COP509

Natural Language Processing

The Vector Space Model

Part 1: Introduction to Ranked Retrieval

Prof. Georgina Cosma

Department of Computer Science
Loughborough University

g.cosma@lboro.ac.uk

January 2026

Learning Outcomes

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

1. Explain why we need ranked retrieval instead of Boolean retrieval
2. Understand the concept of document-query scoring
3. Calculate the Jaccard coefficient for document similarity
4. Identify the limitations of the Jaccard coefficient
5. Understand why we need more sophisticated scoring methods

Prerequisites

This lecture assumes you understand:

- Boolean retrieval (AND, OR, NOT queries)
- Basic text preprocessing (tokenisation, stop words)
- Set operations (union, intersection)

Key notation used in this lecture:

Symbol	Meaning
A, B	Sets (collections of unique items)
$A \cap B$	**Intersection**: items in **both** A and B
$A \cup B$	**Union**: items in A **or** B (or both)
$\|A\|$	**Size** of set A (number of elements it contains)
$\frac{a}{b}$	Division: a divided by b
$[0, 1]$	A range of values from 0 to 1 (inclusive)

Example: If $A = \{\text{cat, dog, fish}\}$ and $B = \{\text{dog, bird}\}$

- $A \cap B = \{\text{dog}\}$ (only "dog" is in both)
- $A \cup B = \{\text{cat, dog, fish, bird}\}$ (all unique items)
- $|A| = 3$, $|B| = 2$, $|A \cap B| = 1$, $|A \cup B| = 4$

Outline for Part 1

1. The Problem with Boolean Retrieval
2. Introduction to Ranked Retrieval
3. Document-Query Scoring
4. The Jaccard Coefficient
5. Limitations of Jaccard
6. Preview: What We Need Next

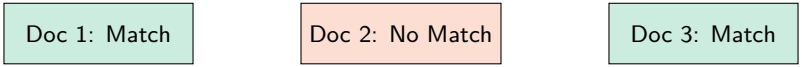
Section 1

The Problem with Boolean Retrieval

In **Boolean retrieval**, we learned that:

- Queries use Boolean operators: `AND`, `OR`, `NOT`
- Documents either **match** or **don't match** a query
- Results are **unordered**: all matching documents are "equally relevant"

Example query: "machine AND learning AND NOT deep"



Problem: Doc 1 and Doc 3 are treated as *equally relevant*, but are they really?

Why Boolean Retrieval Falls Short

Boolean retrieval works well for:

- Expert users who know exactly what they want
- Small, well-organised collections
- Precise legal or patent searches

But it fails for everyday users because:

Too many results:

- Query "artificial intelligence" might return 50,000 documents
- All are "relevant" according to Boolean logic
- User must wade through all of them

Too few results:

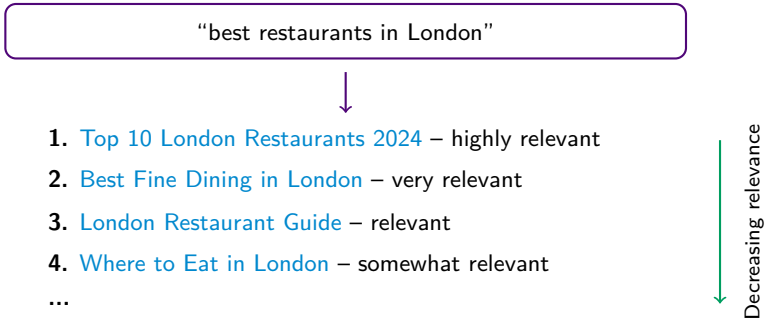
- Query "quantum machine learning applications healthcare" might return 0 documents
- User must reformulate the query
- Frustrating trial-and-error process

The Real Problem

Most users cannot write Boolean queries and don't want to see thousands of unranked results. This is especially true for **web search**.

What Users Actually Want

When you search Google, what do you expect?



Users expect:

- Results **ranked by relevance**: best results first
- A **small number** of highly relevant results (top 10)
- **No Boolean operators** required: just type natural language

Section 2

Introduction to Ranked Retrieval

The Solution: Ranked Retrieval

Instead of Boolean matching, we use **ranked retrieval**:

Key Idea

Assign a **relevance score** to each document for a given query, then **rank documents by score** (highest first).

Boolean Retrieval:

- Binary decision: match or no match
- Results: $\{0, 1\}$
- No ordering of results

Ranked Retrieval:

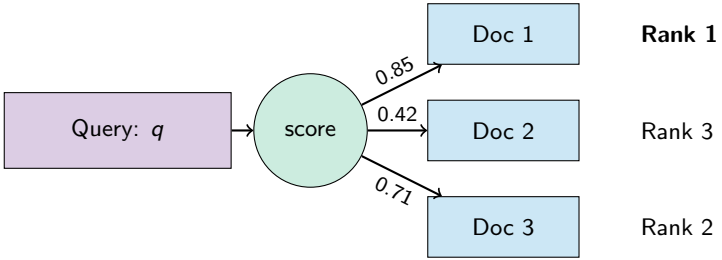
- Continuous score: how well does it match?
- Results: $[0, 1]$ (or any range)
- Documents sorted by score

The fundamental question: How do we compute a score that measures how well a document matches a query?

Document-Query Scoring: The Basic Idea

We need a **scoring function** score(q, d) where:

- q = the query (what the user is searching for)
- d = a document in our collection
- Output = a number indicating relevance (higher = more relevant)



Goal: Design a scoring function where documents that are more relevant to the query get higher scores.

What Makes a Good Scoring Function?

A good scoring function should capture these intuitions:

1. **Term presence:** If a query term appears in a document, the document should get a higher score
2. **Term frequency:** If a query term appears *many times* in a document, the document is probably more relevant
3. **Term importance:** Rare terms (like "arachnocentric") should matter more than common terms (like "the")
4. **Document length:** A short document with query terms should not be unfairly penalised compared to a long document

Our Journey

We'll build up to the **Vector Space Model** by starting simple and gradually addressing each of these requirements.

Section 3

The Jaccard Coefficient

A Simple First Attempt

Our First Scoring Function: Jaccard Coefficient

Let's start with the simplest approach: measure the **overlap** between query and document terms.

Jaccard Coefficient

Given two sets A and B, the Jaccard coefficient is:

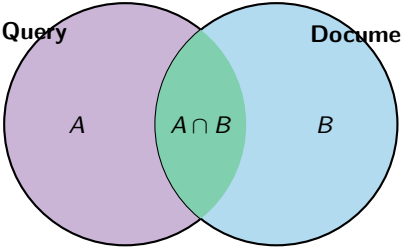
$$\text{JACCARD}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

where $|A \cap B|$ is the size of the **intersection** and $|A \cup B|$ is the size of the **union**.

For document retrieval:

- A = set of unique terms in the **query**
- B = set of unique terms in the **document**
- Jaccard measures: "What fraction of all terms appear in both?"

Understanding Jaccard: Visual Explanation



$$\text{JACCARD} = \frac{\text{Green area (intersection)}}{\text{Total coloured area (union)}}$$

Properties of Jaccard:

- JACCARD$(A, B) = 1$ if and only if $A = B$ (only identical sets have maximum similarity)
- JACCARD$(A, B) = 0$ if A and B share no terms
- Always returns a value between 0 and 1

Jaccard Coefficient: Worked Example 1

Query: "flowers of March"
Document: "Caesar died in March"

Step 1: Convert to sets of terms

- Query terms: {flowers, of, March}
- Document terms: {Caesar, died, in, March}

Step 2: Find intersection and union

- Intersection: {**March**} $|A \cap B| = 1$
- Union: {flowers, of, March, Caesar, died, in} $|A \cup B| = 6$

Step 3: Calculate Jaccard

$$\text{JACCARD}(q, d) = \frac{|A \cap B|}{|A \cup B|} = \frac{1}{6} \approx \mathbf{0.167}$$

Interpretation: The query and document share only 1 out of 6 total unique terms, so similarity is low.

Jaccard Coefficient: Worked Example 2

Query: "information on cars"
Document: "all you've ever wanted to know about cars"

Step 1: Convert to sets of terms

- Query terms: {information, on, cars}
- Document terms: {all, you've, ever, wanted, to, know, about, cars}

Step 2: Find intersection and union

- Intersection: {**cars**} $|A \cap B| = 1$
- Union: 10 unique terms $|A \cup B| = 10$

Step 3: Calculate Jaccard

$$\text{JACCARD}(q, d) = \frac{1}{10} = \mathbf{0.10}$$

Note: Even though this document is *about* cars, the Jaccard score is low because only one term matches exactly.

Your Turn: Exercise

Calculate the Jaccard coefficient for:

Query: "information on cars"
Document: "information on trucks, information on planes, information on trains"

Hint: Remember that sets contain only *unique* terms!

Your Turn: Exercise

Calculate the Jaccard coefficient for:

Query: "information on cars"
Document: "information on trucks, information on planes, information on trains"

Hint: Remember that sets contain only *unique* terms!

Solution:

- Query terms: {information, on, cars} (3 terms)
- Document terms: {information, on, trucks, planes, trains} (5 unique terms)
- Intersection: {information, on} (2 terms)
- Union: {information, on, cars, trucks, planes, trains} (6 terms)

$$\text{JACCARD}(q, d) = \frac{2}{6} = \frac{1}{3} \approx \mathbf{0.333}$$

Section 4

Limitations of Jaccard

Why We Need Something Better

Problem 1: Jaccard Ignores Term Frequency

Consider these two documents for query "machine learning":

Document A:
"This paper discusses **machine learning** briefly among other topics like databases and networks."

"machine learning" appears: ***1 time***

Document B:
"**Machine learning** is transforming AI. Deep **learning**, a subset of **machine learning**, enables **learning** from data."

"machine" appears: ***2 times***
"learning" appears: ***4 times***

Jaccard treats both documents equally!

Both documents contain the terms "machine" and "learning", so Jaccard sees no difference. But intuitively, Document B is *more* about machine learning.

Limitation: Jaccard uses sets, so it only cares about *presence*, not *frequency*.

Problem 2: Jaccard Ignores Term Importance

Consider query: "the machine"

Document A:
"**The** quick brown fox jumps over **the** lazy dog."

Contains: "the" ✓, "machine" ×

Document B:
"This **machine** learning algorithm is fast."

Contains: "the" ×, "machine" ✓

Both documents match exactly one query term!

But which match is more valuable?

- "the" appears in almost every document: **not informative**
- "machine" is relatively rare: **highly informative**

Limitation: Jaccard treats all terms as equally important, but rare terms should matter more.

Problem 3: Jaccard Has Document Length Issues

Consider query: "machine learning"

Short Document:
"Machine learning is useful."

- Terms: {machine, learning, is, useful}
- Intersection: {machine, learning}
- Union: 4 terms
- Jaccard: $\frac{2}{4} = 0.50$

Long Document:
"Machine learning is a field of AI that uses statistical methods to enable computers to learn from data without explicit programming..." (100 words)

- Intersection: {machine, learning}
- Union: 50 unique terms
- Jaccard: $\frac{2}{50} = 0.04$

The long document might be more comprehensive, but gets a lower score!

Limitation: Jaccard penalises longer documents unfairly.

Summary: What Jaccard Gets Wrong

Limitation	What We Need Instead
Ignores term frequency	Words appearing more often should increase relevance
Treats all terms equally	Rare terms should be weighted more heavily
Poor document length handling	Need proper normalisation for document length

The Solution: Vector Space Model

In the next parts, we will develop the **Vector Space Model**, which addresses all these limitations using:

- **Term Frequency (TF)**: count how often terms appear
- **Inverse Document Frequency (IDF)**: weight rare terms higher
- **Cosine Similarity**: properly normalise for document length

Summary

Part 1: Key Takeaways

Part 1: Summary

What we learned:

1. **Boolean retrieval** returns unranked results, which is not user-friendly for large collections
2. **Ranked retrieval** assigns scores to documents and orders them by relevance
3. The **Jaccard coefficient** is a simple scoring function:

$$\text{JACCARD}(q, d) = \frac{|q \cap d|}{|q \cup d|}$$

4. Jaccard has limitations: ignores term frequency, term importance, and document length

Coming up in Part 2:

- Term Frequency (TF) and log scaling
- Inverse Document Frequency (IDF)
- The TF-IDF weighting scheme

Quick Reference: Jaccard Coefficient

Jaccard Coefficient Formula:

$$\text{JACCARD}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

How to calculate:

1. Convert query and document to **sets of unique terms**
2. Find the **intersection** (terms in both)
3. Find the **union** (all terms from both)
4. Divide intersection size by union size

Properties:

- Range: $[0, 1]$
- 0 = no overlap, 1 = identical

The Matching Problem in Your Coursework

Your task: Match each recommendation to its response

Example setup:
You have Recommendation 2a and need to find which response it matches.

> **Recommendation 2a:**
> "A permanent memorial be established in the UK and consideration be given to memorials in each of Northern Ireland, Wales and Scotland..."

You need to check it against ALL responses:

- Response 1 (about compensation schemes)
- Response 2a (about memorials) ← Which one matches?
- Response 2b (about Treloar's memorial)
- Response 3a (about medical training)
- ... and 54 more responses

Using Jaccard to Find the Match

Step 1: Calculate Jaccard similarity between Rec 2a and each response

Comparison	**Jaccard Score**
Jaccard(Rec 2a, Response 1)	0.05 (few shared words)
Jaccard(Rec 2a, Response 2a)	**0.18 (highest!)**
Jaccard(Rec 2a, Response 2b)	0.03
Jaccard(Rec 2a, Response 3a)	0.06
...	...

Step 2: Pick the response with the highest Jaccard score

Result: Response 2a has the highest score $\rightarrow$ it's the match! ✓

Why did it work? Let's look at the shared words...

Why Jaccard Worked for Rec 2a

Recommendation 2a:
"A permanent **memorial** be established in the **UK** and consideration be given to **memorials** in each of **Northern Ireland**, **Wales** and **Scotland**..."

Response 2a:
"This recommendation is accepted in full by the **UK** Government, the **Scottish** Government, the **Welsh** Government and the **Northern Ireland** Executive. A steering committee is being formed to decide what **memorial(s)**..."

Shared words: memorial, memorials, UK, Northern Ireland, Wales/Welsh, Scotland/Scottish

These specific place names and the word "memorial" gave enough overlap for Jaccard to identify the correct match!

When Jaccard Fails: Wrong Match Example

Now try matching Recommendation 2b:

Recommendation 2b:
"A **memorial** be established at public expense, dedicated specifically to the **children infected** at **Treloar's School**..."

Response 2b (correct match):
"This recommendation is accepted in full by the UK Government, the Scottish Government, the Welsh Government and the Northern Ireland Executive."

Problem: Only "memorial" overlaps! The response doesn't mention "children", "Treloar's", or "school"

Risk: Jaccard might wrongly match this to Response 2a instead (which mentions "memorial" many times)

Why? Response 2b is very short and generic - relies on the numbering system to indicate the match, not content

Key Takeaway for Your Coursework

When Jaccard helps:

- When rec and response share specific keywords (place names, technical terms, unique words)
- As a first-pass filter to narrow down candidates
- To verify that structurally-matched pairs actually discuss the same topic

When Jaccard fails:

- Paraphrasing: "children infected" $\rightarrow$ "young people affected"
- Synonyms: "memorial" vs "monument", "UK" vs "Britain"
- Generic responses with few content words
- Multiple recommendations addressed in one response

Better approaches:

- Combine with structural matching (numbers/IDs)
- Use TF-IDF to weight important words more
- Use semantic embeddings (BERT, sentence transformers)
- Implement preprocessing (stemming, stop word removal)

Questions?

Prof. Georgina Cosma

g.cosma@lboro.ac.uk