COP509

Natural Language Processing

Vector Space Model

Complete Worked Example with Shakespeare

Prof. Georgina Cosma
Department of Computer Science
Loughborough University

What This Lecture Covers

One complete example from start to finish.

We will work through **every calculation** step by step:

Part A: Build the Index (once)

1. Count term frequencies
2. Calculate TF (log frequencies)
3. Calculate IDF (rarity)
4. Combine into TF-IDF
5. Normalise document vectors

Part B: Answer a Query

6. Create query vector
7. Normalise query vector
8. Compute dot products
9. Rank the results

Key Insight

Weight = Local (TF) $\times$ Global (IDF) $\times$ Length Normalisation

The Problem We Are Solving

Task

Given a **query** and a **collection of documents**, rank the documents from most relevant to least relevant.

Our Example:

Query: "BRUTUS CAESAR"

Documents: Three Shakespeare plays

- d_1: *Julius Caesar*
- d_2: *Anthony and Cleopatra*
- d_3: *The Tempest*

Question: Which play is most relevant to the query "BRUTUS CAESAR"?
Expected answer: Julius Caesar (it's literally about Brutus and Caesar!)

Part A

Building the Index

(Done once when documents are added to the system)

Step 1

Count Term Frequencies

How many times does each word appear in each document?

We count how many times each word appears in each document:

Term	Julius Caesar	Ant & Cleo	Tempest
BRUTUS	40	5	0
CAESAR	50	30	0
MERCY	2	5	8

What this tells us:

- BRUTUS appears 40 times in Julius Caesar, 5 times in Ant & Cleo, 0 times in Tempest
- CAESAR appears 50 times in Julius Caesar, 30 times in Ant & Cleo
- MERCY appears in all three plays

Step 2

Calculate Term Frequency (TF)

Use logarithms to dampen the raw counts

Problem: A word appearing 100 times is NOT 100 times more important than a word appearing once.

Example: If "CAESAR" appears 100 times and "BRUTUS" appears 10 times, is CAESAR really 10× more important? Probably not!

Solution: Use **logarithms** to "dampen" large counts.

What Logarithms Do

Raw Count	$\log_{10}$**(count)**	**TF = 1 + log**
1	0	1.00
10	1	2.00
100	2	3.00
1000	3	4.00

A 1000× increase in count only gives a 4× increase in weight!

Term Frequency Formula

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

Why add 1? So that a word appearing once gets weight 1, not 0.

Let's calculate TF for BRUTUS:

Julius Caesar (count = 40): $\text{tf} = 1 + \log_{10}(40) = 1 + 1.60 = \mathbf{2.60}$

Ant & Cleo (count = 5): $\text{tf} = 1 + \log_{10}(5) = 1 + 0.70 = \mathbf{1.70}$

Tempest (count = 0): $\text{tf} = \mathbf{0}$

Applying the formula to all terms:

CAESAR:

- Julius Caesar: $1 + \log_{10}(50) = 1 + 1.70 = 2.70$
- Ant & Cleo: $1 + \log_{10}(30) = 1 + 1.48 = 2.48$
- Tempest: 0

MERCY:

- Julius Caesar: $1 + \log_{10}(2) = 1 + 0.30 = 1.30$
- Ant & Cleo: $1 + \log_{10}(5) = 1 + 0.70 = 1.70$
- Tempest: $1 + \log_{10}(8) = 1 + 0.90 = 1.90$

Term	Julius Caesar	Ant & Cleo	Tempest
BRUTUS	2.60	1.70	0
CAESAR	2.70	2.48	0
MERCY	1.30	1.70	1.90

This is the LOCAL component.
It tells us how important each term is **within each document**.

But there's a problem: MERCY appears in **all** documents, so it's not useful for distinguishing between them!

Step 3

Calculate Inverse Document Frequency (IDF)

Rare words are more useful for finding relevant documents

Step 3: Why Do We Need IDF?

Problem: Some words appear in many documents (common), others appear in few (rare).

Key Insight: **Rare words are more useful** for finding relevant documents!

In our example:

- BRUTUS appears in **2 out of 3** documents – fairly rare
- CAESAR appears in **2 out of 3** documents – fairly rare
- MERCY appears in **3 out of 3** documents – not useful!

Think about it: If a word appears in EVERY document, it can't help us distinguish between them. It's like searching for "the" – useless!

Step 3: Document Frequency (df)

Document Frequency = number of documents containing the term

Term	df	Which documents?
BRUTUS	2	Julius Caesar, Ant & Cleo
CAESAR	2	Julius Caesar, Ant & Cleo
MERCY	3	Julius Caesar, Ant & Cleo, Tempest

Note: We count **how many documents** contain the term, not how many times it appears total.

Inverse Document Frequency Formula

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right)$$

where:

- N = total number of documents (in our case, $N = 3$)
- df_t = number of documents containing term t

What this does:

- If df is small (rare word) $\rightarrow$ N/df is large $\rightarrow$ IDF is large
- If df is large (common word) $\rightarrow$ N/df is small $\rightarrow$ IDF is small
- If df = N (word in every doc) $\rightarrow$ $N/\text{df} = 1 \rightarrow$ IDF = 0!

Step 3: Calculate IDF Values

$N = 3$ (we have 3 documents)

BRUTUS: df $= 2$

$$\text{idf}_{\text{BRUTUS}} = \log_{10}\left(\frac{3}{2}\right) = \log_{10}(1.5) = \mathbf{0.18}$$

CAESAR: df $= 2$

$$\text{idf}_{\text{CAESAR}} = \log_{10}\left(\frac{3}{2}\right) = \log_{10}(1.5) = \mathbf{0.18}$$

MERCY: df $= 3$ (appears in ALL documents!)

$$\text{idf}_{\text{MERCY}} = \log_{10}\left(\frac{3}{3}\right) = \log_{10}(1) = \mathbf{0}$$

Term	df	N/df	IDF
BRUTUS	2	$3/2 = 1.5$	**0.18**
CAESAR	2	$3/2 = 1.5$	**0.18**
MERCY	3	$3/3 = 1.0$	**0**

Key Insight

MERCY has IDF = 0 because it appears in **every** document!
A word that appears everywhere has **zero discriminating power**.

Rule: Higher IDF = rarer word = more useful for finding relevant documents

Step 4

Calculate TF-IDF Weights

Multiply TF (local) by IDF (global)

TF-IDF Formula

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$$

This is: **Local** $\times$ **Global**

What this achieves:

- High weight if term is frequent in THIS document (high TF)
- High weight if term is rare ACROSS documents (high IDF)
- Low weight if term is in every document (IDF $= 0 \rightarrow$ weight $= 0$)

Step 4: Calculate TF-IDF for BRUTUS

BRUTUS (IDF $= 0.18$)

Julius Caesar:	tf-idf $= 2.60 \times 0.18 =$ **0.47**
Ant & Cleo:	tf-idf $= 1.70 \times 0.18 =$ **0.31**
Tempest:	tf-idf $= 0 \times 0.18 =$ **0**

Note: The Tempest gets 0 because BRUTUS doesn't appear there (TF $= 0$).

Step 4: Calculate TF-IDF for CAESAR and MERCY

CAESAR (IDF = 0.18)

Julius Caesar:	$2.70 \times 0.18 = \mathbf{0.49}$
Ant & Cleo:	$2.48 \times 0.18 = \mathbf{0.45}$
Tempest:	$0 \times 0.18 = \mathbf{0}$

MERCY (IDF = 0)

Julius Caesar:	$1.30 \times 0 = \mathbf{0}$
Ant & Cleo:	$1.70 \times 0 = \mathbf{0}$
Tempest:	$1.90 \times 0 = \mathbf{0}$

MERCY is 0 everywhere because it appears in all documents (IDF = 0)!

Term	Julius Caesar	Ant & Cleo	Tempest
BRUTUS	**0.47**	0.31	0
CAESAR	**0.49**	0.45	0
MERCY	0	0	0

This is our TF-IDF Weight Matrix!

Observations:

- Julius Caesar has the highest weights for BRUTUS and CAESAR
- MERCY is useless (all zeros) – it can't help us distinguish documents
- The Tempest has all zeros for our query terms

Step 5

Normalise Document Vectors

Make all documents comparable regardless of length

Each column of the TF-IDF matrix is a **document vector**:

Vector format: (BRUTUS weight, CAESAR weight, MERCY weight)

$$\vec{d}_1 \text{ (Julius Caesar)} = (0.47, 0.49, 0)$$

$$\vec{d}_2 \text{ (Ant \& Cleo)} = (0.31, 0.45, 0)$$

$$\vec{d}_3 \text{ (Tempest)} = (0, 0, 0)$$

What is a vector? Just an ordered list of numbers!
Each number represents the importance of that term in that document.

Problem: Longer documents have more words, so they might get higher scores unfairly.

Solution: Divide each vector by its **length** so all vectors have length 1.

Vector Length Formula

$$|\vec{d}| = \sqrt{d_1^2 + d_2^2 + d_3^2 + \dots}$$

This is Pythagoras' theorem extended to multiple dimensions!

Simple example: If $\vec{v} = (3, 4)$

$$|\vec{v}| = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

Julius Caesar: $\vec{d}_1 = (0.47, 0.49, 0)$

$$|\vec{d}_1| = \sqrt{0.47^2 + 0.49^2 + 0^2} = \sqrt{0.22 + 0.24 + 0} = \sqrt{0.46} = \mathbf{0.68}$$

Anthony & Cleopatra: $\vec{d}_2 = (0.31, 0.45, 0)$

$$|\vec{d}_2| = \sqrt{0.31^2 + 0.45^2 + 0^2} = \sqrt{0.10 + 0.20 + 0} = \sqrt{0.30} = \mathbf{0.55}$$

Tempest: $\vec{d}_3 = (0, 0, 0)$

$$|\vec{d}_3| = \sqrt{0^2 + 0^2 + 0^2} = \mathbf{0}$$

Normalisation formula: Divide each component by the length.

$$\hat{d} = \frac{\vec{d}}{|\vec{d}|} = \left(\frac{d_1}{|\vec{d}|}, \frac{d_2}{|\vec{d}|}, \frac{d_3}{|\vec{d}|} \right)$$

Julius Caesar:

$$\hat{d}_1 = \frac{(0.47, 0.49, 0)}{0.68} = \left(\frac{0.47}{0.68}, \frac{0.49}{0.68}, \frac{0}{0.68} \right) = (\mathbf{0.69}, \mathbf{0.72}, \mathbf{0})$$

Verify length = 1:

$$\sqrt{0.69^2 + 0.72^2 + 0^2} = \sqrt{0.48 + 0.52} = \sqrt{1.0} = 1 \quad \checkmark$$

Step 5: All Normalised Document Vectors

Before normalisation:

$$\vec{d}_1 = (0.47, 0.49, 0) \quad |\vec{d}_1| = 0.68$$
$$\vec{d}_2 = (0.31, 0.45, 0) \quad |\vec{d}_2| = 0.55$$
$$\vec{d}_3 = (0, 0, 0) \qquad |\vec{d}_3| = 0$$

After normalisation (divide by length):

$$\hat{d}_1 \text{ (Julius Caesar)} = (\mathbf{0.69}, \mathbf{0.72}, \mathbf{0})$$

$$\hat{d}_2 \text{ (Ant \& Cleo)} = (\mathbf{0.57}, \mathbf{0.82}, \mathbf{0})$$

$$\hat{d}_3 \text{ (Tempest)} = (\mathbf{0}, \mathbf{0}, \mathbf{0})$$

Now all vectors have length 1! (Called "unit vectors")

Part B

Answering a Query

(Done every time a user searches)

Step 6

Create the Query Vector

Represent the query in the same way as documents

Query: "BRUTUS CAESAR"

The query becomes a vector in the **same space** as the documents.

Simple approach: Give weight 1 to each query term, 0 to others.

$$\vec{q} = (\underbrace{1}_{\text{BRUTUS}}, \underbrace{1}_{\text{CAESAR}}, \underbrace{0}_{\text{MERCY}}) = (1, 1, 0)$$

Why this works: We want to find documents where BRUTUS and CAESAR are important. By giving them equal weight (1), we're saying they're equally important in the query.

Step 7

Normalise the Query Vector

Same process as for documents

Query vector: $\vec{q} = (1, 1, 0)$

Step 1: Calculate length

$$|\vec{q}| = \sqrt{1^2 + 1^2 + 0^2} = \sqrt{1 + 1 + 0} = \sqrt{2} = \mathbf{1.41}$$

Step 2: Divide by length

$$\hat{q} = \frac{(1, 1, 0)}{1.41} = \left(\frac{1}{1.41}, \frac{1}{1.41}, \frac{0}{1.41}\right) = (\mathbf{0.71}, \mathbf{0.71}, \mathbf{0})$$

Verify: $\sqrt{0.71^2 + 0.71^2 + 0^2} = \sqrt{0.50 + 0.50} = 1$ ✓

Step 8

Compute Dot Products

For normalised vectors, dot product = similarity!

Step 8: What is a Dot Product?

The **dot product** multiplies corresponding elements and adds them up.

Dot Product Formula

$$\vec{a} \cdot \vec{b} = a_1 \times b_1 + a_2 \times b_2 + a_3 \times b_3$$

Example:

$$(2, 3, 1) \cdot (4, 0, 2) = (2 \times 4) + (3 \times 0) + (1 \times 2) = 8 + 0 + 2 = 10$$

Key Insight

For **unit vectors** (length $= 1$), the dot product equals the **cosine similarity**!
This means: **dot product = how similar the vectors are**

Normalised query: $\hat{q} = (0.71, 0.71, 0)$

Julius Caesar: $\hat{d}_1 = (0.69, 0.72, 0)$

$$\begin{aligned}\hat{q} \cdot \hat{d}_1 &= (0.71 \times 0.69) + (0.71 \times 0.72) + (0 \times 0) \\ &= 0.49 + 0.51 + 0 = \mathbf{1.00}\end{aligned}$$

Ant & Cleo: $\hat{d}_2 = (0.57, 0.82, 0)$

$$\begin{aligned}\hat{q} \cdot \hat{d}_2 &= (0.71 \times 0.57) + (0.71 \times 0.82) + (0 \times 0) \\ &= 0.40 + 0.58 + 0 = \mathbf{0.99}\end{aligned}$$

Tempest: $\hat{d}_3 = (0, 0, 0)$

$$\hat{q} \cdot \hat{d}_3 = 0 + 0 + 0 = \mathbf{0}$$

Step 9

Rank the Documents

Highest similarity wins!

Query: "BRUTUS CAESAR"

Rank	Document	Similarity Score
1	Julius Caesar	**1.00**
2	Anthony & Cleopatra	0.99
3	The Tempest	0.00

Julius Caesar wins! This makes sense because:

- It has the highest TF-IDF weights for both BRUTUS and CAESAR
- The play is literally about these characters!

The Tempest scores 0 because it contains neither query term.

Summary

Everything in One Place

Why Pre-Normalisation is Smart

The old way:
At query time, calculate:

$$\text{similarity} = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}| \times |\vec{d}|}$$

Problem: Must calculate $|\vec{d}|$ for every document, every time!

The smart way:
Pre-normalise documents once:

$$\hat{d} = \frac{\vec{d}}{|\vec{d}|}$$

Then at query time:

$$\text{similarity} = \hat{q} \cdot \hat{d}$$

Just multiply and add – no square roots!

Why It Works

$$\frac{\vec{q} \cdot \vec{d}}{|\vec{q}| \times |\vec{d}|} = \frac{\vec{q}}{|\vec{q}|} \cdot \frac{\vec{d}}{|\vec{d}|} = \hat{q} \cdot \hat{d}$$

Same answer, much faster!

The Complete Process: Summary

Step	Name	What You Do
1	Raw Counts	Count term occurrences in each document
2	TF	$\text{tf} = 1 + \log_{10}(\text{count})$
3	IDF	$\text{idf} = \log_{10}(N/\text{df})$
4	TF-IDF	$\text{weight} = \text{tf} \times \text{idf}$
5	Normalise Docs	$\hat{d} = \vec{d}/\lvert\vec{d}\rvert$
— Index built. Now answer queries: —		
6	Query Vector	Weight 1 for each query term
7	Normalise Query	$\hat{q} = \vec{q}/\lvert\vec{q}\rvert$
8	Dot Products	$\text{score} = \hat{q} \cdot \hat{d}$ for each document
9	Rank	Sort documents by score (highest first)

Key Formulas

1. **Term Frequency (Local importance):**

$$\text{tf}_{t,d} = 1 + \log_{10}(\text{count}) \quad \text{or } 0 \text{ if count} = 0$$

2. **Inverse Document Frequency (Global importance):**

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right)$$

3. **TF-IDF Weight:**

$$\text{tf-idf}_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$$

4. **Vector Length:**

$$|\vec{d}| = \sqrt{\sum_i d_i^2}$$

5. **Similarity (for normalised vectors):**

$$\text{similarity} = \hat{q} \cdot \hat{d} = \sum \hat{q}_i \times \hat{d}_i$$

Remember This!

$$\textbf{Weight} = \text{Local (TF)} \times \text{Global (IDF)} \times \text{Length Norm}$$

- **TF (Local):** Terms appearing often in this document are important
- **IDF (Global):** Rare terms are more useful for finding relevant documents
- **Length Norm:** Makes fair comparison regardless of document size

Pre-normalise documents $\rightarrow$ Dot product = Cosine similarity

Questions?

Prof. Georgina Cosma

g.cosma@lboro.ac.uk