TF-IDF (Term Frequency-Inverse Document Frequency)

💡 **It is a technique used to convert text into numbers, just like Bag of Words, but it's smarter — it tries to capture importance of words.**

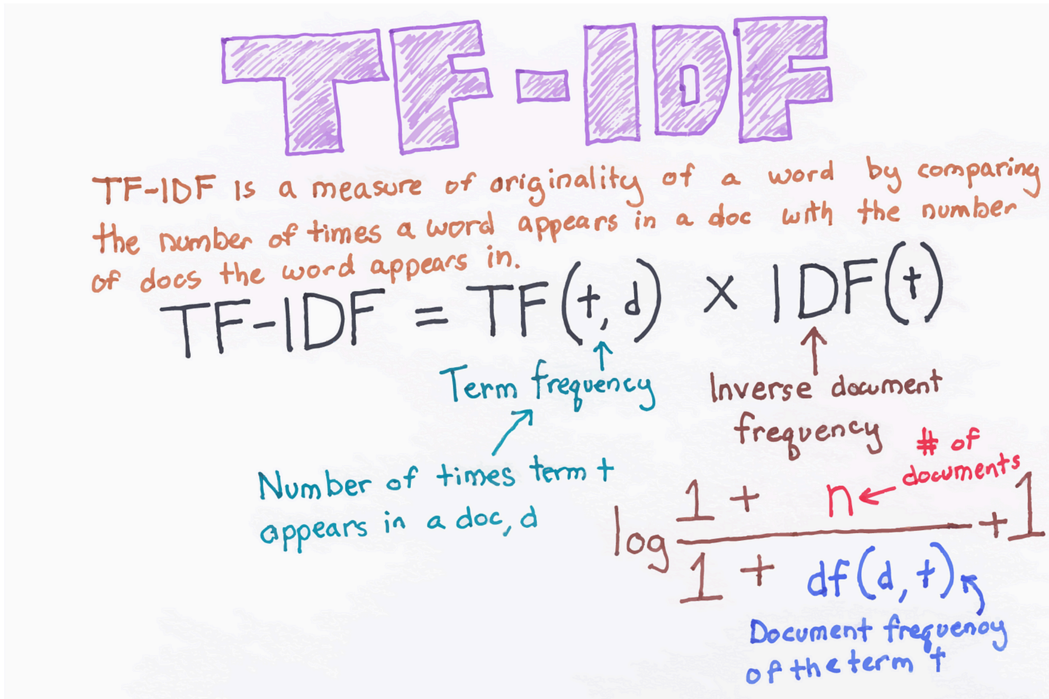
🎯 Purpose of TF-IDF

- **BoW** treats all words equally — even common words like "the", "is", "and" get high scores.
- **TF-IDF** solves this by **reducing the weight of common words** and **increasing the weight of rare but meaningful words**.

In short:

> "TF-IDF tells you how important a word is in a document, relative to a collection of documents."

🧱 TF-IDF = TF × IDF



🔷 1. Term Frequency (TF)

- This is the **number of times** a word appears in a document.
- But we usually **normalize** it (to prevent bias from longer docs):

$$\mathrm{TF}(w, d) = \frac{\text{Count of word } w \text{ in document } d}{\text{Total words in document } d}$$

🔷 2. Inverse Document Frequency (IDF)

This tells us **how rare or common** a word is across all documents.

$$\mathrm{IDF}(w) = \log\left(\frac{N}{1 + \text{Number of documents containing } w}\right)$$

💡 👆The **log** is → log_e

- **N** = Total number of documents
- Rare words → high IDF
- Common words → low IDF

💡 **Corpus** → Paragraph

Document → Sentence

Vocabulary → Unique Words

Words → All the words present in corpus

🧮 TF-IDF = TF × IDF

- **High** when: the word appears often in one document but not in others.
- **Low** when: the word is common in many documents.

(4) TF-IDF [Term Frequency – Inverse Document Frequency]

S1 → good boy

S2 → good girl

S3 → boy girl good

$$\text{Term Freq (TF)} = \frac{\text{No. of rep of words in sentence}}{\text{No. of words in sentence}}$$

$$\text{IDF} = \log_e\left(\frac{\text{No. of sentences}}{\text{No. of sentences containing the word}}\right)$$

Term Frequency

	S1	S2	S3
good	$1/2$	$1/2$	$1/3$
boy	$1/2$	0	$1/3$
girl	0	$1/2$	$1/3$

*

IDF

Words	IDF
good	$\log_e(3/3) = 0$
boy	$\log_e(3/2)$
girl	$\log_e(3/2)$

	good	boy	girl
Sent 1	0	$\frac{1}{2} \times \log_e(3/2)$	0
Sent 2	0	0	$\frac{1}{2} \log_e(3/2)$
Sent 3	0	$\frac{1}{3} \log_e(3/2)$	$\frac{1}{3} \log_e(3/2)$

🧪 Example

Suppose you have 3 documents:

```
D1 = "I love pizza"
D2 = "I love burgers"
D3 = "I hate pizza"
```

Step 1: Vocabulary

`["I", "love", "pizza", "burgers", "hate"]`

Step 2: Count word frequencies → TF

Normalize counts (optional)

Step 3: Count in how many documents each word appears → IDF

Word	Appears in Docs	IDF Calculation	IDF (approx)
I	3	log(3 / (1+3)) = log(1)	0.0
love	2	log(3 / (1+2)) = log(1.5)	0.4
pizza	2	log(3 / (1+2)) = log(1.5)	0.4
burgers	1	log(3 / (1+1)) = log(1.5)	0.4
hate	1	log(3 / (1+1)) = log(1.5)	0.4

Then multiply **TF × IDF** for each word in each document.

$\mathtt{tf}(t, d)$

	blue	bright	can	see	shining	sky	sun	today
1	1/2	0	0	0	0	1/2	0	0
2	0	1/3	0	0	0	0	1/3	1/3
3	0	1/3	0	0	0	1/3	1/3	0
4	0	1/6	1/6	1/6	1/6	0	1/3	0

x

$\mathtt{idf}(t, D)$

	blue	bright	can	see	shining	sky	sun	today
	0.602	0.125	0.602	0.602	0.602	0.301	0.125	0.602

- TF-IDF: Multiply TF and IDF scores, use to rank importance of words within documents
  - Most important word for each document is highlighted

$$\mathtt{tfidf}(t, d, D) = \mathtt{tf}(t, d) \cdot \mathtt{idf}(t, D)$$

	blue	bright	can	see	shining	sky	sun	today
1	**0.301**	0	0	0	0	0.151	0	0
2	0	0.0417	0	0	0	0	0.0417	**0.201**
3	0	0.0417	0	0	0	**0.100**	0.0417	0
4	0	0.0209	**0.100**	**0.100**	**0.100**	0	0.0417	0

✅ Python Example using scikit-learn

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

docs = ["I love pizza", "I love burgers", "I hate pizza"]

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(docs)

# View results
print(X.toarray())  # The TF-IDF matrix
print(vectorizer.get_feature_names_out())
```

```
[[0.         0.         0.70710678 0.70710678]
 [0.79596054 0.         0.60534851 0.        ]
 [0.         0.79596054 0.         0.60534851]]
['burgers' 'hate' 'love' 'pizza']
```

Key Parameters in `TfidfVectorizer`

Parameter	Description	Example
`stop_words`	Remove common words (`'english'`)	`stop_words='english'`
`ngram_range`	Include word pairs (`(1,2)`)	`ngram_range=(1, 2)`
`max_features`	Limit vocabulary size	`max_features=1000`
`norm`	Normalize vectors (`'l2'` , `'l1'`)	`norm='l2'`

🧠 Interpretation of Output

- The **TF-IDF matrix** contains floating-point numbers.
- Each number shows how important a word is in that document.
- **Words that appear in many documents have low TF-IDF.**

🧮 Comparison with Bag of Words

Feature	Bag of Words	TF-IDF
Word Count	Yes	Yes
Penalize Common Words	❌ No	✅ Yes
Capture Importance	❌ No	✅ Yes
Values	Integers (0,1,2...)	Floats (0.0 to 1.0 approx.)
Sparse Matrix	Yes	Yes

✨ Where TF-IDF is Used

- Document classification
- Search engines (ranking relevant documents)
- Spam filtering
- Information retrieval
- Text clustering

✅Advantages of TF-IDF

1. **Highlights important words** and downplays common ones (***IMP for interview***)

- If a word is present in every sentence, less importance is given to that word

2. **Improves accuracy** of classical ML models
3. **Fast and efficient** to compute
4. **Works without labeled data** (unsupervised)
5. **Interpretable** — easy to see which words matter
6. **Customizable** with parameters like `min_df`, `max_df`, `ngram_range`

⚠️ Limitations of TF-IDF

Limitation	Why it happens
❌ Ignores word order	Like BoW
❌ No context	"bank" in money vs river — treated same
❌ Not deep	Cannot detect meaning or sarcasm
❌ Still sparse	For large text, matrix still full of zeros

📚 Trivia

- **TF-IDF is unsupervised** — no labels are needed.
- It's still used as a **baseline** in many NLP systems.
- In deep learning, it's replaced by **word embeddings** (Word2Vec, BERT, etc.) — but TF-IDF is still useful when:
  - You want fast results
  - You want explainable features
  - You're working with small data

> 💡 **Word2Vec is still best.**