1985
جامعة محمد بوضياف - المسيلة
Université Mohamed Boudiaf - M'sila

Information retrieval (IR)

By: **Dr. LOUNNAS Bilal**

Ranked retrieval

Ranked retrieval

- Thus far, our queries have all been Boolean.
  - For that document either match or don't.
- Good for expert users with precise understanding of thier needs from the collection.
  - Also good for applications that 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 works)
  - Most users don't want to wade through 1000s of results.
    - This is particularly true of web search

Ranked retrieval

The commun 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 take 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

The solution is to developed ranked retrieval system

- Rather than a set of documents satisfying a query expression, in Ranked retrieval models, the system returns an ordering over the (top) documents in the collection with respect to 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.

Ranked retrieval

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 (may be 10) results.
  - We don't overwhelm the user

  - Premise: Results that are more relevant are ranked higher than results that are less relevant - the ranking algorithm works.

Ranked retrieval

Scoring as the basis of ranked retrieval

- Assign a score to each query-document pair, say in [0, 1].
- This score measures how well document and query "match".
- If the query consists of just one term:
  - For example: Dlink
  - Score should be 0 if the query term does not occur in the document.
  - The more frequent the query term in the document, the higher the score

Scoring with Jaccard Coefficient

Jaccard Coefficient

- A commonly used measure of overlap of two sets
- Let A and B be two sets
  Jaccard coefficient:
  $jaccard(A, B) = (|A \cap B|) \div (|A \cup B|)$
  $(A! = \emptyset or B! = \emptyset)$
- $jaccard(A, A) = 1$
- $jaccard(A, B) = 0 if A \cap B = 0$
- A and B dont have to be the same size.
- Always assigns a number between 0 and 1.

Scoring with 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

- Results:

jaccard(q, d1) = 1/6 - jaccard(q, d2) = 1/5

Scoring with 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

- Results:

jaccard(q, d1) = 1/6 - jaccard(q, d2) = 1/5

Scoring with Jaccard Coefficient

Issues with Jaccard for scoring

- It doesn't consider term frequency (how many occurrences a term has).
- It also does not consider the fact that Rare terms in a collection are more informative than frequent terms.
- We need a more sophisticated way of normalizing for length.
  - divide by the union isn't quit right.
  - later we will see other normalisation by using cosin similarity.

Recall: Binary term-document incidence matrix

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

Term document

- Consider the number of occurrences of a term in a document:
  - Each document is a count vector in $N^{|v|}$: a column below

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Anthony	157	73	0	0	0	1
Brutus	4	157	0	2	0	0
Caesar	232	227	0	2	1	0
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	8	5	8
worser	2	0	1	1	1	5

- each document is represented by a count vector $\in N^{|v|}$

Term frequency tf (raw frequency)

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.
- We want to use tf when computing query-document match scores. But how?
- We could not just use *tf* as is ("raw term frequency").
  - A document with $tf = 10$ occurrences of the term is more relevant than a document with $tf = 1$ occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

NB: frequency = count in IR not the ration

Instead of raw frequency: Log frequency weighting

logarithmically scaled frequency

- The log frequency weight of term t in d is defined as follows.

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

$\mathrm{tf}_{t,d}$	0	1	2	10	1000
$w_{t,d}$	0	1	1.3	2	4

- Score for a document-query pair: sum over terms t in both q and d:

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

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

Instead of raw frequency: Log frequency weighting

Differents kind of normalisation

Variants of term frequency (TF) weight

weighting scheme	TF weight
binary	$0, 1$
raw count	$f_{t,d}$
term frequency	$f_{t,d} \Big/ \sum_{t' \in d} f_{t',d}$
log normalization	$1 + \log(f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$
double normalization K	$K + (1 - K) \frac{f_{t,d}}{\max_{\{t' \in d\}} f_{t',d}}$

Document Frequency

Document Frequency

- Rare terms are more informative than frequent terms
  - Recall stop words. (has the effect of effecting the scoring, while the rare terms are very important).
- 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.

Document Frequency

Document Frequency - continued

- Frequent terms are less informative than rare terms.
- Consider a query term that is frequent in the collection (e.g., high, increase, line)
- A document containing such a term is more likely to be relevant than a document that doesn't
- But it's not a sure indicator of relevance.

1. For frequent terms, we want high positive weights for words like high, increase, and line.
2. And lower weights than for rare terms.

We will use document frequency (df) to capture this.

idf weight

Inverse Document Frequency (idf)

- df_t (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* weight of term t as follows:

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

  - idf_t is a **measure** of the informativeness of the term.
  - $\log \dfrac{N}{df_t}$ instead of $\dfrac{N}{df_t}$ to "dampen" the effect of *idf*
- Note that we use the log transformation for both term frequency and document frequency.

idf weight

Inverse Document Frequency (idf) Example

Compute idf_t using the formula: $idf_t = \log_{10}(\frac{1000000}{df_t})$

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

Effect of idf on ranking

Effect of idf on ranking

- *idf* affects the ranking of documents for queries with at least two terms.
- For example, in the query "arachnocentric line", idf weighting increases the relative weight of **arachnocentric** and decreases the relative weight of **line**.
- *idf* has little effect on ranking for one-term queries.

tf-idf

tf-idf weighting

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

tf-idf weight

$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$$

- 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

Final ranking of documents for a query

Score of documents for a query

$$\text{Score}(q,d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$$

What we achived until now

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 $\in \{0, 1\}^{|V|}$

What we achived until now

Term-document count matrices

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Anthony	157	73	0	0	0	1
Brutus	4	157	0	2	0	0
Caesar	232	227	0	2	1	0
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	8	5	8
worser	2	0	1	1	1	5

- each document is represented by a count vector $\in N^{|v|}$

What we achived until now

Weight matrix (tf-idf)

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Anthony	5.25	3.18	0.0	0.0	0.0	0.35
Brutus	1.21	6.10	0.0	1.0	0.0	0.0
Caesar	8.59	2.54	0.0	1.51	0.25	0.0
Calpurnia	0.0	1.54	0.0	0.0	0.0	0.0
Cleopatra	2.85	0.0	0.0	0.0	0.0	0.0
mercy	1.51	0.0	1.90	0.12	5.25	0.88
worser	1.37	0.0	0.11	4.15	0.25	1.95

- each document is represented by a real-valued vector of if-idf weights $\in R^{|v|}$

Vector Space Model

Vector Space Model

- Each document is represented by a real-valued vector of if-idf weights $\in R^{|v|}$
- So we have a |V| - dimensional real-valued vector space.
- Terms are axes of the space.
- Documents are points or vectors in this space.
- Very high-dimensional: tens of millions of dimensions when you apply this to web search engines.

Queries as vectors

So if we have that vectors space of documents

how we can handles Queries?

- Key idea 01: do the same for queries: represent them as vectors in the high-dimensional space.
- Key idea 02: Rank documents according to their proximity to the query.

Proximity = similarity of vectors

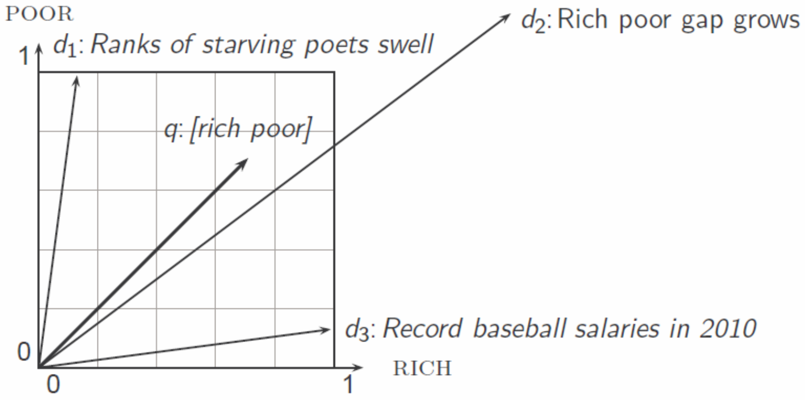
This allows us to rank relevant documents higher than nonrelevant documents

Formalize vector space

How do we formalize vector space similarity?

- First cut: distance between two points
  - (= distance between the end points of the two vectors)
  - Euclidean distance?
- Euclidean distance is a bad idea ... Why?
- ... because Euclidean distance is large for vectors of different lengths.

Why distance is a bad idea



The Euclidean distance of q and d2 is large although the distribution of terms in the query q and the distribution of terms in the document d2 are very similar.

Use angle instead of distance

Rank documents according to angle with query

- Thought experiment: take a document d and append it to itself. Call this document d'. d' is twice as long as d.
  - "Semantically" d and d' have the same content.
- The Euclidean distance between the two documents can be quite large.
- The angle between the two documents is 0, corresponding to maximal similarity . . .

From angles to cosines

- The following two notions are equivalent:
  - Rank documents according to the angle between query and document in decreasing order.
  - Rank documents according to cosine(query,document) in increasing order.
- Why Cosine?:
  - Because it's very efficient standard tool for measuring the similarity between two vectors using their angle.
  - Simple to implement and use.

Length normalization

- How do we compute the cosine?
- A vector can be (length-) normalized by dividing each of its components by its length here we use the L2 norm: $||x||_2 = \sqrt{\sum_i x_i^2}$
- As a result, longer documents and shorter documents have weights of the same order of magnitude.
- Effect on the two documents d and d' (d appended to itself) from earlier slide: they have identical vectors after length-normalization.

Cosine similarity between query and document

$$\cos(\vec{q},\vec{d}) = \mathrm{SIM}(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

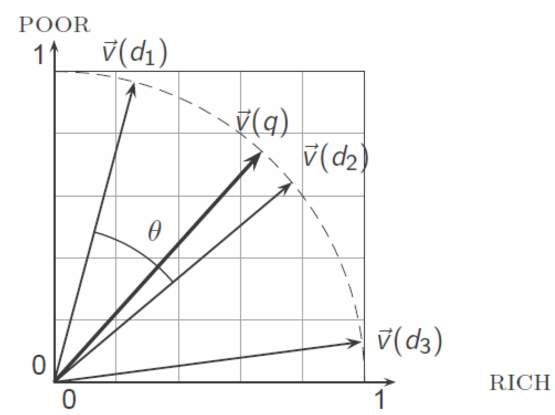
- q_i is the tf-idf weight of term i in the query.
- d_i is the tf-idf weight of term i in the document.
- $|\vec{q}|$ and $|\vec{d}|$ are the lengths of $\vec{q}$ and $\vec{d}$.
- This is the cosine similarity of $\vec{q}$ and $\vec{d}$ . . . . . . or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$

Cosine for normalized vectors

$$\cos(\vec{q},\vec{d}) = \vec{q} \bullet \vec{d} = \sum_{i=1}^{|V|} q_i d_i$$

- For normalized vectors, the cosine is equivalent to the dot product or scalar product.

Cosine similarity illustrated



Cosine Example

How similar are the following novels?

SaS: Sense and Sensibility

PaP: Pride and Prejudice

WH: Wuthering Heights

Term frequencies (raw counts)

term	SaS	PaP	WH
AFFECTION	115	58	20
JEALOUS	10	7	11
GOSSIP	2	0	6
WUTHERING	0	0	38

Cosine Example

	Term frequencies (raw counts)			Log frequency weighting			Log frequency weighting and cosine normalisation		
term	SaS	PaP	WH	SaS	PaP	WH	SaS	PaP	WH
AFFECTION	115	58	20	3.06	2.76	2.30	0.789	0.832	0.524
JEALOUS	10	7	11	2.0	1.85	2.04	0.515	0.555	0.465
GOSSIP	2	0	6	1.30	0.00	1.78	0.335	0.000	0.405
WUTHERING	0	0	38	0.00	0.00	2.58	0.000	0.000	0.588

- (To simplify this example, we don't do idf weighting.)
- $\cos(\text{SaS},\text{PaP}) \approx$
  $0.789 * 0.832 + 0.515 * 0.555 + 0.335 * 0.0 + 0.0 * 0.0 \approx 0.94.$
- $\cos(\text{SaS},\text{WH}) \approx 0.79$
- $\cos(\text{PaP},\text{WH}) \approx 0.69$

SMART Notation

Term frequency		Document frequency		Normalization	
n (natural)	$\text{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1+\log(\text{tf}_{t,d})$	t (idf)	$\log\frac{N}{\text{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5+\frac{0.5\times\text{tf}_{t,d}}{\max_t(\text{tf}_{t,d})}$	p (prob idf)	$\max\{0,\log\frac{N-df_t}{df_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases}1 & \text{if } tf_{t,d}>0\\ 0 & \text{otherwise}\end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha<1$
L (log ave)	$\frac{1+\log(tf_{t,d})}{1+\log(ave_{t\in d}(tf_{t,d}))}$				

Best known combination of weighting options

- SMART information retrieval system
  - can used to document and query differently. (Next Slide)
  - Many notation such as LTC

Weighting may differ in queries vs documents

- Many search engines allow for different weightings for queries vs. documents.
- SMART Notation: denotes the combination in use in an engine, with the notation ddd.qqq, using the acronyms from the previous table.

Example: lnc.ltn

Document:

- l ogarithmic tf
- n o df weighting
- c osine normalization

Query:

- l ogarithmic tf
- t – means idf
- n o normalization

SMART Notation example: INC.ITN

Query: "best car insurance". Document: "car insurance auto insurance".

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0	0	5000	2.3	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0	0	0	0	0
car	1	1	10000	2.0	2.0	1	1	1	0.52	1.04
insurance	1	1	1000	3.0	3.0	2	1.3	1.3	0.68	2.04

Key to columns: tf-raw: raw (unweighted) term frequency, tf-wght: logarithmically weighted term frequency, df: document frequency, idf: inverse document frequency, weight: the final weight of the term in the query or document, n'lized: document weights after cosine normalization, product: the product of final query weight and final document weight

$\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92$
$1/1.92 \approx 0.52$
$1.3/1.92 \approx 0.68$

Final similarity score between query and document: $\sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08$

Summary: Ranked retrieval in the vector space model

- Represent the query as a weighted tf-idf vector.
- Represent each document as a weighted tf-idf vector.
- Compute the cosine similarity between the query vector and each document vector.
- Rank documents with respect to the query.
- Return the top K (e.g., K = 10) to the user.