IDF values would be **too large**, making rare words **overly important**.

Example Without Log:

Word	DF (Number of Docs Containing Word)	IDF (Without Log) = Total Docs / DF
"amazing"	2	20 / 2 = 10
"bad"	10	20 / 10 = 2
"was"	20	20 / 20 = 1

➡ **Issue:** The difference between **rare** and **common words** is **too extreme**.

Example With Log:

Using **log transformation**:

$$IDF(w) = \log \left(\frac{20}{DF} \right)$$

Word	DF	IDF (With Log)
"amazing"	2	1.609
"bad"	10	0.693
"was"	20	0.000

✅ **Log smooths the differences** and makes the values more comparable.

2 Handling Rare Words More Realistically

- **Rare words** should be **more important** than common words but **not overly dominant**.
- **Log scaling** reduces the weight of **extremely rare words** so that they don't dominate the model.

➡ Example:

- Without log:
 - If a word appears **once in 20 documents**, it would have $IDF = 20$, which is **too high**.
 - With log:

$$\log(20) = 2.995$$

→ More reasonable.

3 Handling Common Words

- **Common words** like "was", "the" should have **lower IDF**.
- Using **log** ensures these words don't get **IDF values near zero** but remain low.

➡ Example:

- If a word appears **in every document**:
 - Without log → $IDF = 1$ (not much difference from rare words).
 - With log → $IDF = 0$, correctly indicating it is not useful.
-

4 Computational Stability

- **Without log**, differences between large and small values cause **unstable training** for machine learning models.
- Using **log** helps maintain **balance in text vectorization**.

We want to **calculate the weight of words** using the following formula:

$$IDF(w) = \log \left(\frac{\text{Total Number of Reviews}}{\text{Number of Reviews Containing Word (w)}} \right)$$

Given Data:

- Total Reviews = 10,000
- Reviews containing "the" = 10,000
- Reviews containing "food" = 5,000
- Reviews containing "terrible" = 1

Term Frequency (TF)?

 **Imagine Two Reviews About a Product:**

① **Short Review:**

"The phone is excellent." (Total words = 4)

✓ **"phone" appears 1 time → TF(phone) = 1**

② **Long Review:**

"The phone is excellent. The phone is very fast and smooth. I love this phone!" (Total words = 14)

✓ **"phone" appears 3 times → TF(phone) = 3**

 **Issue:**

- The second review has a **higher raw TF score (3 vs. 1)** just because it is **longer**.
- **Longer reviews will always dominate** shorter ones, even if both mention the same word.
- **This is unfair!** We need a way to **normalize** term frequency.

We divide **TF (Term Frequency)** by the **total number of words in a document** to **normalize word frequency** so that longer documents do not unfairly dominate the scores.

◆ Solution: Normalize by Total Words in the Review

✓ Formula for Normalized TF (Term Frequency):

$$TF(w, d) = \frac{\text{Number of times word } w \text{ appears in document } d}{\text{Total words in document } d}$$

✚ Applying the Formula to Our Example:

Review	Total Words	TF(phone) (Count)	TF(phone) (Normalized)
Short Review	4	1	1 / 4 = 0.25
Long Review	14	3	3 / 14 = 0.21

✓ Now, both reviews are fairly compared because the longer review no longer dominates unfairly.

Activity 2

Calculate TF-IDF of the reviews below-

✓ Sentence 1: *"The food was bad, really bad, extremely bad."*

- TF("bad") =
- IDF("bad") =
- TF-IDF("bad") =

✓ Sentence 2: *"The food was bad."*

- TF("bad") =
- IDF("bad") =
- TF-IDF("bad") =

Activity Discussion

- ★ TF helps differentiate between reviews where a word is strongly emphasized and where it is just mentioned once.
- ★ Words appearing more frequently in a sentence get higher weight.
- ★ This prevents IDF from treating all occurrences of a rare word equally.

TF Calculation:

Word	TF Calculation	IDF Value	TF-IDF Score
the	$1/4 = 0.25$	0.00	0.00 🚫
food	$1/4 = 0.25$	0.30	0.075
good	$1/4 = 0.25$	0.30	0.075

✓ Observations:

- "the" has no impact because its TF-IDF = 0.00.
- "food" and "good" have moderate importance because they appear in 5,000 reviews.
- If we had a rare word (e.g., "terrible"), its TF-IDF score would be much higher!

Limitations

1. TF-IDF Does Not Capture Word Order

📌 Problem:

- TF-IDF only counts words and does not consider their position in a sentence.
- This means it cannot differentiate between phrases with opposite meanings.

💡 Example:

- "Not good at all." ❌ (Negative Sentiment)
- "Good, not at all bad." ✓ (Positive Sentiment)

✓ TF-IDF assigns the same scores to both sentences because it ignores word order.

🚀 Better Solution: N-grams or Transformer models capture word sequences and dependencies.

2. TF-IDF Fails with Large or Dynamic Datasets

📌 Problem:

- TF-IDF requires precomputed document frequencies, making it inefficient for real-time applications.
- When new documents are added, the entire IDF calculation needs to be updated.

💡 Example:

- If a search engine adds new web pages, the IDF values must be recalculated, making TF-IDF slow for dynamic indexing.

Applications

- ✓ Filters Out Common Words ("hotel" has 0 weight)
- ✓ Boosts Important Words ("amazing", "service" are ranked higher)
- ✓ Ranks Documents by Relevance Based on Query

Real-World Uses:

- Google & Bing use TF-IDF to rank search results.
- E-commerce websites (Amazon, Flipkart) use it for product search ranking.
- News websites (NY Times, BBC) use TF-IDF to recommend relevant articles.

Example: How TF-IDF is Stored & Used in E-Commerce Product Search (Amazon, Flipkart, etc.)

Scenario:

You search for "budget smartphone with good camera" on Amazon. The system must rank millions of products instantly.

Step 1: Creating an Inverted Index for TF-IDF

- ✓ Amazon has millions of products with titles and descriptions like:

Product ID	Product Title	Description
P1	"Budget smartphone with excellent camera."	Affordable phone with 64MP camera.
P2	"Best smartphone under ₹10,000 with good battery."	Long-lasting battery, but average camera.
P3	"Premium smartphone with DSLR-like camera."	High-end flagship phone.

TF-IDF values are computed beforehand for words in the product catalog and stored in an index.

💡 This is called an **inverted index**, which helps retrieve products quickly.

Step 2: Storing TF-IDF in an Inverted Index

📌 Instead of recalculating TF-IDF for every search, Amazon stores precomputed scores like this:

Word	Product P1 (Budget Smartphone)	Product P2 (Best Smartphone)	Product P3 (Premium Smartphone)
budget	1.2	0.5	0.0
smartphone	0.7	1.1	1.0
good	0.0	1.5	0.0
camera	2.0	0.8	3.0

Step 3: Computing the Relevance Score

$$\text{Total Score} = TF - IDF(\text{budget}) + TF - IDF(\text{smartphone}) + TF - IDF(\text{good}) + TF - IDF(\text{camera})$$

Amazon retrieves and sums the precomputed TF-IDF values from the inverted index for all words in the search query and ranks products based on the total relevance score.

Product	TF-IDF (budget)	TF-IDF (smartphone)	TF-IDF (good)	TF-IDF (camera)	Total Score
P1 (Budget Smartphone)	1.2	0.7	0.0	2.0	3.9 ✅
P2 (Best Smartphone)	0.5	1.1	1.5	0.8	3.9 ✅
P3 (Premium Smartphone)	0.0	1.0	0.0	3.0	4.0 🚀

Step 4: Ranking & Retrieving Search Results

📌 Final Product Ranking Based on Relevance Score: ① P3 (Premium Smartphone) → Score = 4.0 ✅ (Most relevant!)

② P1 (Budget Smartphone) → Score = 3.9 ✅

③ P2 (Best Smartphone) → Score = 3.9

🚀 Amazon then displays these results within milliseconds!

Code

https://colab.research.google.com/drive/1djeci_xPXk95PsPBrVZ_klkq6VYDX6pH#scrollTo=yOW-nkU-4B1g