Université Jean Monnet, Saint-Étienne
Faculté des Sciences et Techniques de Saint-Étienne
Laboratoire Hubert Curien, UMR CNRS 5516

Recherche d'Information (RI)

Mathias Géry
Mathias.Gery@univ-st-etienne.fr

Information Retrieval and Web Search: Outline

Chapter 3
Ranked Retrieval

3.1: Scoring as the basis of ranked retrieval
3.2: Term frequency
3.3: Document frequency
3.4: Vector space model
3.5: *tf.idf* variants

Boolean retrieval issues

- 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.
- Dominant language in commercial systems until the Web
- 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 read 1000s of results.
    - This is particularly true of web search.

Boolean retrieval problem: feast or famine?

- Boolean queries often result in either too few (=0) or too many (1000s) results.

- Boolean retrieval (conjunction):
  - Query 1: "*standard user dlink 650*" → 200,000 hits - **feast**
  - Query 2: "*standard user dlink 650 no card found*" → 0 hits - **famine**

- In Boolean retrieval, 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

Another example

- Information need:
  - information about climbing the Rucu Pichincha in april from the Teleferiqo

- Boolean query?
  - climbing AND the AND rucu AND pichincha AND in AND april AND from AND the AND teleferiqo ➔ 2 results!
  - climbing OR the OR rucu OR pichincha OR in OR april OR from OR the OR teleferiqo ➔ 798,739,568 results!
  - climbing OR rucu OR pichincha OR april OR teleferiqo ➔ 11,888,589 results!

➔ Ranking!

Ranked vs Boolean retrieval models

- Queries:
  - Boolean retrieval: a query language of operators and expressions.
  - Ranked retrieval: the user's query is just one or more words in a human language (free text queries).

- Results:
  - Boolean retrieval: a set of documents satisfying a query expression.
  - Ranked retrieval: an ordering over the (top) documents in the collection for a query.

- Theory: two separate choices here.
- Practice: ranked retrieval associated with free text queries and Boolean retrieval with a query language.

Ranked retrieval: feast or famine?

- With ranking, large result sets are not an issue:
  - Just show the top k (≈ 10) results.
  - Doesn't overwhelm the user.
  - Premise: the ranking algorithm works, i.e.: More relevant results are ranked higher than less relevant results.

Scoring as the basis of ranked retrieval

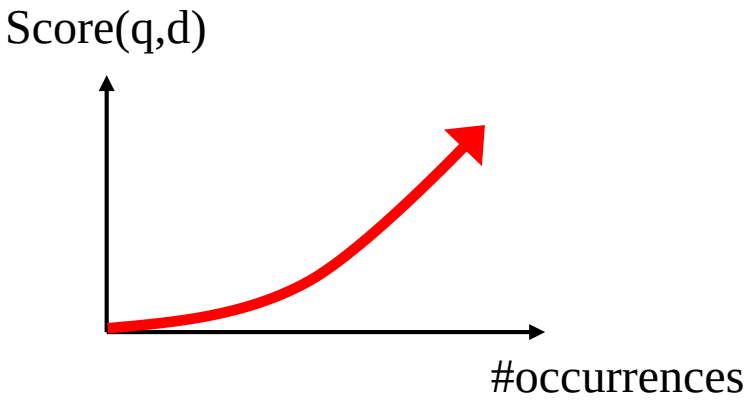
- We wish to rank documents that are more relevant in the collection with respect to a query, higher than documents that are less relevant.

- How?
  - Assign a score to each query-document pair, say in [0, 1].
  - This score measures how well document and query “match”.

Query-document matching scores

- How do we compute the score of 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).



- 2-terms query ?
- n-terms query ?

- We will look at a number of alternatives.

Take 1: Jaccard coefficient

- Jaccard Coefficient: a commonly used measure of overlap of two sets A and B:
  - $Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|}$
  - $Jaccard(A, A) = 1$
  - $Jaccard(A, B) = 0 \; if \; |A \cap B| = 0$
- 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*
  - Document$_2$: *the long march*

- $Jaccard(Query, Document_1) = \frac{1}{6}$
- $Jaccard(Query, Document_2) = \frac{1}{5}$

Issues with Jaccard for scoring?

Query: *when to climb Cotopaxi*

- Term in a document (informativeness):

  Document$_1$: *term$_1$ term$_2$ term$_3$ Cotopaxi term$_5$ term$_6$ term$_7$*
  Document$_2$: *term$_1$ term$_2$ term$_3$ Cotopaxi term$_5$ term$_6$ Cotopaxi*

  – A document with 10 occurrences of a query term is (probably) more relevant than a document with 1 occurrence of the same term.

- Term in the collection (discriminant power):

  Document$_1$: *term$_1$ term$_2$ term$_3$ climb term$_5$ to term$_7$*
  Document$_2$: *term$_1$ term$_2$ term$_3$ Cotopaxi term$_5$ term$_6$ term$_7$*

  – Rare terms in a collection are more informative than frequent terms.

- Length of documents (normalization):

  Document$_1$: *term$_1$ term$_2$ term$_3$ Cotopaxi term$_5$*
  Document$_2$: *term$_1$ term$_2$ term$_3$ Cotopaxi term$_5$ term$_6$ term$_7$ term$_8$ term$_9$ term$_{10}$*

  – A small document containing some relevant information related to the query is more relevant than a huge document containing the same information.

Chapter 3
Ranked Retrieval

3.1: Scoring as the basis of ranked retrieval
3.2: Term frequency
3.3: Document frequency
3.4: Vector space model
3.5: *tf.idf* variants

(Recall) Binary term-document incidence matrix

- Consider if a document contains a term (1) or not (0):

	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:

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	1
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

- Each document is represented by a count vector $\in \mathbb{N}^{|V|}$.

Bag of words model (BOW)

- Vector representation doesn't consider the ordering of words in a document:
  - *John is quicker than Mary*
  - *Mary is quicker than John*
  - ➔ same vectors

- This is called the bag of words model (BOW).

Term frequency $tf_{t,d}$

- The term frequency $tf_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.

NB: in IR, frequency = count

- We want to use $tf_{t,d}$ when computing query-document match scores. But how?

- Raw term frequency?
  - E.g.: Score(q="brutus caesar", d) = $tf_{brutus,d} + tf_{caesar,d}$?
  - A document with $tf_{t,d}$ = 10 occurrences of the term is more relevant than a document with $tf_{t,d}$ = 1 occurrence of the term.
  - But not 10 times more relevant.

Document$_1$: *term$_1$ Brutus term$_3$ Brutus term$_5$ term$_6$ term$_7$*
Document$_2$: *term$_1$ Caesar term$_3$ Brutus term$_5$ term$_6$ term$_7$*

- Relevance does not increase proportionally with term frequency.

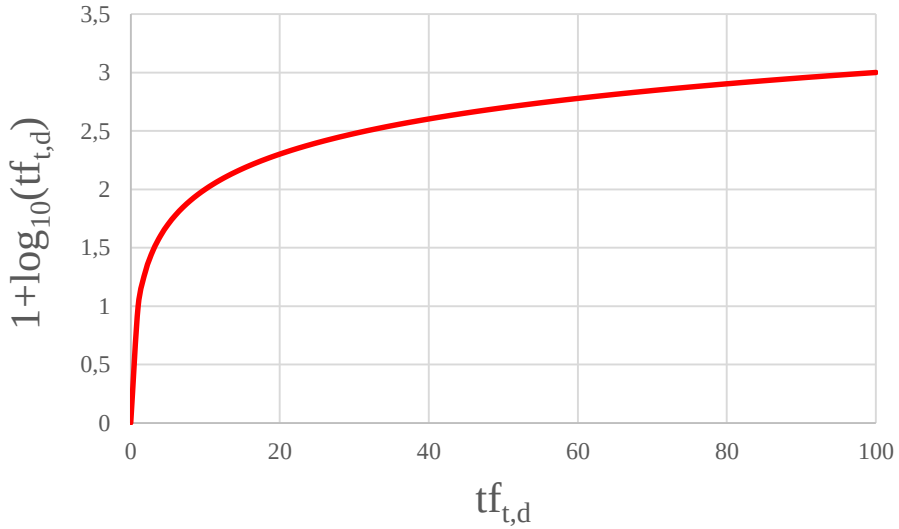
1 + 1 > 2 + 0 !!!

Instead of raw frequency: Log-frequency weighting

- The log frequency weight of term t in d is:

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

$\mathrm{tf}_{t,d}$	$1+\log_{10}(\mathrm{tf}_{t,d})$
0	0
1	1
2	1,30
3	1,48
10	2,00
50	2,70
100	3,00
1000	4,00



Log-frequency weighting

- Consider the Log-frequency weighting of a term in a document:

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

$$tf_{Antony,Antony\ and\ Cleopatra} = 157$$

$$w_{Antony,Antony\ and\ Cleopatra} = 1 + \log_{10} \mathrm{tf}_{t,d} = 3.20$$

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	3.20	2.86	0.00	0.00	0.00	1.00
Brutus	1.60	3.20	0.00	1.00	0.00	0.00
Caesar	3.37	3.36	0.00	1.30	1.00	1.00
Calpurnia	0.00	2.00	0.00	0.00	0.00	0.00
Cleopatra	2.76	0.00	0.00	0.00	0.00	0.00
mercy	1.30	0.00	1.48	1.70	1.70	1.00
worser	1.30	0.00	1.00	1.00	1.00	0.00

Each document is represented by a real valued vector in $\mathbb{R}^{|V|}$.

Scoring with Log-frequency weighting

- Score for a document-query pair:
  - Sum over terms *t* in both *q* and *d*.
  - Scoring function: Relevant Status Value (RSV).

$$\begin{aligned} \text{RSV(d,q)} &= \sum_{t \in q \cap d} w_{t,d} \\ &= \sum_{t \in q} (1 + \log(tf_{t,d})) \end{aligned}$$

  - The score is 0 if none of the query terms is present in the document.

Scoring with Log-frequency weighting

- Consider:
  - the Log-frequency terms weights in the documents ;
  - the binary term weights in the query “Brutus Caesar” :

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	Query
Antony	3.20	2.86	0.00	0.00	0.00	1.00	0.00
Brutus	1.60	3.20	0.00	1.00	0.00	0.00	1.00
Caesar	3.37	3.36	0.00	1.30	1.00	1.00	1.00
Calpurnia	0.00	2.00	0.00	0.00	0.00	0.00	0.00
Cleopatra	2.76	0.00	0.00	0.00	0.00	0.00	0.00
mercy	1.30	0.00	1.48	1.70	1.70	1.00	0.00
worser	1.30	0.00	1.00	1.00	1.00	0.00	0.00
RSV(d,q)	4.97	6.56	0.00	2.30	1.00	1.00	

$$\text{RSV(d,q)} = \sum_{t \in q \cap d} w_{t,d}$$

Issues with Jaccard for scoring?

Query: *when to climb Cotopaxi*

- Term in a document (informativeness):

  Document$_1$: *term$_1$ term$_2$ term$_3$ Cotopaxi term$_5$ term$_6$ term$_7$*
  Document$_2$: *term$_1$ term$_2$ term$_3$ Cotopaxi term$_5$ term$_6$ Cotopaxi*

  – A document with 10 occurrences of a query term is (probably) more relevant than a document with 1 occurrence of the same term.

- Term in the collection (discriminant power):

  Document$_1$: *term$_1$ term$_2$ term$_3$ climb term$_5$ to term$_7$*
  Document$_2$: *term$_1$ term$_2$ term$_3$ Cotopaxi term$_5$ term$_6$ term$_7$*

  – Rare terms in a collection are more informative than frequent terms.

- Length of documents (normalization):

  Document$_1$: *term$_1$ term$_2$ term$_3$ Cotopaxi term$_5$*
  Document$_2$: *term$_1$ term$_2$ term$_3$ Cotopaxi term$_5$ term$_6$ term$_7$ term$_8$ term$_9$ term$_{10}$*

  – A small document containing some relevant information related to the query is more relevant than a huge document containing the same information.

Chapter 3
Ranked Retrieval

3.1: Scoring as the basis of ranked retrieval
3.2: Term frequency
3.3: Document frequency
3.4: Vector space model
3.5: *tf.idf* variants

Document frequency

- In addition to term frequency (the frequency of the term in the document) . . .
- . . . we also want to use the frequency of the term in the whole collection.

- Idea: rare terms in a collection are more informative than frequent terms:
  - Recall stop words…
  - A term in the query that is rare in the collection: e.g., *Cotopaxi*.
  - A document containing this term is very likely to be relevant.

    → We want a high weight for rare terms like *Cotopaxi*.

Query: *when to climb Cotopaxi*

- How to measure? Rare terms / discriminant power.

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 discriminant power of t
  - $df_t \leq N$ (N is the number of documents in the collection).

- We define the idf (inverse document frequency) of t by:

$$idf_t = log_{10}\left(\frac{N}{df_t}\right)$$

  - idf_t is a measure of the discriminant power of the term.
  - Note that we use the log transformation for both term frequency ($tf_{t,d}$) and document frequency (df_t).

idf example, suppose N = 1 million

$$idf_t = log_{10}\left(\frac{N}{df_t}\right)$$

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

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:
  - Query = “Calpurnia”
  - ?

  - idf has no effect on ranking one term queries.
  - idf affects the ranking of documents for queries with at least two terms.
  - Example: Query = “Calpurnia fly”
  - idf weighting increases the relative weight of Calpurnia and decreases the relative weight of fly.

Collection vs. Document frequency

- The collection frequency of *t* is the number of occurrences of *t* in the collection, counting multiple occurrences.
- Example:

Term	Collection frequency	Document frequency
insurance	10440	3997
try	10422	10360

- Which word is a better search term (and should get a higher weight)?

- Rare terms / discriminant power!

tf-idf weighting

SMART ltn :

$$w_{t,d} = TF_{t,d} * IDF_t = (1 + \log(tf_{t,d})) * \log\left(\frac{N}{df_t}\right)$$

- The tf-idf weight . . .
  - . . . increases with the number of occurrences within a document (tf).
  - . . . increases with the rarity of the term in the collection (idf).
- There are many variants
  - How "tf" is computed (with/without logs).
  - Whether the terms in the query are also weighted.
  - …

tf-idf weighting

- Consider the tf.idf terms weights (e.g. ltn) in the documents:

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0.00	0.00	0.00	0.35
Brutus	1.21	6.10	0.00	1.00	0.00	0.00
Caesar	8.59	2.54	0.00	1.51	0.25	0.13
Calpurnia	0.00	1.54	0.00	0.00	0.00	0.00
Cleopatra	2.85	0.00	0.00	0.00	0.00	0.00
mercy	1.51	0.00	1.90	0.12	5.25	0.88
worser	1.37	0.00	0.11	4.15	0.25	0.00

$$w_{t,d} = TF_{t,d} * IDF_t = (1+\log(tf_{t,d})) * \log\left(\frac{N}{df_t}\right)$$

Scoring with tf.idf *ltn*

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

SMART ltn :

$$w_{t,d} = TF_{t,d} * IDF_t = (1+\log(tf_{t,d})) * \log\left(\frac{N}{df_t}\right)$$

$$\text{RSV}_{\text{ltn}}(d,q) = \sum_{t \in q \cap d} w_{t,d}$$

$$= \sum_{t \in q} \left(\left(1 + \log(tf_{t,d})\right) * \log\left(\frac{N}{df_t}\right) \right)$$

Scoring with tf.idf *ltn*

- Consider:
  - the tf.idf terms weights (e.g. ltn) in the documents ;
  - the binary term weights in the query "Brutus Caesar" :

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	Query
Antony	5.25	3.18	0.00	0.00	0.00	0.35	0.00
Brutus	1.21	6.10	0.00	1.00	0.00	0.00	1.00
Caesar	8.59	2.54	0.00	1.51	0.25	0.13	1.00
Calpurnia	0.00	1.54	0.00	0.00	0.00	0.00	0.00
Cleopatra	2.85	0.00	0.00	0.00	0.00	0.00	0.00
mercy	1.51	0.00	1.90	0.12	5.25	0.88	0.00
worser	1.37	0.00	0.11	4.15	0.25	0.00	0.00
$RSV_{ltn}(d,q)$	9.80	8.64	0.00	2.51	0.25	0.13	

$$RSV_{ltn}(d,q) = \sum_{t \in q \cap d} w_{t,d}$$