COP509

Natural Language Processing

The Vector Space Model

Part 2: Term Frequency and TF-IDF

Prof. Georgina Cosma

Department of Computer Science
Loughborough University

`g.cosma@lboro.ac.uk`

Learning Outcomes

By the end of this session, you will be able to:

1. Explain the concept of term frequency (TF) and why it matters
2. Apply log scaling to dampen term frequency
3. Understand document frequency and inverse document frequency (IDF)
4. Calculate TF-IDF weights for terms in documents
5. Explain why TF-IDF is better than raw term frequency

Recap from Part 1

- Ranked retrieval assigns scores to documents
- The Jaccard coefficient ignores term frequency and term importance
- We need a better approach!

Mathematical Notation Guide

Key notation used in this lecture:

Symbol	Meaning
$\text{tf}_{t,d}$	**Term frequency**: number of times term t appears in document d
df_t	**Document frequency**: number of documents containing term t
N	Total number of documents in the collection
$\log_{10}(x)$	Logarithm base 10 of x (how many times you multiply 10 to get x)
$w_{t,d}$	The **weight** assigned to term t in document d
$\|V\|$	The size of the vocabulary (total number of unique terms)
$\sum$	Summation (add up all the values)

Subscript convention: t = term, d = document. So $\text{tf}_{t,d}$ reads as "term frequency of term t in document d".

Quick Refresher: Logarithms

What is a logarithm? $\log_{10}(x)$ asks: "10 to what power gives x?"

Examples:

- $\log_{10}(10) = 1$ because $10^1 = 10$
- $\log_{10}(100) = 2$ because $10^2 = 100$
- $\log_{10}(1000) = 3$ because $10^3 = 1000$
- $\log_{10}(1) = 0$ because $10^0 = 1$

Key property for NLP: Logarithms grow **slowly**

- If x increases from 1 to 1000 ($1000\times$ bigger), $\log_{10}(x)$ only goes from 0 to 3
- This "dampens" large values, which is useful for term frequencies!

Note: We use $\log_{10}$ in these slides. Some systems use natural log (ln). The principle is the same.

Outline for Part 2

1. From Sets to Counts: The Term-Document Matrix
2. Term Frequency (TF)
3. Log Frequency Weighting
4. Document Frequency and IDF
5. Combining TF and IDF: The TF-IDF Weight
6. TF-IDF in Practice

Section 1

From Sets to Counts

The Term-Document Matrix

Representing Documents: Binary vs. Count

Recall from Boolean Retrieval: We used a **binary incidence matrix**

	Hamlet	**Othello**	**Macbeth**	**Julius Caesar**	**Tempest**
BRUTUS	1	0	0	1	0
CAESAR	1	1	1	1	0
MERCY	1	1	1	0	1

Problem: Only tells us if a term is present (1) or absent (0).

Better approach: Use a **count matrix**. Record how many times each term appears!

	Hamlet	**Othello**	**Macbeth**	**Julius Caesar**	**Tempest**
BRUTUS	2	0	0	157	0
CAESAR	8	1	1	227	0
MERCY	8	5	8	0	3

Now we can see that “BRUTUS” appears 157 times in Julius Caesar!

The Bag of Words Model

When we use term counts, we adopt the **Bag of Words** model:

Bag of Words Assumption

The **order** of words in a document doesn't matter. Only the **counts** matter.

Example:

- Document 1: "John is quicker than Mary"
- Document 2: "Mary is quicker than John"

In the bag of words model, these documents are **identical**!

Advantages:

- Simple and efficient
- Works surprisingly well
- Easy to compute

Disadvantages:

- Loses word order
- "not good" = "good not"
- No phrase understanding

Section 2

Term Frequency (TF)

What is Term Frequency?

Definition: Term Frequency

The **term frequency** $\mathrm{tf}_{t,d}$ is the number of times term t appears in document d.

Example: In document $d =$ "machine learning uses learning algorithms for learning"

- $\mathrm{tf}_{\text{machine},d} = 1$
- $\mathrm{tf}_{\text{learning},d} = 3$
- $\mathrm{tf}_{\text{algorithms},d} = 1$
- $\mathrm{tf}_{\text{uses},d} = 1$
- $\mathrm{tf}_{\text{for},d} = 1$

Intuition: A document with more occurrences of a query term should be more relevant.

If you search for "learning", a document mentioning "learning" 10 times is probably more focused on learning than one mentioning it once.

Using Term Frequency for Scoring

Simple approach: Sum the term frequencies of query terms in the document.

Query: “machine learning”
Document: “machine learning uses learning algorithms for learning”

Matching score:

$$\text{score}(q, d) = \text{tf}_{\text{machine},d} + \text{tf}_{\text{learning},d} = 1 + 3 = 4$$

Problem: Is This Linear Relationship Correct?

A document with tf $= 10$ occurrences is *more relevant* than one with tf $= 1$.
But is it really **10 times** more relevant?

Think about it: The 10th occurrence of “machine” probably adds less relevance than the 1st occurrence. We need to *dampen* the effect of high frequencies.

Section 3

Log Frequency Weighting

Dampening Term Frequency

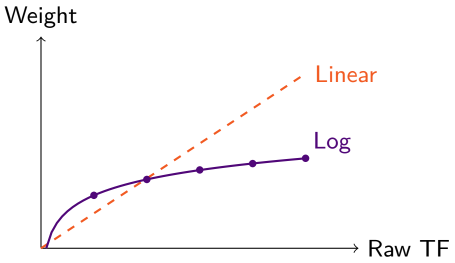
The Log Frequency Weight

Instead of using raw term frequency, we use **log-scaled term frequency**:

Log Frequency Weight

$$w_{t,d} = \begin{cases} 1 + \log_{10}(\mathrm{tf}_{t,d}) & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Why add 1? So that a term appearing once gets weight 1 (not 0).
Why use log? To dampen the effect of high frequencies.



Log Frequency: Worked Examples

Question: Why do we use $w_{t,d} = 1 + \log_{10}(\text{tf}_{t,d})$ instead of raw term frequency?

Answer: To dampen large frequencies. Let's see how:

Raw TF	Calculation	Log Weight
0	(term absent, use 0)	**0**
1	$1 + \log_{10}(1) = 1 + 0$	**1.00**
2	$1 + \log_{10}(2) = 1 + 0.30$	**1.30**
10	$1 + \log_{10}(10) = 1 + 1$	**2.00**
100	$1 + \log_{10}(100) = 1 + 2$	**3.00**
1000	$1 + \log_{10}(1000) = 1 + 3$	**4.00**

The Key Insight

Raw frequency: tf increases from $1 \rightarrow 1000$ (1000× increase)
Log weight: weight increases from $1.00 \rightarrow 4.00$ (only 4× increase)

Why this matters: A document mentioning "algorithm" 1000 times isn't 1000× more relevant than one mentioning it once. Logarithm scaling better reflects human perception of relevance.

Your Turn: Exercise

Calculate the log frequency weights for these term frequencies:

1. $\text{tf} = 5$ $\quad w = 1 + \log_{10}(5) = ?$
2. $\text{tf} = 7$ $\quad w = 1 + \log_{10}(7) = ?$
3. $\text{tf} = 50$ $\quad w = 1 + \log_{10}(50) = ?$

Your Turn: Exercise

Calculate the log frequency weights for these term frequencies:

1. tf $= 5$ $\quad w = 1 + \log_{10}(5) = ?$
2. tf $= 7$ $\quad w = 1 + \log_{10}(7) = ?$
3. tf $= 50$ $\quad w = 1 + \log_{10}(50) = ?$

Solutions:

1. $w = 1 + \log_{10}(5) = 1 + 0.70 = \mathbf{1.70}$
2. $w = 1 + \log_{10}(7) = 1 + 0.85 = \mathbf{1.85}$
3. $w = 1 + \log_{10}(50) = 1 + 1.70 = \mathbf{2.70}$

Tip: Use $\log_{10}(5) \approx 0.70$, $\log_{10}(7) \approx 0.85$, $\log_{10}(50) = \log_{10}(5 \times 10) = 0.70 + 1 = 1.70$

Scoring with Log Frequency

The TF matching score sums log-weighted frequencies for query terms:

$$\text{tf-score}(q, d) = \sum_{t \in q \cap d} (1 + \log_{10} \text{tf}_{t,d})$$

Example:
Query: "machine learning"
Document with: $\text{tf}_{\text{machine}} = 5$, $\text{tf}_{\text{learning}} = 20$

$$\begin{aligned} \text{score} &= (1 + \log_{10} 5) + (1 + \log_{10} 20) \\ &= (1 + 0.70) + (1 + 1.30) \\ &= 1.70 + 2.30 = \mathbf{4.0} \end{aligned}$$

Note: If a query term doesn't appear in the document, it contributes 0 to the score.

Section 4

Document Frequency and IDF

The Problem: Not All Terms Are Equal

Consider two query terms:

- "**the**": appears in almost every document
- "**arachnocentric**": appears in very few documents

Question: Which term is more useful for finding relevant documents?

The Problem: Not All Terms Are Equal

Consider two query terms:

- "**the**": appears in almost every document
- "**arachnocentric**": appears in very few documents

Question: Which term is more useful for finding relevant documents?

Answer: "arachnocentric" is much more informative!

- If a document contains "arachnocentric", it's probably relevant to spiders
- If a document contains "the", it tells us almost nothing

Key Insight

Rare terms are more informative than common terms.
We should give **higher weights to rare terms** and **lower weights to common terms**.

Document Frequency (DF)

Definition: Document Frequency

The **document frequency** df_t is the number of documents in the collection that contain term t.

Important distinction:

- **Term Frequency (TF):** How often a term appears *in one document*
- **Document Frequency (DF):** How many *documents* contain the term

Example: In a collection of 1,000,000 documents:

Term	**Document Frequency**	**Informativeness**
the	1,000,000	Very low (appears everywhere)
computer	100,000	Low
algorithm	10,000	Medium
arachnocentric	1	Very high (extremely rare)

Inverse Document Frequency (IDF)

Problem: Not all terms are equally informative!

- "the" appears in almost every document → not useful for matching
- "arachnocentric" appears in very few documents → highly informative!

Solution: Give higher weight to rarer terms using **Inverse Document Frequency (IDF)**

IDF Formula

$$\mathrm{idf}_t = \log_{10} \frac{N}{\mathrm{df}_t}$$

where N = total documents in collection, df_t = documents containing term t

How it works:

Term type	**DF value**	**IDF value**
Rare term (appears in few docs)	Small df_t	**High IDF**
Common term (appears in many docs)	Large df_t	**Low IDF**
Universal term (in every doc: $\mathrm{df}_t = N$)	$\mathrm{df}_t = N$	**IDF = 0**

IDF: Worked Examples

Collection size: $N = 1,000,000$ **documents**

Calculate $\text{idf}_t = \log_{10} \frac{N}{\text{df}_t}$:

Term	df	Calculation	IDF
calpurnia	1	$\log_{10}(1,000,000/1)$	**6**
animal	100	$\log_{10}(1,000,000/100)$	**4**
sunday	1,000	$\log_{10}(1,000,000/1,000)$	**3**
fly	10,000	$\log_{10}(1,000,000/10,000)$	**2**
under	100,000	$\log_{10}(1,000,000/100,000)$	**1**
the	1,000,000	$\log_{10}(1,000,000/1,000,000)$	**0**

Key observation:

- Rare terms like "calpurnia" get **high IDF** (6)
- Common terms like "the" get **IDF = 0** (no discriminative value)

Document Frequency vs. Collection Frequency

Don't confuse these two measures!

	Collection Frequency (CF)	**Document Frequency (DF)**
Definition	Total times term appears across all documents	Number of documents containing the term
Example	"insurance" appears 10,440 times total	"insurance" appears in 3,997 documents

Why use DF instead of CF for IDF?

Consider these terms:

Term	**CF**	**DF**
INSURANCE	10,440	3,997
TRY	10,422	8,760

Same CF, but "INSURANCE" appears in *fewer* documents. It's more specific and informative. DF captures this; CF doesn't.

Section 5

TF-IDF Weighting

Combining TF and IDF

The TF-IDF Weight

We combine term frequency and inverse document frequency into one weight:

TF-IDF Formula

$$w_{t,d} = \underbrace{(1 + \log_{10} \mathrm{tf}_{t,d})}_{\text{TF component}} \times \underbrace{\log_{10} \frac{N}{\mathrm{df}_t}}_{\text{IDF component}}$$

What TF-IDF captures:

1. **TF component:** Terms appearing more often in a document are more important for that document
2. **IDF component:** Terms appearing in fewer documents are more discriminative

Result: High TF-IDF for terms that are *frequent in this document* but *rare across the collection*.

Understanding TF-IDF: When Is It High?

TF	**DF**	**TF-IDF**	**Interpretation**
High	Low	**High**	Term is important and distinctive
High	High	Medium	Term is frequent but not distinctive
Low	Low	Medium	Term is distinctive but not emphasised
Low	High	**Low**	Term is neither important nor distinctive
Any	$= N$	**Zero**	Term appears everywhere (no value)

Best case: A term appears many times in this document (high TF) but rarely in other documents (high IDF).

Worst case: A term appears everywhere (like "the"). Then IDF $= 0$, so TF-IDF $= 0$.

TF-IDF: Worked Example

Collection: 10,000 documents
Document *d*: Contains "algorithm" 15 times, "the" 50 times
DF: "algorithm" appears in 500 documents, "the" in 10,000

Calculate TF-IDF for "algorithm":

$$\begin{aligned}\text{TF-IDF}_{\text{algorithm},d} &= (1+\log_{10} 15) \times \log_{10} \frac{10,000}{500} \\ &= (1+1.18) \times \log_{10} 20 \\ &= 2.18 \times 1.30 = \mathbf{2.83}\end{aligned}$$

Calculate TF-IDF for "the":

$$\begin{aligned}\text{TF-IDF}_{\text{the},d} &= (1+\log_{10} 50) \times \log_{10} \frac{10,000}{10,000} \\ &= (1+1.70) \times \log_{10} 1 \\ &= 2.70 \times 0 = \mathbf{0}\end{aligned}$$

Result: "algorithm" gets a meaningful weight; "the" gets zero!

Section 6

TF-IDF in Practice

The TF-IDF Weighted Matrix

Instead of raw counts, we store TF-IDF weights:

	Doc 1	**Doc 2**	**Doc 3**	**Doc 4**	**Doc 5**
ANTHONY	5.25	3.18	0.0	0.0	0.35
BRUTUS	1.21	6.10	0.0	1.0	0.0
CAESAR	8.59	2.54	0.0	1.51	0.0
MERCY	1.51	0.0	1.90	0.12	0.88

Reading the table: Each cell = TF-IDF weight for that term in that document

- BRUTUS in Doc 2 has weight 6.10 (appears often in Doc 2, rare overall)
- BRUTUS in Doc 3 has weight 0.0 (doesn't appear at all)

Mathematical notation: Each document is now a **vector** of TF-IDF weights:

$$\vec{d} \in \mathbb{R}^{|V|}$$

This reads as: "document vector $\vec{d}$ is in the space of real numbers with $|V|$ dimensions"

- $\vec{d}$ = document represented as a vector (arrow notation indicates vector)
- $\mathbb{R}$ = real numbers (any decimal value like 5.25, 0.0, 8.59)
- $|V|$ = vocabulary size (total number of unique terms across all documents)

Example: If vocabulary has 10,000 unique words, each document is a vector with 10,000 dimensions!

Summary of Weighting Components

Component	**What It Measures**	**Formula**
Raw TF	Times term appears in doc	$\mathrm{tf}_{t,d}$
Log TF	Dampened term frequency	$1 + \log_{10}(\mathrm{tf}_{t,d})$
DF	Docs containing the term	df_t
IDF	Term's discriminative power	$\log_{10}(N/\mathrm{df}_t)$
TF-IDF	Combined weight	$(1 + \log_{10} \mathrm{tf}_{t,d}) \cdot \log_{10}(N/\mathrm{df}_t)$

Key Properties of TF-IDF

- **Increases** with term frequency in the document
- **Increases** with rarity of the term in the collection
- **Zero** if term doesn't appear in the document
- **Zero** if term appears in every document

Summary

Part 2: Key Takeaways

Part 2: Summary

What we learned:

1. **Term Frequency (TF):** Count how often a term appears in a document
2. **Log Frequency:** Use $1 + \log_{10}(\text{tf})$ to dampen high frequencies
3. **Document Frequency (DF):** Count how many documents contain a term
4. **Inverse Document Frequency (IDF):** $\log_{10}(N/\text{df})$: rare terms get higher weights
5. **TF-IDF:** Multiply TF and IDF to get a weight that captures both local importance and global rarity

Coming up in Part 3:

- The Vector Space Model
- Cosine similarity for document comparison
- Putting it all together: ranked retrieval with TF-IDF and cosine

Quick Reference: Formulas

Term Frequency (raw): $\text{tf}_{t,d} = \text{count of } t \text{ in } d$

Log Frequency Weight:

$$w_{t,d} = \begin{cases} 1 + \log_{10}(\text{tf}_{t,d}) & \text{if } \text{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Inverse Document Frequency:

$$\text{idf}_t = \log_{10} \frac{N}{\text{df}_t}$$

TF-IDF Weight:

$$\text{tf-idf}_{t,d} = (1 + \log_{10} \text{tf}_{t,d}) \times \log_{10} \frac{N}{\text{df}_t}$$

TF-IDF for Coursework: The Problem

Recall the matching task: Match Recommendation 3b against ALL responses

Using Jaccard: All words weighted equally

- "the", "and", "should" count the same as "training", "testimony", "healthcare"
- Common words dominate the similarity score
- Distinctive keywords get lost in the noise

One possible improvement: TF-IDF weighting

TF-IDF idea: Not all overlapping words are equally useful for matching!

- **Common words** like "recommendation", "government", "the" → **Low TF-IDF** (appear in most responses)
- **Distinctive words** like "training", "testimony", "healthcare" → **High TF-IDF** (rare, informative)

TF-IDF automatically down-weights common words and emphasises distinctive matches! **This is just one approach you could explore.**

TF-IDF Example: Recommendation 3b

Recommendation 3b: "They should look favourably upon putting together a package of **training materials**, with excerpts from oral and written **testimony**, to underpin what can happen in **healthcare**..."

Calculate TF-IDF scores for key words across all 58 response documents:

Word	DF (appears in X responses)	IDF	Informativeness
recommendation	58 (all)	$\log_{10}(58/58) = 0$	None (useless!)
government	55	$\log_{10}(58/55) = 0.02$	Very low
the	58 (all)	$\log_{10}(58/58) = 0$	None
training	5	$\log_{10}(58/5) = 1.06$	**High!**
testimony	2	$\log_{10}(58/2) = 1.46$	**Very high!**
healthcare	8	$\log_{10}(58/8) = 0.86$	**High!**

Result: When Response 3b contains "training", "testimony", "healthcare", these matches contribute MUCH more to the similarity score than generic words like "recommendation" or "government"

Note: Words like "accepted", "rejected", "partial" are used for *classification* (Task 2b), not for matching recommendations to responses.

Comparing Jaccard vs TF-IDF Matching

Scenario: Match Recommendation 3b to Response 3b (correct match) vs Response 1 (wrong match)

Response 3b: ”Accepted in full. NHS England, the nations’ blood services and UK SEBs are working together to map educational resources, including Technology Enhanced Learning and e-learning that deliver blood **transfusion training**...”

Response 1: ”This **recommendation** is accepted in full. The UK **Government** accepts this recommendation. The Victims and Prisoners Act provided...”

Method	**Rec 3b → Res 3b**	**Rec 3b → Res 1**
Jaccard (equal weighting)	Moderate overlap (generic + distinctive)	Moderate overlap (mostly generic words)
TF-IDF (smart weighting)	**HIGH score** (**training** match!)	**LOW score** (only generic matches)

What this example shows:

- TF-IDF can help distinguish relevant from irrelevant matches
- It’s particularly useful when distinctive keywords are present
- But it’s not the only approach - you might use other methods!

How to Implement TF-IDF (If You Choose This Approach)

Step-by-step approach for TF-IDF-based matching:

1. **Build document frequency (DF) counts**
   - Count how many response documents contain each word
   - Words in all 58 responses $\rightarrow$ IDF $=0$ (ignore them!)
   - Rare words $\rightarrow$ High IDF (valuable for matching!)
2. **Calculate TF-IDF vectors**
   - For each recommendation: compute TF-IDF for each word
   - For each response: compute TF-IDF for each word
   - Result: Each text represented as vector of weighted terms
3. **Calculate similarity**
   - Use cosine similarity between TF-IDF vectors (next lecture!)
   - Or sum TF-IDF scores for matching words
   - Rank responses by similarity score

Python libraries: `sklearn.feature_extraction.text.TfidfVectorizer`

Remember: This is one approach. You might find others work better for this task!

TF-IDF: One Tool in Your Toolkit

Jaccard Limitation	**What TF-IDF Offers**
Treats all words equally	Weights rare, distinctive words higher
"recommendation" and "testimony" contribute equally	"recommendation" $\rightarrow$ weight 0, "testimony" $\rightarrow$ high weight
Generic matches score high	Generic matches automatically down-weighted
Can't distinguish important from trivial overlaps	Emphasises meaningful keyword matches
Binary: word present or not	Considers frequency AND rarity

Possible progression for your coursework:

- Start with Jaccard as baseline (simple, interpretable)
- Consider TF-IDF weighting (handles common vs rare words)
- Explore semantic embeddings (handles synonyms/paraphrasing)
- Or try a completely different approach!

Key requirement: Document the approaches you tried, explain your reasoning, and show evidence of what worked and what didn't in your evaluation!

Questions?

Prof. Georgina Cosma

g.cosma@lboro.ac.uk