


- ↳ Term Incidence
- ↳ Inverted Index
- ↳ boolean query
- ↳ 
- ↳ now need to give
Score, Rank the Term:

Information Retrieval

Scoring, Term Weighting and the Vector
Space Model



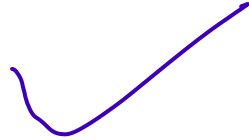
This lecture; IIR Sections 6.2-6.4.3

- Ranked retrieval ✓
- Scoring documents ✓
- Term frequency ✓
- Collection statistics ✓
- Weighting schemes ✓
- Vector space scoring ✓

Ranked retrieval ✓ disabled /s wheel chair

- Thus far, our queries have all been Boolean.
 - Documents either match or don't.
- Good for expert users with precise understanding of their needs and the collection.
 - Also good for applications: Applications can easily consume 1000s of results.
- Not good for the majority of users. ✓
 - Most users incapable of writing Boolean queries (or they are, but they think it's too much work).
 - Most users don't want to wade through 1000s of results.
 - This is particularly true of web search.

Problem with Boolean search: feast or famine

- Boolean queries often result in either too few (=0) or too many (1000s) results.
 - Query 1: "standard user dlink 650" → 200,000 hits
 - Query 2: "standard user dlink 650 no card found": 0 hits
 - It takes a lot of skill to come up with a query that produces a manageable number of hits.
 - AND gives too few; OR gives too many
- 

Ranked retrieval models

- Rather than a set of documents satisfying a query expression, in **ranked retrieval**, the system returns an ordering over the (top) documents in the collection for a query
- **Free text queries**: Rather than a query language of operators and expressions, the user's query is just one or more words in a human language
- In principle, there are two separate choices here, but in practice, ranked retrieval has normally been associated with free text queries and vice versa

Feast or famine: not a problem in ranked retrieval

- When a system produces a ranked result set, large result sets are not an issue
 - Indeed, the size of the result set is not an issue
 - We just show the top k (≈ 10) results
 - We don't overwhelm the user
- Premise: the ranking algorithm works

Scoring as the basis of ranked retrieval

- We wish to return in order the documents most likely to be useful to the searcher
- How can we rank-order the documents in the collection with respect to a query?
- Assign a score – say in $[0, 1]$ – to each document
- This score measures how well document and query “match”.

Take 1: Jaccard coefficient

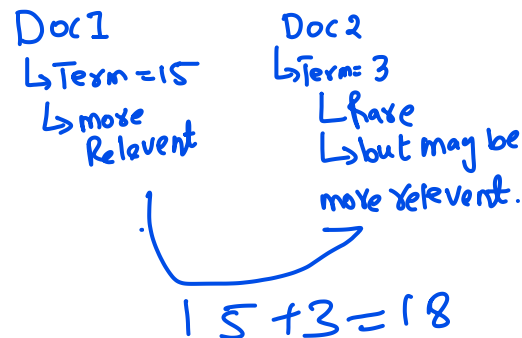
- A common measure of overlap of two sets A and B
- $\text{jaccard}(A, B) = |A \cap B| / |A \cup B|$
- $\text{jaccard}(A, A) = 1$
*↳ better way
↳ length*
- $\text{jaccard}(A, B) = 0$ if $A \cap B = 0$
- A and B don't have to be the same size.
- Always assigns a number between 0 and 1.

Jaccard coefficient: Scoring example

- What is the query-document match score that the Jaccard coefficient computes for each of the two documents below?
- Query: *ides of march*
- Document 1: *caesar died in march* $\frac{1}{8}$
- Document 2: *the long march* $\frac{1}{5}$

Issues with Jaccard for scoring


- It doesn't consider **term frequency** (how many times a term occurs in a document)
- **Rare terms** in a collection are more informative than frequent terms. Jaccard doesn't consider this information
- We need a more sophisticated way of normalizing for length



Query-document matching scores

- We need a way of assigning a score to a query/document pair
- **Let's start with a one-term query**
- If the query term does not occur in the document: score should be 0
- **The more frequent the query term in the document, the higher the score (should be)**
- We will look at a number of alternatives for this.

Recall (Lecture 2): Binary term-document incidence matrix



	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Each document is represented by a binary vector $\in \{0,1\}^{|V|}$

Term-document count matrices

- Consider the number of occurrences of a term in a document:

- Each document is a **count vector** in \mathbb{N}^v : a column below

$$Tf = \frac{157}{454} = 0.34$$

$$1 + \log(0.34) = 0.53 \times \log_{10}\left(\frac{2}{6}\right)$$

there is no
positional
information.

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Bag of words model

- Vector representation doesn't consider the ordering of words in a document
- *John is quicker than Mary* and *Mary is quicker than John* have the same vectors
- This is called the bag of words model.
- In a sense, this is a step back: The positional index was able to distinguish these two documents.

	Doc1	Doc2
John	1	1
Quicker	1	1
than	1	1
Mary	1	1
<u>No positional Information</u>		

Doc 1
School 1
Doc 2
10

Term frequency **tf**

- The term frequency $tf_{t,d}$ of term t in document d is defined as the number of times that t occurs in d .
 - Note: Frequency means count in IR
- We want to use **tf** when computing query-document match scores. But how?
- Raw term frequency is not what we want:
 - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
 - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Log-frequency weighting

- The log frequency weight of term t in d is

$$W_{t,d} = \begin{cases} 1 + \log_{10} \underline{tf_{t,d}} , & \text{if } tf_{t,d} > 0 \\ 0, & \text{Otherwise} \end{cases}$$

- $0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 1.3, 10 \rightarrow 2, 1000 \rightarrow 4$, etc.
- Score for a document-query pair: sum over terms t in both q and d :
Information good Subject

- $\text{Score} = \sum_{t \in q \cap d} (1 + \log_{10} tf_{t,d})$
 $q \cap d = \{\text{Information, good}\}$

- The score is 0 if none of the query terms is present in the document.
*Step 1: Make Term freq Matrix
 Step 2: Calculate Score of each Term from Matrix:*

Rare terms are more informative

- Rare terms are more informative than frequent terms
 - Recall stop words
- Consider a term in the query that is rare in the collection (e.g., *arachnocentric*)
- A document containing this term is very likely to be relevant to the query *arachnocentric*
- → We want a high weight for rare terms like *arachnocentric*.

Collection vs. Document frequency

- Collection frequency of t is the number of occurrences of t in the collection
- Document frequency of t is the number of documents in which t occurs

- Example:

Word	Collection frequency	Document frequency
<i>insurance</i>	10440	3997
<i>try</i>	10422	8760

- Which word is for better search (gets higher weight)

idf weight

- df_t is the document frequency of t : the number of documents that contain t
 - df_t is an inverse measure of the informativeness of t
 - $df_t \leq N$
- We define the idf (inverse document frequency) of t by
$$idf_t = \log_{10} (N/df_t)$$
 - We use $\log (N/df_t)$ instead of N/df_t to “dampen” the effect of idf.

idf example, suppose $N = 1$ million

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1,000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

$$idf_t = \log_{10} (N/df_t)$$

There is one idf value for each term t in a collection.

Effect of idf on ranking

- Does idf have an effect on ranking for one-term queries, like
 - iPhone
- idf has no effect on ranking one term queries
 - idf affects the ranking of documents for queries with at least two terms
- For the query capricious person, idf weighting makes occurrences of capricious count for much more in the final document ranking than occurrences of person.

tf-idf weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight.

$$W_{t,d} = 1 + \log(\text{tf}_{t,d}) \times \log_{10}(N / \text{df}_t)$$

- Best known weighting scheme in information retrieval
 - Note: the “-” in tf-idf is a hyphen, not a minus sign!
 - Alternative names: tf.idf, tf x idf
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Score for a document given a query

$$Score(q, d) = \sum_{t \in q \cap d} tf \cdot idf_{t,d}$$

- There are many variants
 - How “tf” is computed (with/without logs)
 - Whether the terms in the query are also weighted
 - ...

Binary \rightarrow count \rightarrow weight matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$ $= 157 \times 10^9$