

Introduction to Information Retrieval

<http://informationretrieval.org>

IIR 1: Boolean Retrieval

Hinrich Schütze

Center for Information and Language Processing, University of Munich

2014-04-09

Boolean retrieval

- The Boolean model is arguably the simplest model to base an information retrieval system on.
- Queries are Boolean expressions, e.g., CAESAR AND BRUTUS
- The search engine returns all documents that satisfy the Boolean expression.

Does Google use the Boolean model?

Outline

- 1 Introduction
- 2 Inverted index
- 3 Processing Boolean queries
- 4 Query optimization
- 5 Course overview

Unstructured data in 1650: Shakespeare



Unstructured data in 1650

- Which plays of Shakespeare contain the words BRUTUS AND CAESAR, but NOT CALPURNIA?
- One could grep all of Shakespeare's plays for BRUTUS and CAESAR, then strip out lines containing CALPURNIA.
- Why is grep not the solution?
 - Slow (for large collections)
 - grep is line-oriented, IR is document-oriented
 - "NOT CALPURNIA" is non-trivial
 - Other operations (e.g., find the word ROMANS near COUNTRYMAN) not feasible

Term-document incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	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	

...

Entry is 1 if term occurs. Example: CALPURNIA occurs in *Julius Caesar*. Entry is 0 if term doesn't occur. Example: CALPURNIA doesn't occur in *The tempest*.

Incidence vectors

- So we have a 0/1 vector for each term.
- To answer the query BRUTUS AND CAESAR AND NOT CALPURNIA:
 - Take the vectors for BRUTUS, CAESAR, and CALPURNIA
 - Complement the vector of CALPURNIA
 - Do a (bitwise) AND on the three vectors
 - $110100 \text{ AND } 110111 \text{ AND } 101111 = 100100$

0/1 vectors and result of bitwise operations

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	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	
...							
result:	1	0	0	1	0	0	

Answers to query

Anthony and Cleopatra, Act III, Scene ii

Agrippa [Aside to Domitius Enobarbus]: Why, Enobarbus,
When Antony found Julius Caesar dead,
He cried almost to roaring; and he wept
When at Philippi he found Brutus slain.

Hamlet, Act III, Scene ii

Lord Polonius: I did enact Julius Caesar: I was killed i' the
Capitol; Brutus killed me.

Bigger collections

- Consider $N = 10^6$ documents, each with about 1000 tokens
- \Rightarrow total of 10^9 tokens
- On average 6 bytes per token, including spaces and punctuation \Rightarrow size of document collection is about $6 \cdot 10^9 = 6$ GB
- Assume there are $M = 500,000$ distinct terms in the collection
- (Notice that we are making a term/token distinction.)

Can't build the incidence matrix

- $M = 500,000 \times 10^6 =$ half a trillion 0s and 1s.
- But the matrix has no more than one billion 1s.
 - Matrix is extremely sparse.
- What is a better representations?
 - We only record the 1s.

Inverted Index

For each term t , we store a list of all documents that contain t .

BRUTUS →

1	2	4	11	31	45	173	174
---	---	---	----	----	----	-----	-----

CAESAR →

1	2	4	5	6	16	57	132	...
---	---	---	---	---	----	----	-----	-----

CALPURNIA →

2	31	54	101
---	----	----	-----

⋮


dictionary


postings

Tokenization and preprocessing

Doc 1. I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me.

Doc 2. So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious:



Doc 1. i did enact julius caesar i was killed i' the capitol brutus killed me

Doc 2. so let it be with caesar the noble brutus hath told you caesar was ambitious

Generate postings

Doc 1. i did enact julius caesar i was
killed i' the capitol brutus killed me

Doc 2. so let it be with caesar the
noble brutus hath told you caesar was
ambitious



term	docID
i	1
did	1
enact	1
julius	1
caesar	1
i	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Sort postings

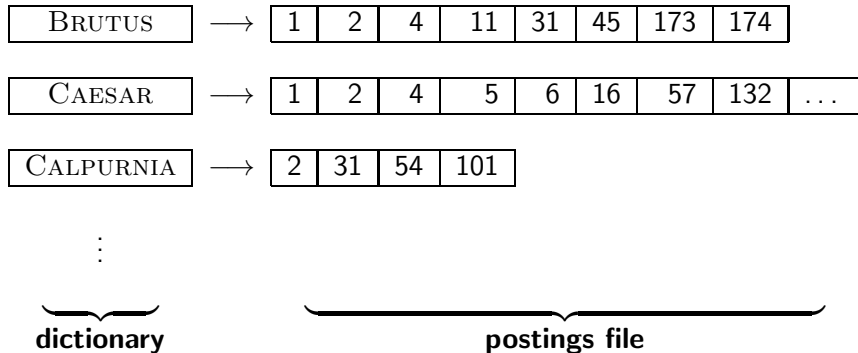
term	docID		term	docID
i	1		ambitious	2
did	1		be	2
enact	1		brutus	1
julius	1		brutus	2
caesar	1		capitol	1
i	1		caesar	1
was	1		caesar	2
killed	1		caesar	2
i'	1		did	1
the	1		enact	1
capitol	1		hath	1
brutus	1		i	1
killed	1		i	1
me	1	⇒	i'	1
so	2		it	2
let	2		julius	1
it	2		killed	1
be	2		killed	1
with	2		let	2
caesar	2		me	1
the	2		noble	2
noble	2		so	2
brutus	2		the	1
hath	2		the	2
told	2		told	2
you	2		you	2
caesar	2		was	1
was	2		was	2
ambitious	2		with	2

Create postings lists, determine document frequency

term	docID			
ambitious	2			
be	2			
brutus	1			
brutus	2			
capitol	1			
caesar	1			
caesar	2			
caesar	2			
did	1			
enact	1			
hath	1			
i	1			
i	1			
i'	1			
it	2			
julius	1			
killed	1			
killed	1			
let	2			
me	1			
me	1			
noble	2			
so	2			
so	2			
the	1			
the	2			
told	2			
you	2			
you	2			
was	1			
was	2			
with	2			

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
killed	1	→	2
let	1	→	2
me	1	→	1
me	1	→	2
noble	1	→	2
so	1	→	2
so	2	→	2
the	2	→	1 → 2
the	1	→	2
the	2	→	2
told	1	→	2
you	1	→	2
you	2	→	2
was	2	→	1 → 2
was	1	→	2
with	1	→	2

Split the result into dictionary and postings file



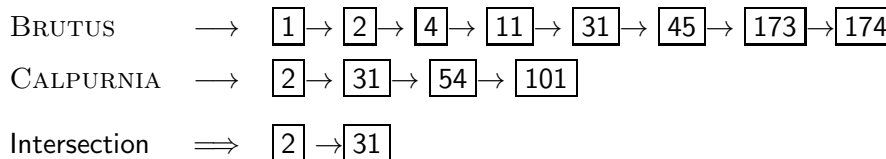
Outline

- 1 Introduction
- 2 Inverted index
- 3 Processing Boolean queries**
- 4 Query optimization
- 5 Course overview

Simple conjunctive query (two terms)

- Consider the query: BRUTUS AND CALPURNIA
- To find all matching documents using inverted index:
 - 1 Locate BRUTUS in the dictionary
 - 2 Retrieve its postings list from the postings file
 - 3 Locate CALPURNIA in the dictionary
 - 4 Retrieve its postings list from the postings file
 - 5 Intersect the two postings lists
 - 6 Return intersection to user

Intersecting two postings lists

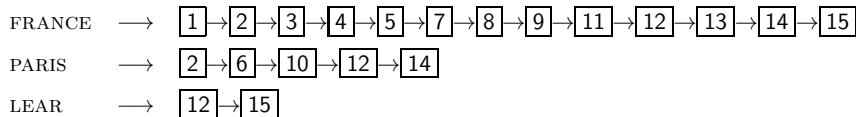


- This is linear in the length of the postings lists.
- Note: This only works if postings lists are sorted.

Intersecting two postings lists

```
INTERSECT( $p_1, p_2$ )
1   $answer \leftarrow \langle \rangle$ 
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $docID(p_1) = docID(p_2)$ 
4      then  $\text{ADD}(answer, docID(p_1))$ 
5           $p_1 \leftarrow next(p_1)$ 
6           $p_2 \leftarrow next(p_2)$ 
7      else if  $docID(p_1) < docID(p_2)$ 
8          then  $p_1 \leftarrow next(p_1)$ 
9          else  $p_2 \leftarrow next(p_2)$ 
10 return  $answer$ 
```

Query processing: Exercise



Compute hit list for ((paris AND NOT france) OR lear)

Boolean retrieval model: Assessment

- The Boolean retrieval model can answer any query that is a Boolean expression.
 - Boolean queries are queries that use AND, OR and NOT to join query terms.
 - Views each document as a **set** of terms.
 - Is precise: Document matches condition or not.
- Primary commercial retrieval tool for 3 decades
- Many professional searchers (e.g., lawyers) still like Boolean queries.
 - You know exactly what you are getting.
- Many search systems you use are also Boolean: spotlight, email, intranet etc.

Commercially successful Boolean retrieval: Westlaw

- Largest commercial legal search service in terms of the number of paying subscribers
- Over half a million subscribers performing millions of searches a day over tens of terabytes of text data
- The service was started in 1975.
- In 2005, Boolean search (called “Terms and Connectors” by Westlaw) was still the default, and used by a large percentage of users . . .
- . . . although ranked retrieval has been available since 1992.

Introduction to Information Retrieval

<http://informationretrieval.org>

IIR 2: The term vocabulary and postings lists

Hinrich Schütze

Center for Information and Language Processing, University of Munich

2014-04-09

Outline

- 1 Recap
- 2 Documents
- 3 **Terms**
 - General + Non-English
 - English
- 4 Skip pointers
- 5 Phrase queries

Definitions

- **Word** – A delimited string of characters as it appears in the text.
- **Term** – A “normalized” word (case, morphology, spelling etc); an equivalence class of words.
- **Token** – An instance of a word or term occurring in a document.
- **Type** – The same as a term in most cases: an equivalence class of tokens.

Normalization

- Need to “normalize” words in indexed text as well as query terms into the same form.
- Example: We want to match *U.S.A.* and *USA*
- We most commonly implicitly define **equivalence classes** of terms.
- Alternatively: do asymmetric expansion
 - window → window, windows
 - windows → Windows, windows
 - Windows (no expansion)
- More powerful, but less efficient
- *Why don't you want to put *window*, *Window*, *windows*, and *Windows* in the same equivalence class?*

Normalization: Other languages

- Normalization and language detection interact.
- *PETER WILL NICHT MIT.* → MIT = mit
- *He got his PhD from MIT.* → MIT ≠ mit

Tokenization: Recall construction of inverted index

- Input:

Friends, Romans, countrymen. So let it be with Caesar ...

- Output:

friend roman countryman so ...

- Each token is a candidate for a postings entry.
- What are valid tokens to emit?

Exercises

In June, the dog likes to chase the cat in the barn. – How many word tokens? How many word types? Why tokenization is difficult

– even in English. **Tokenize:** *Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.*

Tokenization problems: One word or two? (or several)

- Hewlett-Packard
- State-of-the-art
- co-education
- the hold-him-back-and-drag-him-away maneuver
- data base
- San Francisco
- Los Angeles-based company
- cheap San Francisco-Los Angeles fares
- York University vs. New York University

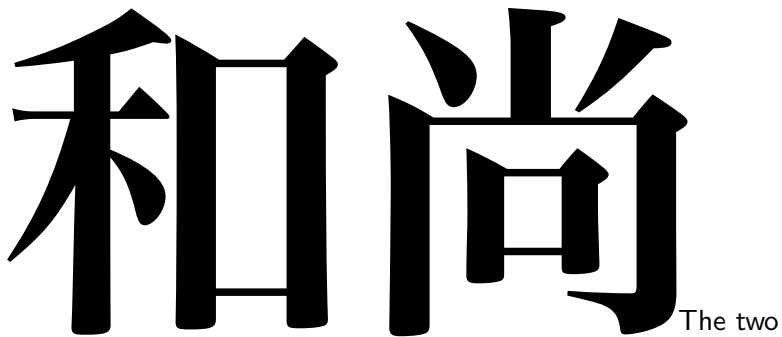
Numbers

- 3/20/91
- 20/3/91
- Mar 20, 1991
- B-52
- 100.2.86.144
- (800) 234-2333
- 800.234.2333
- Older IR systems may not index numbers . . .
- . . . but generally it's a useful feature.
- Google example

Chinese: No whitespace

莎拉波娃现在居住在美国东南部的佛罗里达。今年4月9日，莎拉波娃在美国第一大城市纽约度过了18岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

Ambiguous segmentation in Chinese

The image shows two large, bold Chinese characters, '和' (he) and '尚' (shang), written in a traditional serif font. The character '和' is on the left and '尚' is on the right. They are positioned such that they could be read as a single word '和尚' (monk) or as two separate words '和' (and) and '尚' (still).

The two characters can be treated as one word meaning 'monk' or as a sequence of two words meaning 'and' and 'still'.

Outline

- 1 Recap
- 2 Documents
- 3 **Terms**
 - General + Non-English
 - **English**
- 4 Skip pointers
- 5 Phrase queries

Case folding

- Reduce all letters to lower case
- Even though case can be semantically meaningful
 - capitalized words in mid-sentence
 - MIT vs. mit
 - Fed vs. fed
 - ...
- It's often best to lowercase everything since users will use lowercase regardless of correct capitalization.

Stop words

- stop words = extremely common words which would appear to be of little value in helping select documents matching a user need
- Examples: *a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with*
- Stop word elimination used to be standard in older IR systems.
- But you need stop words for phrase queries, e.g. “King of Denmark”
- Most web search engines index stop words.

More equivalence classing

- Soundex: IIR 3 (phonetic equivalence, Muller = Mueller)
- Thesauri: IIR 9 (semantic equivalence, car = automobile)

Lemmatization

- Reduce inflectional/variant forms to base form
- Example: *am, are, is* → *be*
- Example: *car, cars, car's, cars'* → *car*
- Example: *the boy's cars are different colors* → *the boy car be different color*
- Lemmatization implies doing “proper” reduction to dictionary headword form (the **lemma**).
- Inflectional morphology (*cutting* → *cut*) vs. derivational morphology (*destruction* → *destroy*)

Stemming

- Definition of stemming: Crude heuristic process that **chops off the ends of words** in the hope of achieving what “principled” lemmatization attempts to do with a lot of linguistic knowledge.
- Language dependent
- Often inflectional **and** derivational
- Example for derivational: *automate*, *automatic*, *automation* all reduce to *automat*

Porter algorithm

- Most common algorithm for stemming English
- Results suggest that it is at least as good as other stemming options
- Conventions + 5 phases of reductions
- Phases are applied sequentially
- Each phase consists of a set of commands.
 - Sample command: Delete final *ement* if what remains is longer than 1 character
 - replacement → replac
 - cement → cement
- Sample convention: Of the rules in a compound command, select the one that applies to the longest suffix.

Porter stemmer: A few rules

Rule

SSES → SS

IES → I

SS → SS

S →

Example

caresses → caress

ponies → poni

caress → caress

cats → cat

Three stemmers: A comparison

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Porter stemmer: such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Lovins stemmer: such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Paice stemmer: such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Does stemming improve effectiveness?

- In general, stemming increases effectiveness for some queries, and decreases effectiveness for others.
- Queries where stemming is likely to help: [tartan sweaters], [sightseeing tour san francisco]
- (equivalence classes: {sweater,sweaters}, {tour,tours})
- Porter Stemmer equivalence class *oper* contains all of *operate operating operates operation operative operatives operational*.
- Queries where stemming hurts: [operational AND research], [operating AND system], [operative AND dentistry]

Exercise: What does Google do?

- Stop words
- Normalization
- Tokenization
- Lowercasing
- Stemming
- Non-latin alphabets
- Umlauts
- Compounds
- Numbers

Introduction to Information Retrieval

<http://informationretrieval.org>

IIR 6: Scoring, Term Weighting, The Vector Space Model

Hinrich Schütze

Center for Information and Language Processing, University of Munich

2014-04-30

Outline

- 1 Recap
- 2 Why ranked retrieval?
- 3 Term frequency
- 4 tf-idf weighting
- 5 The vector space model

Ranked retrieval

- Thus far, our queries have been **Boolean**.
 - Documents either match or don't.
- **Good for expert users** with precise understanding of their needs and of the collection.
- Also **good for applications**: Applications can easily consume 1000s of results.
- **Not good for the majority of users**
- Most users are not capable 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 (boolean conjunction): [standard user dlink 650]
 - → 200,000 hits – **feast**
- Query 2 (boolean conjunction): [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.

Feast or famine: No problem in ranked retrieval

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

Scoring as the basis of ranked retrieval

- How can we accomplish a **relevance ranking** of the documents with respect to a query?
- Assign a score to each query-document pair, say in $[0, 1]$.
- This score measures how well document and query “match”.
- Sort documents according to scores

Query-document matching scores

- How do we compute the score of a query-document pair?
- If no query term occurs in the document: score should be 0.
- The more frequent a query term in the document, the higher the score
- The more query terms occur in the document, the higher the score
- We will look at a number of alternatives for doing this.

Take 1: Jaccard coefficient

- A commonly used measure of overlap of two sets
- Let A and B be two sets
- Jaccard coefficient:

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

($A \neq \emptyset$ or $B \neq \emptyset$)

- $\text{JACCARD}(A, A) = 1$
- $\text{JACCARD}(A, B) = 0$ if $A \cap B = \emptyset$
- A and B don't have to be the same size.
- Always assigns a number between 0 and 1.

Jaccard coefficient: Example

- What is the query-document match score that the Jaccard coefficient computes for:
 - Query: “ides of March”
 - Document “Caesar died in March”
 - $\text{JACCARD}(q, d) = 1/6$

What's wrong with Jaccard?

- It doesn't consider term frequency (how many occurrences a term has).
- Rare terms are more informative than frequent terms. Jaccard does not consider this information.
- We need a more sophisticated way of normalizing for the length of a document.
- Later in this lecture, we'll use $|A \cap B| / \sqrt{|A \cup B|}$ (cosine) ...
- ... instead of $|A \cap B| / |A \cup B|$ (Jaccard) for length normalization.

Outline

- 1 Recap
- 2 Why ranked retrieval?
- 3 Term frequency
- 4 tf-idf weighting
- 5 The vector space model

Binary incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	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 as a binary vector $\in \{0, 1\}^{|V|}$.

Count matrix

	Anthony 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 now represented as a count vector $\in \mathbb{N}^{|V|}$.

Bag of words model

- We do not consider the **order** of words in a document.
- *John is quicker than Mary* and *Mary is quicker than John* are represented the same way.
- This is called a **bag of words model**.
- In a sense, this is a step back: The positional index was able to distinguish these two documents.
- We will look at “recovering” positional information later in this course.
- For now: bag of words model

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?
- Raw term frequency is not what we want because:
- 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.

Instead of raw frequency: Log frequency weighting

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

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

- $\text{tf}_{t,d} \rightarrow w_{t,d}$:
 $0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 1.3, 10 \rightarrow 2, 1000 \rightarrow 4$, etc.
- Score for a document-query pair: sum over terms t in both q and d :
 $\text{tf-matching-score}(q, d) = \sum_{t \in q \cap d} (1 + \log \text{tf}_{t,d})$
- The score is 0 if none of the query terms is present in the document.

Exercise

- Compute the Jaccard matching score and the tf matching score for the following query-document pairs.
- q: [information on cars] d: “all you’ve ever wanted to know about cars”
- q: [information on cars] d: “information on trucks, information on planes, information on trains”
- q: [red cars and red trucks] d: “cops stop red cars more often”

Outline

- 1 Recap
- 2 Why ranked retrieval?
- 3 Term frequency
- 4 tf-idf weighting**
- 5 The vector space model

Frequency in document vs. frequency in collection

- 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 collection** for weighting and ranking.

Desired weight for rare terms

- Rare terms are more informative than frequent terms.
- 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.
- → We want **high weights for rare terms** like ARACHNOCENTRIC.

Desired weight for frequent terms

- Frequent terms are less informative than rare terms.
- Consider a term in the query that is **frequent** in the collection (e.g., GOOD, INCREASE, LINE).
- A document containing this term is more likely to be relevant than a document that doesn't ...
- ...but words like GOOD, INCREASE and LINE are not sure indicators of relevance.
- → **For frequent terms** like GOOD, INCREASE, and LINE, we want positive weights ...
- ...but **lower weights** than for rare terms.

Document frequency

- We want **high weights for rare terms** like ARACHNOCENTRIC.
- We want **low (positive) weights for frequent words** like GOOD, INCREASE, and LINE.
- We will use **document frequency** to factor this into computing the matching score.
- The document frequency is **the number of documents in the collection that the term occurs in.**

idf weight

- df_t is the document frequency, the number of documents that t occurs in.
- df_t is an inverse measure of the **informativeness** of term t .
- We define the **idf weight** of term t as follows:

$$\text{idf}_t = \log_{10} \frac{N}{df_t}$$

(N is the number of documents in the collection.)

- idf_t is a measure of the **informativeness** of the term.
- $[\log N/df_t]$ instead of $[N/df_t]$ to “dampen” the effect of idf
- Note that we use the log transformation for both term frequency and document frequency.

Examples for idf

Compute idf_t using the formula: $\text{idf}_t = \log_{10} \frac{1,000,000}{\text{df}_t}$

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

- 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**.

Collection frequency vs. Document frequency

word	collection frequency	document frequency
INSURANCE	10440	3997
TRY	10422	8760

- Collection frequency of t : number of tokens of t in the collection
- Document frequency of t : number of documents t occurs in
- Why these numbers?
- Which word is a better search term (and should get a higher weight)?
- This example suggests that df (and idf) is better for weighting than cf (and “icf”).

tf-idf weighting

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



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

- tf-weight
- idf-weight
- Best known weighting scheme in information retrieval
- Alternative names: tf.idf, tf x idf

Summary: tf-idf

- Assign a tf-idf weight for each term t in each document d :
$$w_{t,d} = (1 + \log \text{tf}_{t,d}) \cdot \log \frac{N}{\text{df}_t}$$
- The tf-idf weight ...
 - ... increases with the number of occurrences within a document. (term frequency)
 - ... increases with the rarity of the term in the collection. (inverse document frequency)

Exercise: Term, collection and document frequency

Quantity	Symbol	Definition
term frequency	$tf_{t,d}$	number of occurrences of t in d
document frequency	df_t	number of documents in the collection that t occurs in
collection frequency	cf_t	total number of occurrences of t in the collection

- Relationship between df and cf ?
- Relationship between tf and cf ?
- Relationship between tf and df ?

Outline

- 1 Recap
- 2 Why ranked retrieval?
- 3 Term frequency
- 4 tf-idf weighting
- 5 **The vector space model**

Binary incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	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 as a binary vector $\in \{0, 1\}^{|V|}$.

Count matrix

	Anthony 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 now represented as a count vector $\in \mathbb{N}^{|V|}$.

Binary \rightarrow count \rightarrow weight matrix

	Anthony 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 now represented as a real-valued vector of tf-idf

weights $\in \mathbb{R}^{|V|}$.

Documents as vectors

- Each document is now represented as a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.
- 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
- Each vector is very sparse - most entries are zero.

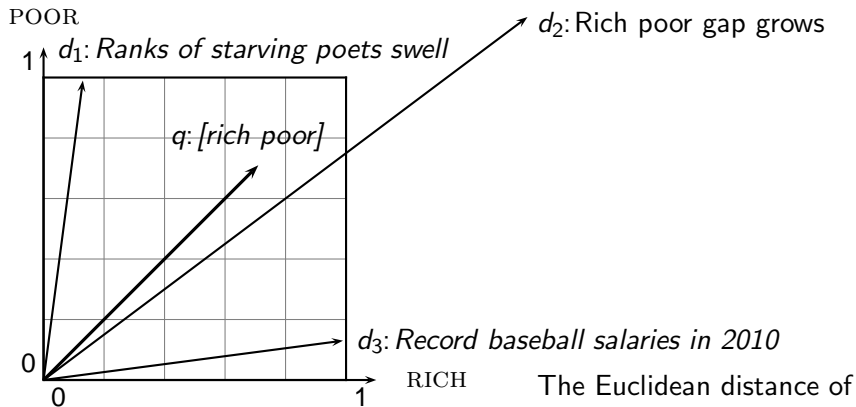
Queries as vectors

- Key idea 1: do the same for queries: represent them as vectors in the high-dimensional space
- Key idea 2: Rank documents according to their proximity to the query
- proximity = similarity
- proximity \approx negative distance
- Recall: We're doing this because we want to get away from the you're-either-in-or-out, feast-or-famine Boolean model.
- Instead: rank relevant documents higher than nonrelevant documents

How do we formalize vector space similarity?

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

Why distance is a bad idea



\vec{q} and \vec{d}_2 is large although the distribution of terms in the query q and the distribution of terms in the document d_2 are very similar.

Questions about basic vector space setup?

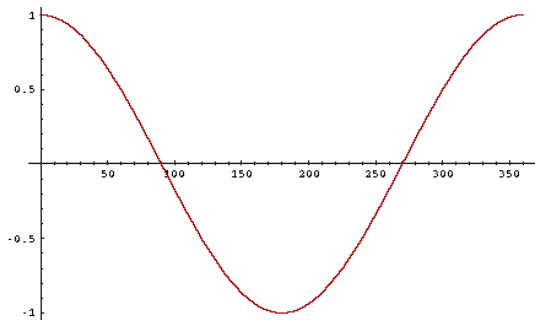
Use angle instead of distance

- 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 angle between the two documents is 0, corresponding to maximal similarity . . .
- . . . even though the Euclidean distance between the two documents can be quite large.

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
- Cosine is a monotonically decreasing function of the angle for the interval $[0^\circ, 180^\circ]$

Cosine



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 L_2 norm:

$$\|x\|_2 = \sqrt{\sum_i x_i^2}$$

- This maps vectors onto the unit sphere ...
- ... since after normalization: $\|x\|_2 = \sqrt{\sum_i x_i^2} = 1.0$
- 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

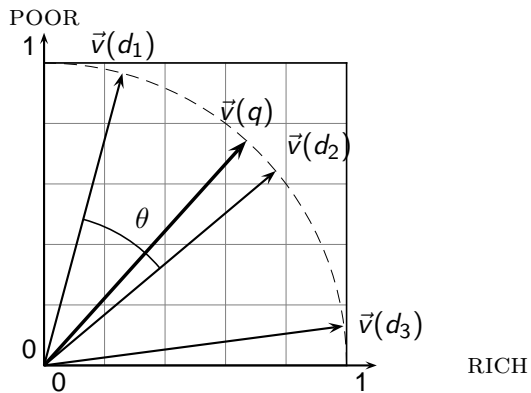
$$\cos(\vec{q}, \vec{d}) = \text{SIM}(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}| |\vec{d}|} = \frac{\sum_{i=1}^{|\mathcal{V}|} q_i d_i}{\sqrt{\sum_{i=1}^{|\mathcal{V}|} q_i^2} \sqrt{\sum_{i=1}^{|\mathcal{V}|} d_i^2}}$$

- 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

- For normalized vectors, the cosine is equivalent to the dot product or scalar product.
- $\cos(\vec{q}, \vec{d}) = \vec{q} \cdot \vec{d} = \sum_i q_i \cdot d_i$
 - (if \vec{q} and \vec{d} are length-normalized).

Cosine similarity illustrated



Cosine: Example

term frequencies (counts)

How similar are
these novels? SaS:
Sense and
Sensibility PaP:
Pride and
Prejudice WH:
Wuthering
Heights

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

Cosine: Example

term frequencies (counts)

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

log frequency weighting

term	SaS	PaP	WH
AFFECTION	3.06	2.76	2.30
JEALOUS	2.0	1.85	2.04
GOSSIP	1.30	0	1.78
WUTHERING	0	0	2.58

(To simplify this example, we don't do idf weighting.)

Cosine: Example

log frequency weighting

term	SaS	PaP	WH
AFFECTION	3.06	2.76	2.30
JEALOUS	2.0	1.85	2.04
GOSSIP	1.30	0	1.78
WUTHERING	0	0	2.58

log frequency weighting
& cosine normalization

term	SaS	PaP	WH
AFFECTION	0.789	0.832	0.524
JEALOUS	0.515	0.555	0.465
GOSSIP	0.335	0.0	0.405
WUTHERING	0.0	0.0	0.588

- $\cos(\text{SaS}, \text{PaP}) \approx 0.789 * 0.832 + 0.515 * 0.555 + 0.335 * 0.0 + 0.0 * 0.0 \approx 0.94$.
- $\cos(\text{SaS}, \text{WH}) \approx 0.79$
- $\cos(\text{PaP}, \text{WH}) \approx 0.69$
- Why do we have $\cos(\text{SaS}, \text{PaP}) > \cos(\text{SAS}, \text{WH})$?

Computing the cosine score

COSINESCORE(q)

```
1  float Scores[N] = 0
2  float Length[N]
3  for each query term  $t$ 
4  do calculate  $w_{t,q}$  and fetch postings list for  $t$ 
5      for each pair( $d, tf_{t,d}$ ) in postings list
6          do  $Scores[d] + = w_{t,d} \times w_{t,q}$ 
7  Read the array  $Length$ 
8  for each  $d$ 
9  do  $Scores[d] = Scores[d] / Length[d]$ 
10 return Top  $K$  components of  $Scores[]$ 
```

Components of tf-idf weighting

Term frequency		Document frequency		Normalization	
n (natural)	$tf_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(tf_{t,d})$	t (idf)	$\log \frac{N}{df_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2 + w_2^2 + \dots + w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times tf_{t,d}}{\max_t(tf_{t,d})}$	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)	$\frac{1 + \log(tf_{t,d})}{1 + \log(\text{ave}_{t \in d}(tf_{t,d}))}$				

Best known combination of weighting options Default: no

weighting

tf-idf example

- We often use **different weightings** for queries and documents.
- Notation: ddd.qqq
- Example: Inc.ltn
- document: logarithmic tf, no df weighting, cosine normalization
- query: logarithmic tf, idf, no normalization
- **Isn't it bad to not idf-weight the document?**
- Example query: “best car insurance”
- Example document: “car insurance auto insurance”

tf-idf example: Inc.ltn

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

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

$$1/1.92 \approx 0.52$$

$$1.3/1.92 \approx 0.68 \quad \text{Final similarity score between query and}$$

$$\text{document: } \sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08 \quad \text{Questions?}$$

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

Take-away today

- **Ranking** search results: why it is important (as opposed to just presenting a set of unordered Boolean results)
- **Term frequency**: This is a key ingredient for ranking.
- **Tf-idf ranking**: best known traditional ranking scheme
- **Vector space model**: Important formal model for information retrieval (along with Boolean and probabilistic models)

Introduction to Information Retrieval

<http://informationretrieval.org>

IIR 7: Scores in a Complete Search System

Hinrich Schütze

Center for Information and Language Processing, University of Munich

2014-05-07

Why is ranking so important?

- Last lecture: Problems with unranked retrieval
 - Users want to look at a few results – not thousands.
 - It's very hard to write queries that produce a few results.
 - Even for expert searchers
 - → Ranking is important because it effectively **reduces a large set of results to a very small one.**
- Next: More data on “users only look at a few results”

Empirical investigation of the effect of ranking

- The following slides are from Dan Russell's JCDL 2007 talk
- Dan Russell was the "Über Tech Lead for Search Quality & User Happiness" at Google.
- How can we measure how important ranking is?
- Observe what searchers do when they are searching in a controlled setting
 - Videotape them
 - Ask them to "think aloud"
 - Interview them
 - Eye-track them
 - Time them
 - Record and count their clicks



So.. Did you notice the FTD official site?

To be honest, I didn't even look at that.

At first I saw "from \$20" and \$20 is what I was looking for.

To be honest, 1800-flowers is what I'm familiar with and why I went there next even though I kind of assumed they wouldn't have \$20 flowers

And you knew they were expensive?

I knew they were expensive but I thought "hey, maybe they've got some flowers for under \$20 here..."

But you didn't notice the FTD?

No I didn't, actually... that's really funny.

Interview video

Rapidly scanning the results

Note scan pattern:

Page 3:
Result 1
Result 2
Result 3
Result 4
Result 3
Result 2
Result 4
Result 5
Result 6 <click>

Q: Why do this?

A: What's learned later influences judgment of earlier content.

The screenshot shows a Google search for "children's unicycle". The search results are listed under the "Web" tab. A red arrow starts at the search bar and points to the first result, then moves down to the second, then to the third, then to the fourth, then to the fifth, then to the sixth, and finally to the seventh result. The results are:

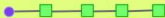
- 1** [Unicycle UK.com - FAQs - What size?](#)
12" wheel unicycle: this is a small children's unicycle size. It's good for children who are too small to ride a 16" unicycle, but it needs smooth ground ...
[www.unicycle.uk.com/FAQ.asp?Category=53 - 23k - Cached - Similar pages](#)
- 2** [Selecting a unicycle Unicycle.com NZ : buy a unicycle or learn ...](#)
16" wheel unicycle: this is a children's unicycle, the small wheel makes it only suitable for smooth areas. Best used indoors or on smooth ground. ...
[www.unicycle.co.nz/View.php?action=Page&Name=Selectingaunicycle - 22k - Cached - Similar pages](#)
- 3** [100 Miles for Kids - The Goal](#)
The Afghan Mobile Mini Circus for Children is an established ... attempt to break the GUINNESS WORLD RECORD for the ONE HOUR UNICYCLE DISTANCE RECORD. ...
[www.unicycle4kids.org/ - 9k - Cached - Similar pages](#)
- 4** [Unicycles page at Juggling World](#)
This is a children's unicycle, the small wheel makes it only suitable for very smooth areas. Best used indoors or on smooth ground, not so good outdoors ...
[www.jugglingworld.biz/shop/products_unicycles.html - 100k - Cached - Similar pages](#)
- 5** [Buy a Unicycle Unicycle.com AU : buy a unicycle or learn unicycling](#)
Check out a Unicycle Learners Pack for an easy and economical way to take your first steps into the One Wheeled World ... Suitable as a Children's Unicycle ...
[www.unicycle.au.com/View.php?action=Page&Name=Unicycles - 10k - Cached - Similar pages](#)
- 6** [Article - News - A unicycle ride for children](#)
Adam Brody, 21, of San Juan Capistrano, led a charity event Saturday that benefits the Orangewood Children's Foundation. The Unicycle Club of Southern ...
[www.ocregister.com/ocregister/news/homepage/article_1293785.php - 31k - Cached - Similar pages](#)

Kinds of behaviors we see in the data

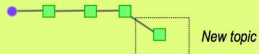
Short / Nav



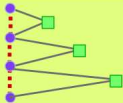
Topic exploration



Topic switch



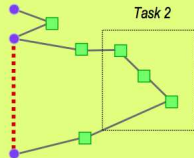
Methodical results exploration



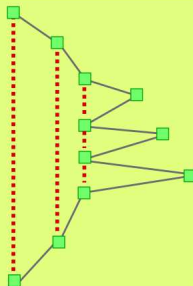
Query reform



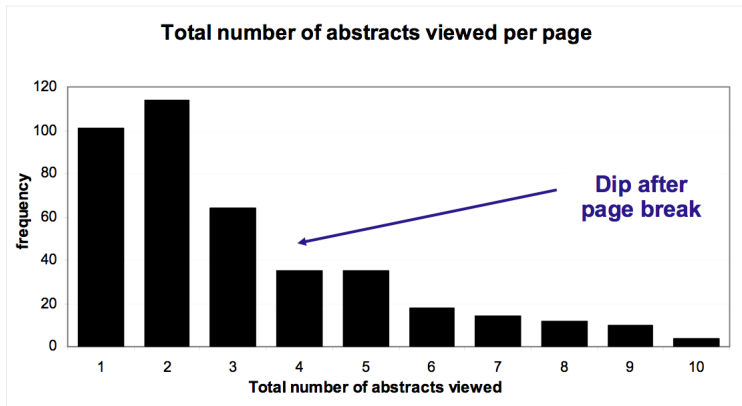
Multitasking



Stacking behavior

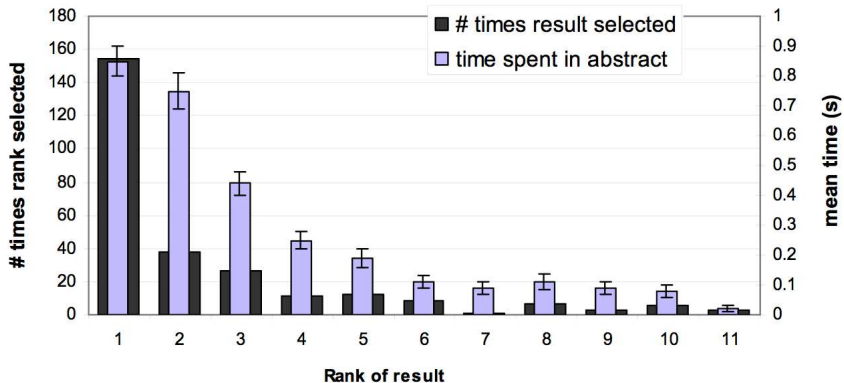


How many links do users view?



Mean: 3.07 Median/Mode: 2.00

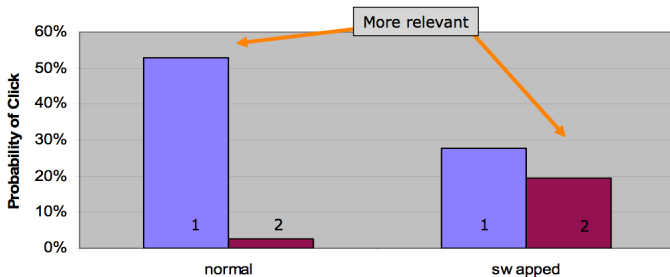
Looking vs. Clicking



- Users view results one and two more often / thoroughly
- Users click most frequently on result one

Presentation bias – reversed results

- Order of presentation influences where users look **AND** where they click



Importance of ranking: Summary

- **Viewing abstracts:** Users are a lot more likely to read the abstracts of the top-ranked pages (1, 2, 3, 4) than the abstracts of the lower ranked pages (7, 8, 9, 10).
- **Clicking:** Distribution is even more skewed for clicking
- In 1 out of 2 cases, users click on the top-ranked page.
- Even if the top-ranked page is not relevant, 30% of users will click on it.
- → Getting the ranking right is very important.
- → Getting the top-ranked page right is most important.

Outline

- 1 Recap
- 2 Why rank?
- 3 More on cosine
- 4 The complete search system**
- 5 Implementation of ranking

Complete search system

