6.TF-IDF: Term Frequency-Inverse Document Frequency

Introduction

TF-IDF (Term Frequency-Inverse Document Frequency) is a numerical statistic used in natural language processing to convert text documents into vectors. It reflects how important a word is to a document within a collection of documents.

Components of TF-IDF

TF-IDF consists of two main components that are multiplied together:

1. Term Frequency (TF)

Term Frequency measures how frequently a word appears in a specific document.

Formula:

$$\text{TF}(word, sentence) = \frac{\text{Number of times word appears in sentence}}{\text{Total number of words in sentence}}$$

Where:

- **Numerator**: Count of how many times the specific word occurs in that sentence
- **Denominator**: Total count of all words in that sentence

Example: In the sentence "good boy":

- TF(good) = $\frac{1}{2}$ = 0.5 (word "good" appears 1 time, total words = 2)
- TF(boy) = $\frac{1}{2}$ = 0.5 (word "boy" appears 1 time, total words = 2)

2. Inverse Document Frequency (IDF)

Inverse Document Frequency measures how important a word is across all documents. It gives higher weight to rare words and lower weight to common words.

Formula (Standard Smoothed Version):

$$\text{IDF}(word) = \log_e\left(\frac{\text{Total number of sentences} + 1}{\text{Number of sentences containing the word} + 1}\right) + 1$$

Where:

- $\log_e$: Natural logarithm (logarithm to base e)
- **Numerator**: Total count of all sentences in the corpus, **plus 1** for smoothing
- **Denominator**: Count of sentences that contain this specific word, **plus 1** for smoothing
- **Final +1**: Added to the entire result to ensure all IDF values are positive

Why this formula?

- The +1 in numerator and denominator prevents division by zero for unseen words
- The final +1 ensures IDF is never negative (always ≥ 1)
- This is the standard implementation used in scikit-learn and most libraries
- Provides numerical stability for real-world applications

Note: The transcript uses a simplified formula without smoothing, but production implementations use this smoothed version.

Example: For word "good" across 3 sentences where it appears in all 3:

- IDF(good) = $\log_e\left(\frac{3+1}{3+1}\right) + 1$ = $\log_e(1) + 1$ = 0 + 1 = **1.0**

For word "boy" that appears in 2 out of 3 sentences:

- IDF(boy) = $\log_e\left(\frac{3+1}{2+1}\right) + 1$ = $\log_e\left(\frac{4}{3}\right) + 1$ = $\log_e(1.333) + 1$ ≈ 0.288 + 1 = **1.288**

3. TF-IDF Score

The final TF-IDF score is calculated by multiplying TF and IDF:

$$\text{TF-IDF}(word, sentence) = \text{TF}(word, sentence) \times \text{IDF}(word)$$

Where:

- **TF(word, sentence)**: Term frequency of the word in that specific sentence
- **IDF(word)**: Inverse document frequency of the word across all sentences

Working Example

Let's work through a complete example with three sentences:

Corpus:

- Sentence 1: "good boy"
- Sentence 2: "good girl"
- Sentence 3: "boy girl good"

Vocabulary: {good, boy, girl}

Step 1: Calculate Term Frequency (TF)

Sentence	good	boy	girl
S1	$\frac{1}{2}$ = 0.5	$\frac{1}{2}$ = 0.5	$\frac{0}{2}$ = 0
S2	$\frac{1}{2}$ = 0.5	$\frac{0}{2}$ = 0	$\frac{1}{2}$ = 0.5
S3	$\frac{1}{3}$ ≈ 0.333	$\frac{1}{3}$ ≈ 0.333	$\frac{1}{3}$ ≈ 0.333

Step 2: Calculate Inverse Document Frequency (IDF)

Word	IDF Calculation	Value
good	$\log_e\left(\frac{3+1}{3+1}\right) + 1 = \log_e(1) + 1$	**1.0**
boy	$\log_e\left(\frac{3+1}{2+1}\right) + 1 = \log_e(1.333) + 1$	**≈ 1.288**
girl	$\log_e\left(\frac{3+1}{2+1}\right) + 1 = \log_e(1.333) + 1$	**≈ 1.288**

Explanation:

- "good" appears in all 3 sentences → IDF = 1.0 (common word, gets minimum weight)
- "boy" appears in 2 sentences → IDF ≈ 1.288 (more distinctive, higher weight)
- "girl" appears in 2 sentences → IDF ≈ 1.288 (more distinctive, higher weight)

Step 3: Calculate TF-IDF Scores

Sentence 1: "good boy"

- good: $\frac{1}{2} \times 1.0$ = **0.5**
- boy: $\frac{1}{2} \times 1.288$ ≈ **0.644**
- girl: $\frac{0}{2} \times 1.288$ = **0**

Final Vector: [0.5, 0.644, 0]

Sentence 2: "good girl"

- good: $\frac{1}{2} \times 1.0$ = **0.5**
- boy: $\frac{0}{2} \times 1.288$ = **0**
- girl: $\frac{1}{2} \times 1.288$ ≈ **0.644**

Final Vector: [0.5, 0, 0.644]

Sentence 3: "boy girl good"

- good: $\frac{1}{3} \times 1.0$ ≈ **0.333**
- boy: $\frac{1}{3} \times 1.288$ ≈ **0.429**
- girl: $\frac{1}{3} \times 1.288$ ≈ **0.429**

Final Vector: [0.333, 0.429, 0.429]

Key Insight: Word Importance

The genius of TF-IDF is that it automatically captures word importance:

- **Words appearing in ALL sentences** (like "good") get the lowest IDF value (1.0)
  - These words don't help distinguish between documents
  - They appear everywhere, so they're less informative
- **Words appearing in FEWER sentences** (like "boy", "girl") get higher IDF values (>1.0)
  - These words are more distinctive and important for understanding context
  - The rarer the word across documents, the higher its importance weight
- **All IDF values are positive and ≥ 1.0**, making interpretation intuitive
  - Higher values = more important/distinctive words
  - Lower values (closer to 1) = common, less distinctive words

Advantages of TF-IDF

1. **Intuitive Implementation**: Easy to understand and implement mathematically
2. **Fixed Input Size**: Vector dimensions are fixed based on vocabulary size, making it compatible with machine learning algorithms
3. **Captures Word Importance ⭐ (Most Important Advantage)**:
    - Automatically identifies which words are more significant
    - Common words (appearing in many documents) are de-emphasized with lower weights
    - Distinctive words (appearing in fewer documents) are emphasized with higher weights
    - This improves model accuracy by focusing on meaningful, discriminative words
    - Unlike Bag of Words which treats all words equally

Disadvantages of TF-IDF

1. **Sparsity**: Vectors still contain many zeros, leading to sparse matrices
    - Most words don't appear in most documents
    - Results in high-dimensional, sparse representations
2. **Out-of-Vocabulary (OOV) Problem**:
    - New words in test data that weren't in training vocabulary are completely ignored
    - Cannot handle unseen words gracefully
3. **No Semantic Understanding**:
    - Words with similar meanings (e.g., "good" and "excellent") are treated as completely different
    - Doesn't capture synonyms, context, or word relationships

TF-IDF vs Bag of Words

Why TF-IDF performs better than Bag of Words:

Aspect	Bag of Words	TF-IDF
Word Weighting	Equal (1 if present, 0 if absent)	Weighted by importance
Common Words	Treated same as rare words	De-emphasized (lower weight)
Distinctive Words	No special treatment	Emphasized (higher weight)
Context Awareness	None	Captures relative importance

Aspect	Bag of Words	TF-IDF
Model Performance	Baseline	Significantly better

Key Difference: TF-IDF weighs words based on how distinctive they are, while Bag of Words treats all words equally. This makes TF-IDF more informative for machine learning models.

Conclusion

TF-IDF is a powerful and widely-used technique for converting text to numerical vectors while capturing the relative importance of words. Although it has limitations (sparsity, OOV issues, no semantic understanding), it performs significantly better than simple Bag of Words approaches and remains a standard baseline in many NLP applications.

For even better performance, modern approaches like Word2Vec, GloVe, and transformer-based embeddings (BERT, GPT) address TF-IDF's limitations by capturing semantic relationships and context.