Introduction to Natural Language Processing

a course taught as B4M36NLP at Open Informatics

by members of the Institute of Formal and Applied Linguistics

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Today:	**Week 7, lecture**
Today's topic:	**Boolen and Vector Space Models for Information Re**
Today's teacher:	**Pavel Pecina**
E-mail:	pecina@ufal.mff.cuni.cz
WWW:	http://ufal.mff.cuni.cz/∼pecina/

Contents

Introduction

Definition of *Information Retrieval*

Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

Boolean retrieval

Boolean retrieval

- 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.

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:
  1. Take the vectors for Brutus, Caesar, and Calpurnia
  2. Complement the vector of Calpurnia
  3. Do a (bitwise) and on the three vectors:

     110100 and 110111 and 101111 = 100100

0/1 vector for Brutus

	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

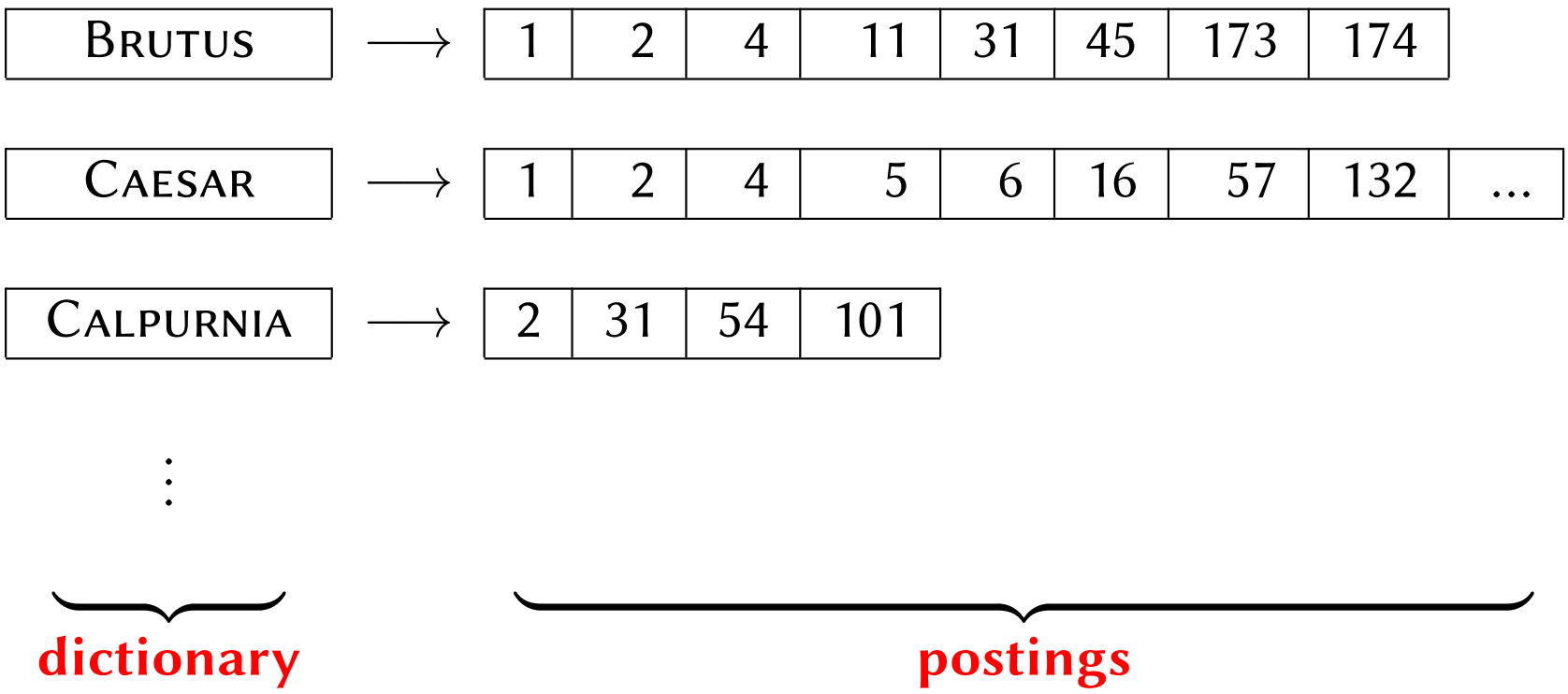
  $\Rightarrow$ $M = 500{,}000 \times 10^6 =$ half a trillion 0s and 1s.
- But the matrix has no more than one billion 1s.

  $\Rightarrow$ Matrix is extremely sparse.
- What is a better representations?

  $\Rightarrow$ We only record the 1s.

Inverted Index

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



Inverted index construction

1. Collect the documents to be indexed:
   Friends, Romans, countrymen. | So let it be with Caesar | ...
2. Tokenize the text, turning each document into a list of tokens:
   Friends | Romans | countrymen | So | ...
3. Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms: friend | roman | countryman | so ...
4. Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and 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:

$\Rightarrow$

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, sort, create lists, determine document frequency

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

⇒

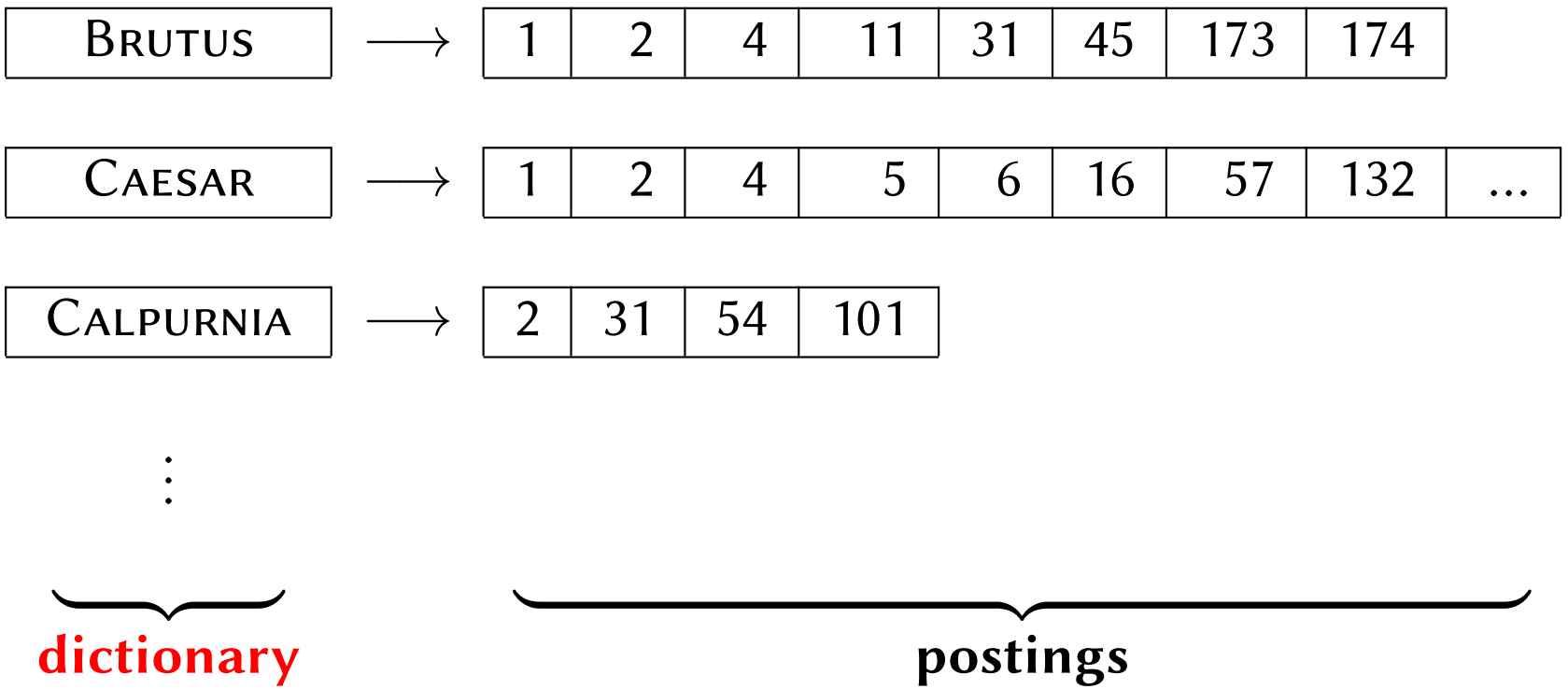
term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
i	1
i	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

⇒

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

Split the result into dictionary and postings file

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



The dictionary is the data structure for storing the term vocabulary.

Dictionary as array of fixed-width entries

- For each term, we need to store a couple of items:
  - document frequency
  - pointer to postings list
  - ...
- Assume for the time being that we can store this information in a fixed-length entry.
- Assume that we store these entries in an array.

Dictionary as array of fixed-width entries

Dictionary:

term	document frequency	pointer to postings list
a	656,265	⟶
aachen	65	⟶
...	...	...
zulu	221	⟶

Space needed: 20 bytes 4 bytes 4 bytes

1. How do we look up a query term q_i in this array at query time?
2. Which data structure do we use to locate the entry (row) in the array where q_i is stored?

Data structures for looking up term

- Two main classes of data structures: hashes and trees.
- Some IR systems use hashes, some use trees.
- Criteria for when to use hashes vs. trees:
  1. Is there a fixed number of terms or will it keep growing?
  2. What are the frequencies with which various keys will be accessed?
  3. How many terms are we likely to have?

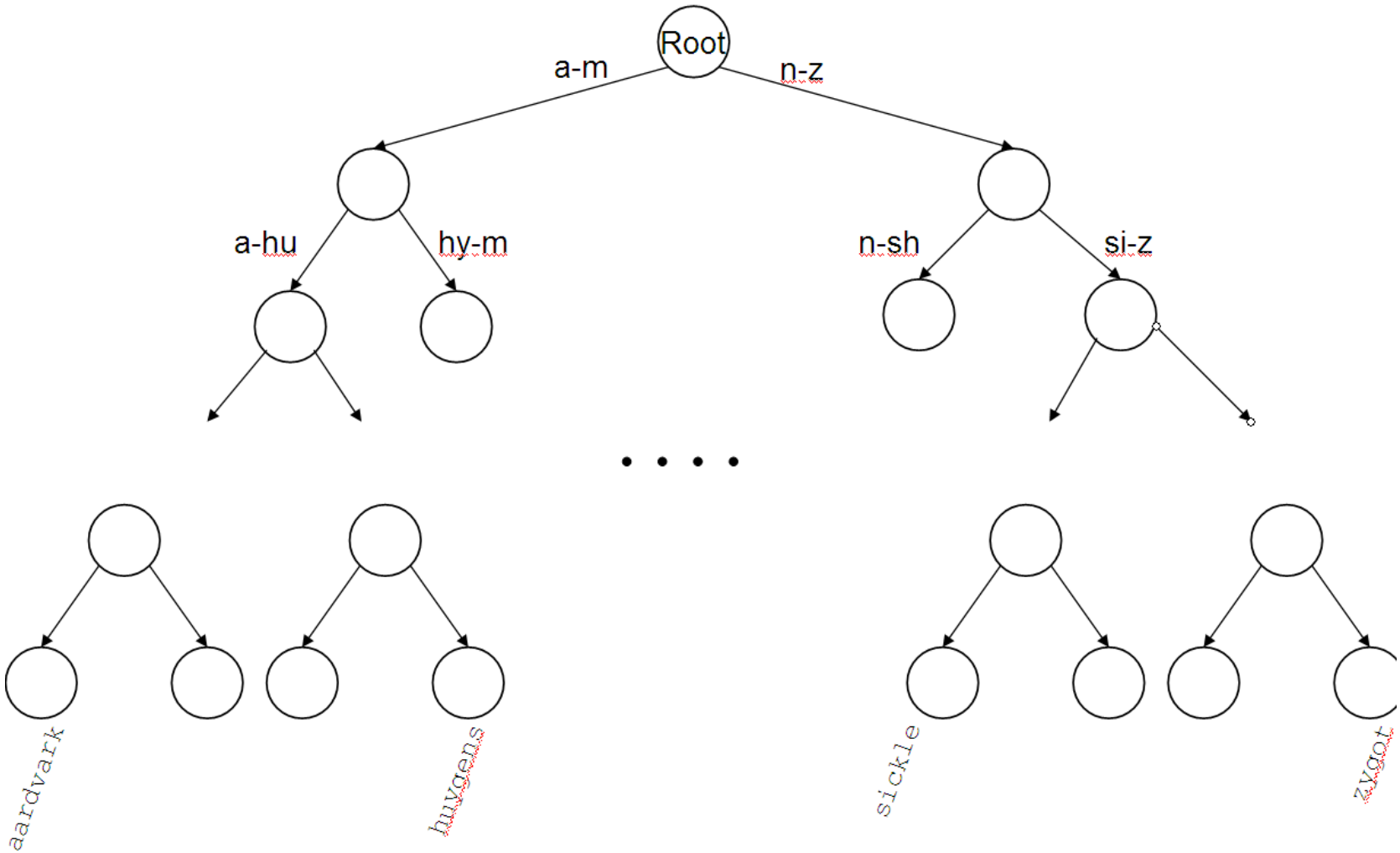
Hashes

- Each vocabulary term is hashed into an integer.
- Try to avoid collisions
- At query time, do the following: hash query term, resolve collisions, locate entry in fixed-width array
- Pros:
  1. Lookup in a hash is faster than lookup in a tree.
  2. Lookup time is constant.
- Cons:
  1. no way to find minor variants (*resume* vs. *résumé*)
  2. no prefix search (all terms starting with *automat*)
  3. need to rehash everything periodically if vocabulary keeps growing

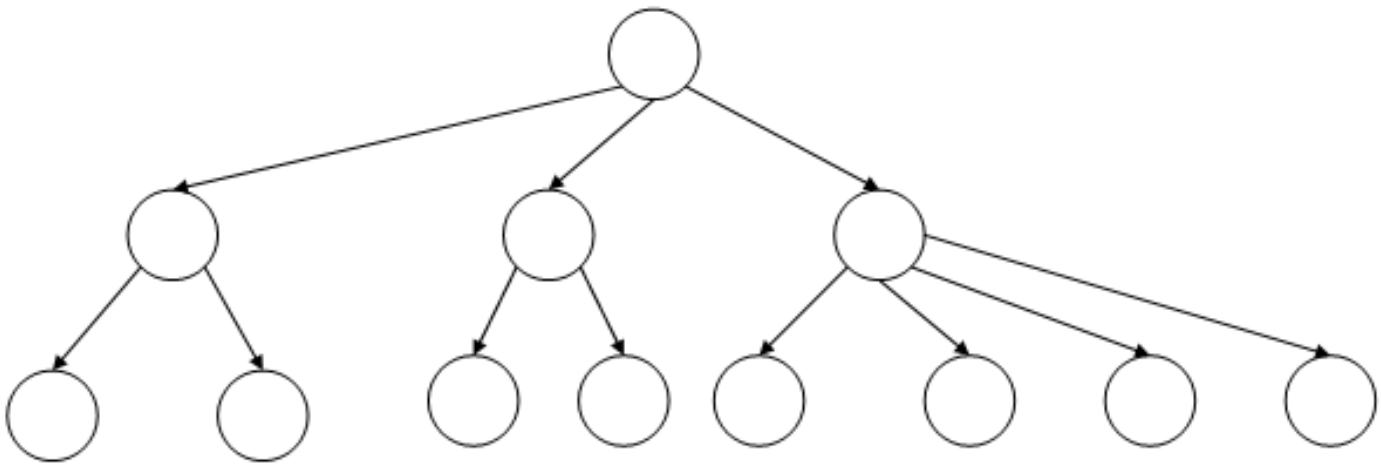
Trees

- Trees solve the prefix problem (e.g. find all terms starting with *auto*).
- Search is slightly slower than in hashes: $O(\log M)$, where M is the size of the vocabulary
- $O(\log M)$ only holds for balanced trees. Rebalancing is expensive.
- B-trees mitigate the rebalancing problem.
- B-tree definition: every internal node has a number of children in the interval $[a, b]$ where a, b are appropriate positive integers, e.g., $[2, 4]$.
- Simplest tree: binary tree

Binary tree example



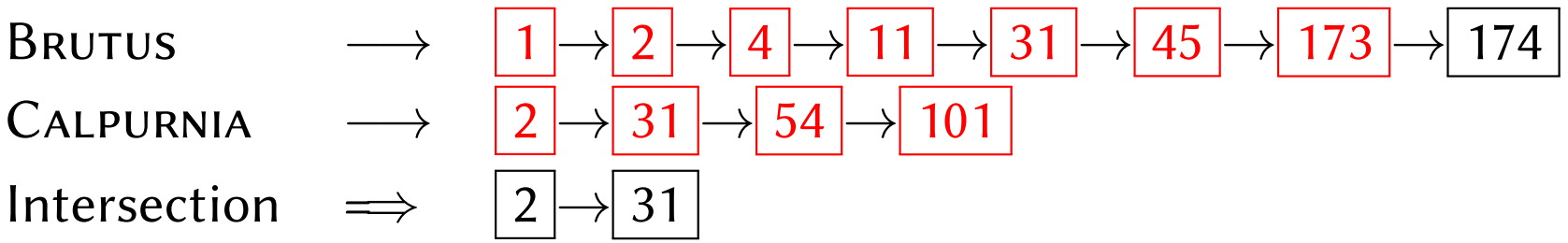
B-tree example



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.

Boolean queries

- Boolean model can answer any query that is a Boolean expression.
  - Boolean queries 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.

Text processing

Parsing a document

- We need to deal with format and language of each document.
- What format is it in? pdf, word, excel, html etc.
- What language is it in?
- What character set is in use?

Definitions

- Word – A delimited string of characters as it appears in the text.
- Term – A "normalized" word (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" terms in indexed text as well as query terms into the same form.
  - We want to match *U.S.A.* and *USA*
- We most commonly implicitly define equivalence classes of terms.
- Normalization and language detection interact.
  - *PETER WILL NICHT MIT.* $\rightarrow$ MIT $=$ mit
  - *He got his PhD from MIT.* $\rightarrow$ MIT $\neq$ mit
- Numbers
  - 3/20/91 vs. 20/3/91
  - (800) 234-2333 vs. 800.234.2333

Tokenization

- What are the delimiters? Space? Apostrophe? Hyphen?
- For each of these: sometimes they delimit, sometimes they don't.
  - 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
- No whitespace in many languages! (e.g., Chinese)
- No whitespace in Dutch, German, Swedish compounds (*Lebensversicherungsgesellschaftsangestellter*)

Accents and diacritics

- Accents: résumé vs. resume (simple omission of accent)
- Umlauts: Universität vs. Universitaet (substitution "ä" and "ae")
- Most important criterion: How are users likely to write their queries for these words?
- Even in languages that standardly have accents, users often do not type them (e.g. Czech)

Case folding

- Reduce all letters to lower case
- Possible exceptions: capitalized words in mid-sentence

  Example: 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: phonetic equivalence, e.g. *Muller = Mueller*
- Thesauri: semantic equivalence, e.g. *car = automobile*

Lemmatization

- Reduce inflectional/variant forms to base form
- Examples:
  - *am, are, is* $\rightarrow$ *be*
  - *car, cars, car's, cars'* $\rightarrow$ *car*
  - *the boy's cars are different colors* $\rightarrow$ *the boy car be different color*
- Lemmatization implies doing "proper" reduction to dictionary headword form (the lemma).
- Two types:
  - inflectional (*cutting* $\rightarrow$ *cut*)
  - derivational (*destruction* $\rightarrow$ *destroy*)

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 (derivational): *automate, automatic, automation* all reduce to *automat*

Porter algorithm (1980)

- Most common algorithm for stemming English
- Results suggest that it is at least as good as other stemming options (1980!)
- Conventions + 5 phases of reductions applied sequentially
- Each phase consists of a set of commands.
- Sample command: Delete final *ement* if what remains is longer than 1 character (replacement $\rightarrow$ replac, cement $\rightarrow$ 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			**Example**		
SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
SS	→	SS	caress	→	caress
S	→		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 analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

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}*
- Queries where stemming hurts:
  - [OPERATIONAL RESEARCH], [OPERATING SYSTEM], [OPERATIVE DENTISTRY]
  - Porter Stemmer equivalence class *oper* contains all of *operate, operating, operates, operation, operative, operatives, operational.*

Ranked retrieval

Ranked retrieval

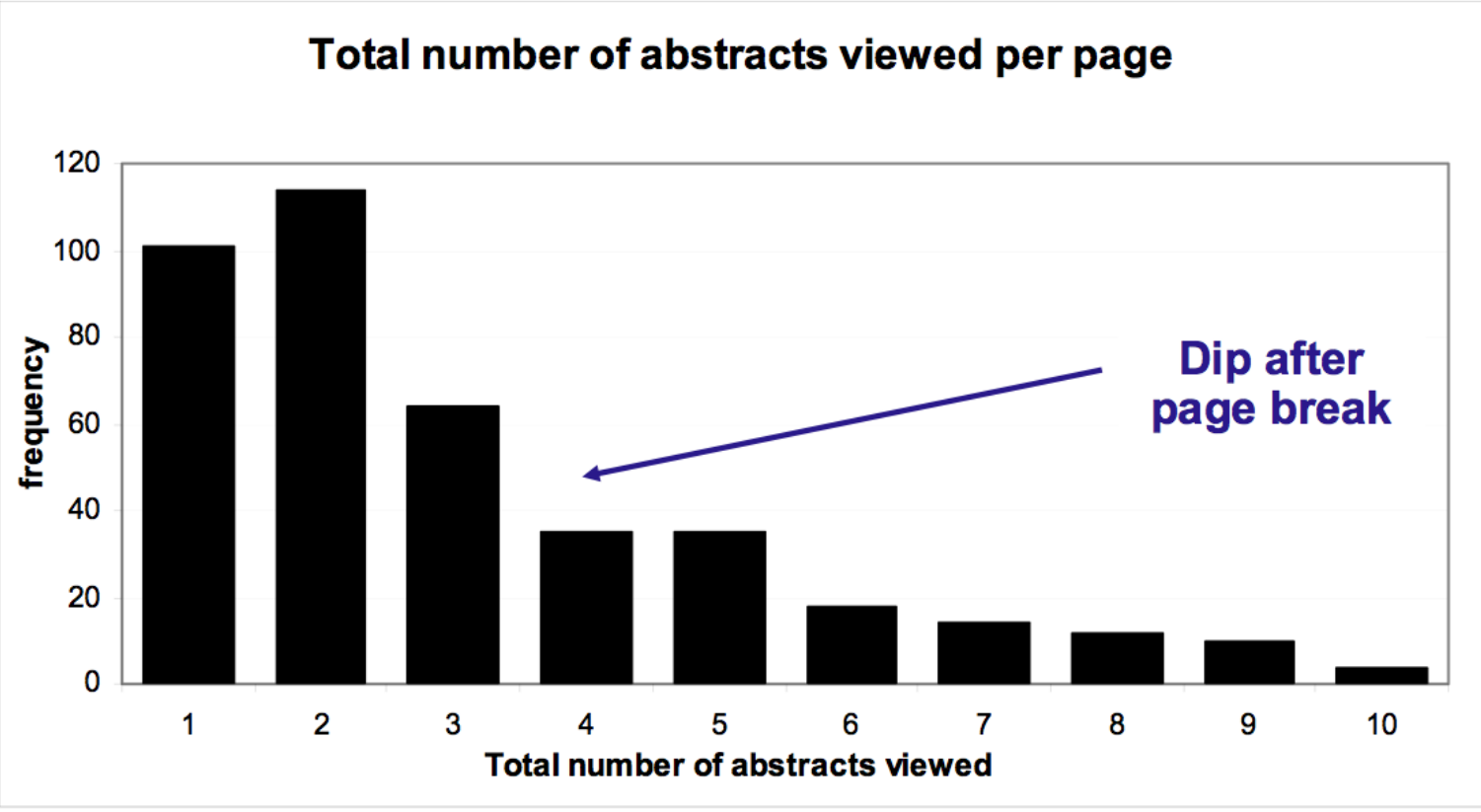
- So far, our queries have been boolean - document is a match or not.
- Good for experts: precise understanding of the needs and collection.
- Good for applications: can easily consume thousands of results.
- Not good for the majority of users.
- Most users are not capable or lazy to write Boolean queries.
- Most users don't want to wade through 1000s of results.
- This is particularly true of web search.

Problem with Boolean search

- Boolean queries often result in either too few or too many results.
  - Query 1: [standard user dlink 650] → 200,000 hits
  - Query 2: [standard user dlink 650 no card found] → 0 hits
- In Boolean retrieval, it takes a lot of skill to come up with a query that produces a manageable number of hits.
- With ranking, large result sets are not an issue.
  - Just show the top 10 results.
  - This doesn't overwhelm the user.
  - Premise: the ranking algorithm works.

...More relevant results are ranked higher than less relevant results.

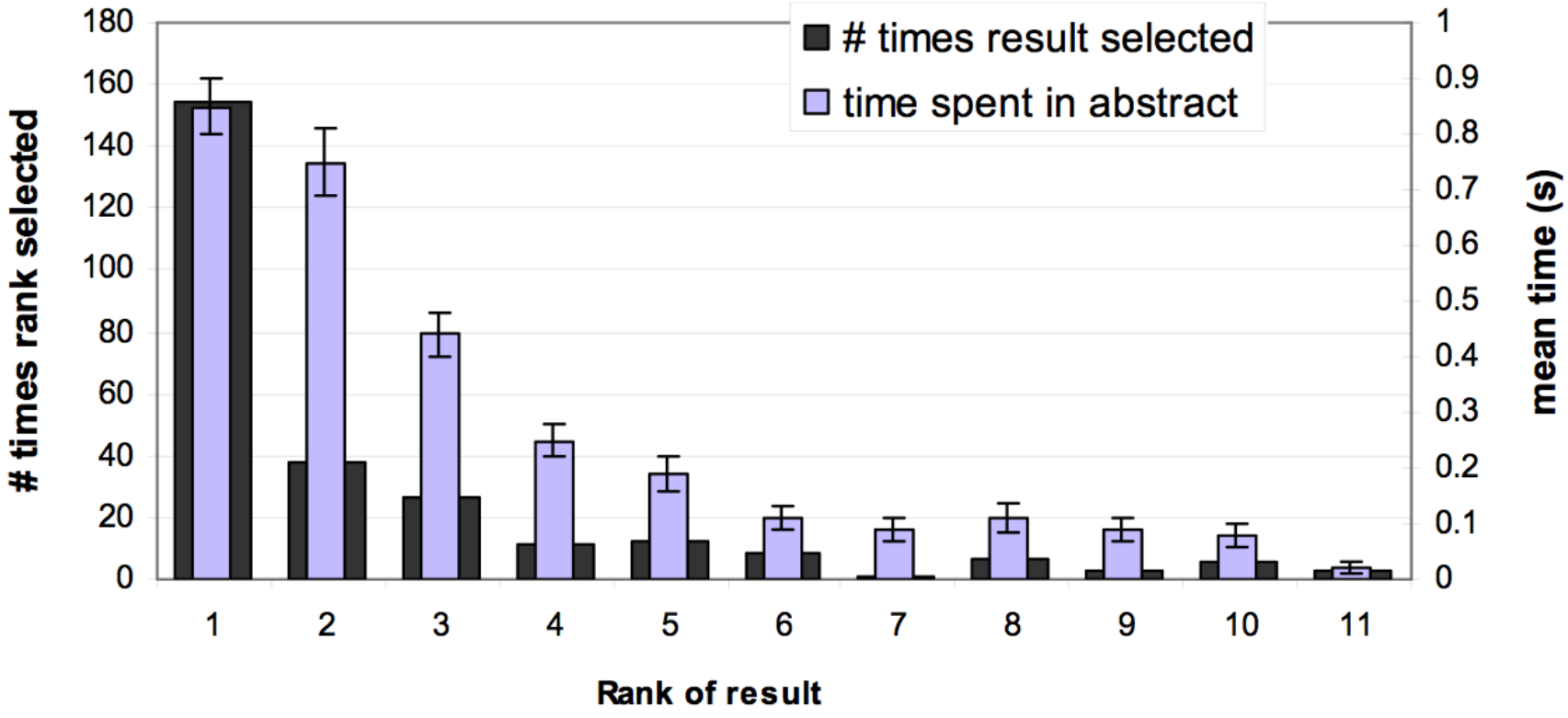
How many links do users view?



Mean: 3.07 Median/Mode: 2.00

Google

Looking vs. Clicking

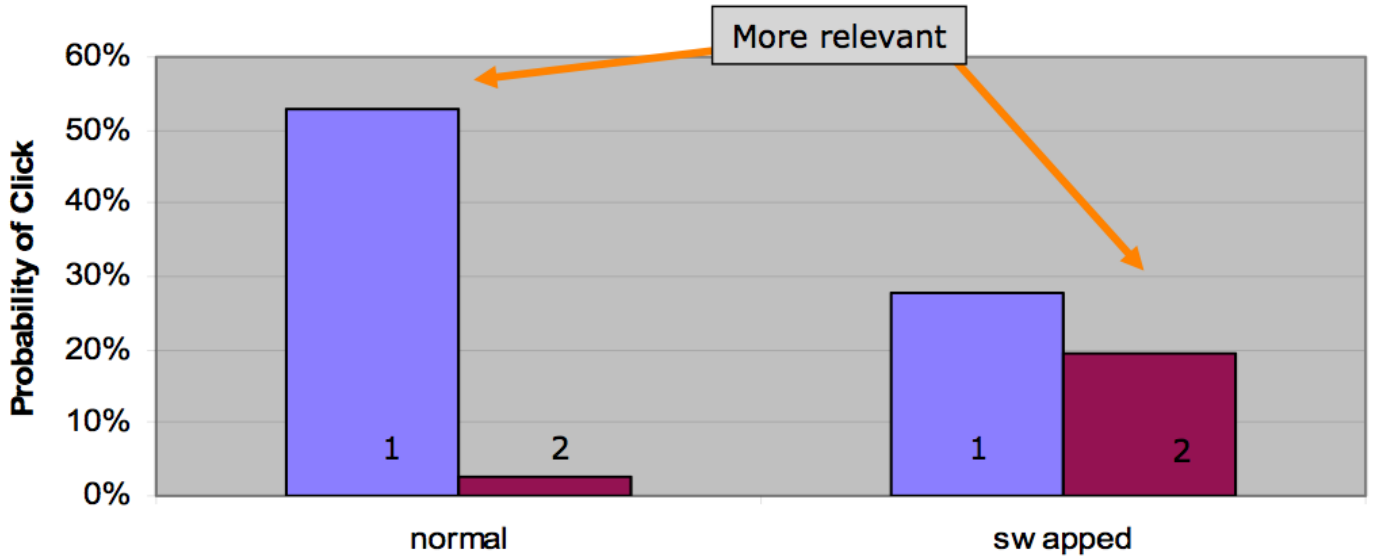


- Users view results one and two more often / thoroughly
- Users click most frequently on result one

Google

Presentation bias – reversed results

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



Google

Scoring as the basis of ranked retrieval

- We wish to rank documents that are more relevant higher than documents that are less relevant.
- How can we accomplish such a ranking of the documents in the collection 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".

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
- 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:

$$\textsc{jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}, \quad \text{where}(A \neq \emptyset \text{ or } B \neq \emptyset)$$

- $\textsc{jaccard}(A, A) = 1$
- $\textsc{jaccard}(A, B) = 0$ 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: Example

What is the query-document score the Jaccard coefficient computes for:

- Query: "ides of March"
- Document: "Caesar died in March"
- $\textsc{jaccard}(q, d) = 1/6$

What's wrong with Jaccard?

- It ignores 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.

Term weighting

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 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 $\text{tf}_{t,d}$ 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 $\text{tf} = 10$ occurrences of the term is more relevant than a document with $\text{tf} = 1$ occurrence of the term.
- But not 10 times more relevant.

Instead of raw frequency: Log frequency weighting

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

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

- $\mathrm{tf}_{t,d} \rightarrow w_{t,d}$: $0 \rightarrow 0$, $1 \rightarrow 1$, $2 \rightarrow 1.3$, $10 \rightarrow 2$, $1000 \rightarrow 4$, etc.

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

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

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

Frequency in document vs. frequency in collection

- In addition, to 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 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.

- 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

- The document frequency (df_t) is the number of documents in the collection that the term occurs in.
- df_t is an inverse measure of the informativeness of term t.
- We define the idf weight of term t in a collection of N documents as:

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

- idf_t is a measure of the informativeness of the term.
- $\log N/\text{df}_t$ instead of $[N/\text{df}_t]$ to "dampen" the effect of idf
- Note that we use the log transformation for both term frequency and document frequency.

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 (should get a higher weight)?
- This example suggests that df/idf is better for weighting than cf.

tf-idf weighting

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

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

- tf-weight
- idf-weight
- Best known weighting scheme in information retrieval.
- Increases with the number of occurrences within a document (tf).
- Increases with the rarity of the term in the collection (idf).
- Note: the "-" in tf-idf is a hyphen, not a minus (altso tf.idf, tf x idf).

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

Weight matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	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	0.00	
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	1.95	
...							

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

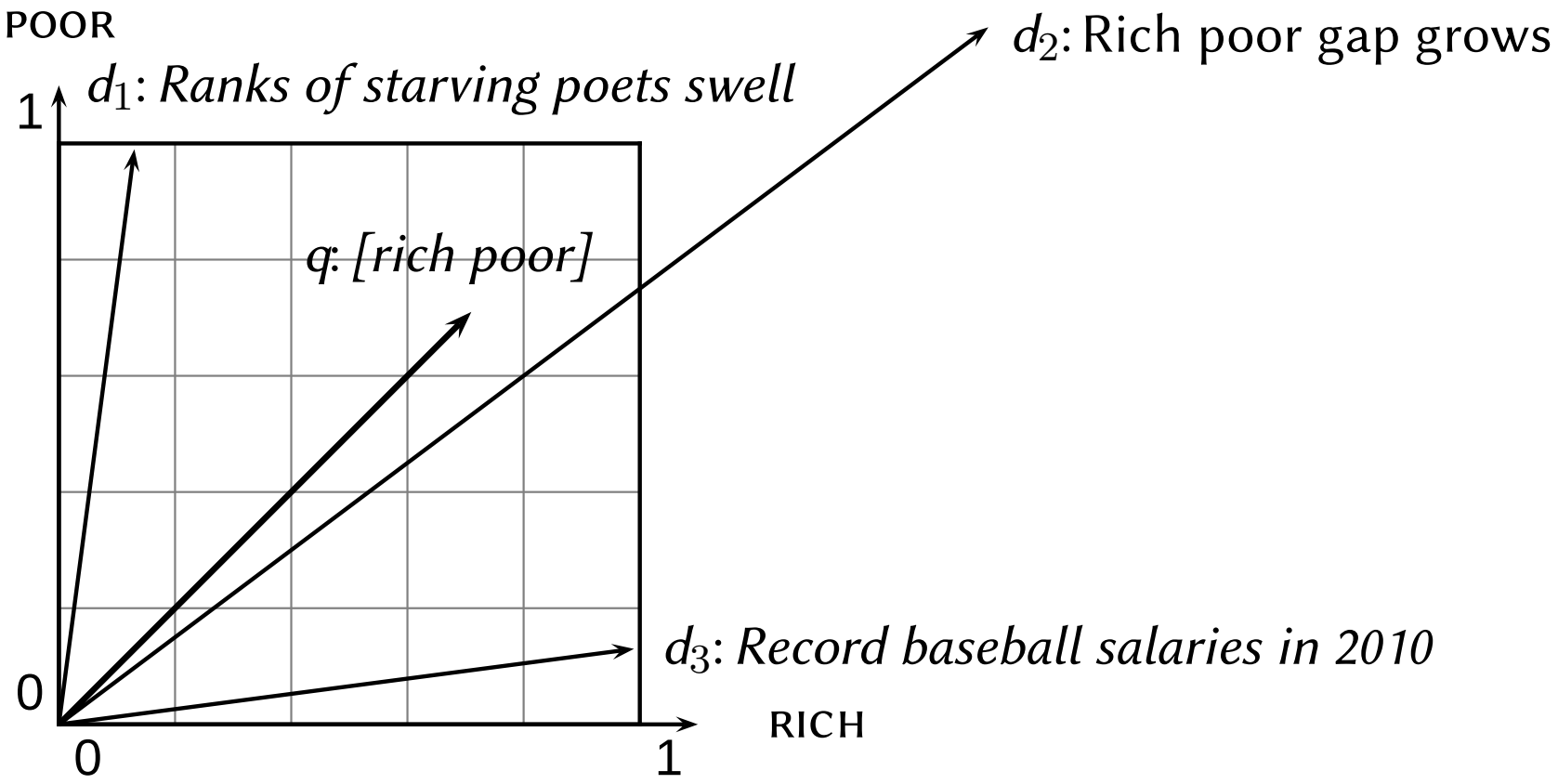
Documents as vectors

- 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/hundreds 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
- Key idea 2: Rank documents according to their proximity to query
  - proximity = similarity
  - proximity $\approx$ negative distance
- Negative distance between two points/end points of the two vectors?
- Euclidean distance?
- Bad idea – Euclidean distance is large for vectors of different lengths.

Why distance is a bad idea



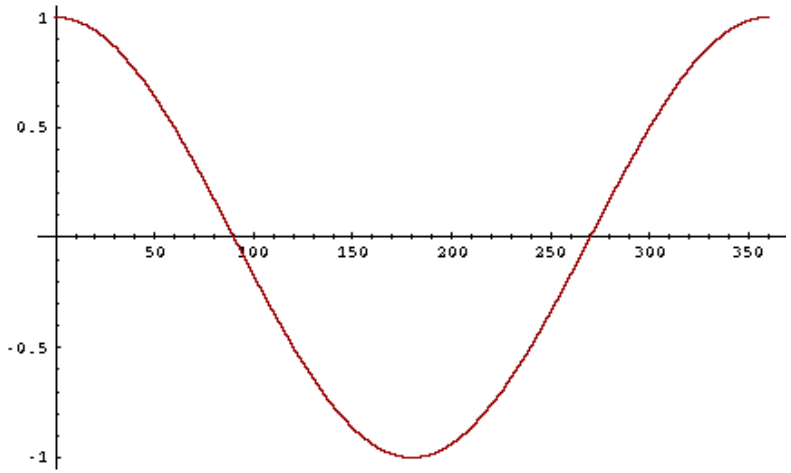
The Euclidean distance of $\vec{q}$ and $\vec{d}_2$ is large although the distribution of terms in query q and the distribution of terms in document d_2 are similar.

Use angle instead of distance

- Rank documents according to angle with query
- Ranking documents according to the angle between query and document in decreasing order

is equivalent to

- Ranking documents according to cosine(query,document) in increasing order.
- Cosine is a monotonically decreasing function of the angle for the interval $[0°, 180°]$



Length normalization

- How do we compute the cosine?
- A vector can be (length-) normalized by dividing each of its components by its length – e.g. by 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

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

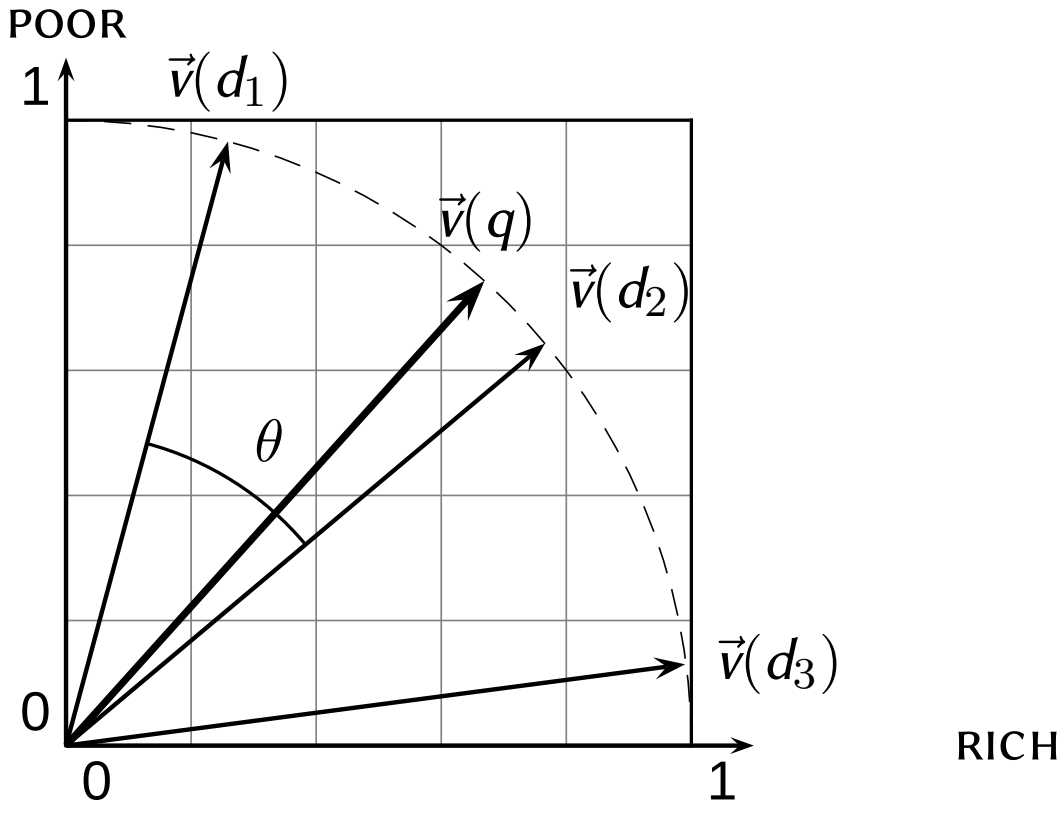
- q_i is the tf-idf weight of term i in the query.
- d_i is the tf-idf weight of term i in the document.
- $|\vec{q}|$ and $|\vec{d}|$ are the lengths of $\vec{q}$ and $\vec{d}$.

- This is the cosine similarity of $\vec{q}$ and $\vec{d}$ or, equivalently: the cosine of the angle between $\vec{q}$ and $\vec{d}$.

- For normalized vectors, the cosine is equivalent to the dot product:

$$\cos(\vec{q}, \vec{d}) = \vec{q} \cdot \vec{d} = \sum_i q_i \cdot d_i$$

Cosine similarity illustrated



Cosine: Example

How similar are these novels?

SaS: Sense and Sensibility

PaP: Pride and Prejudice

WH: Wuthering Heights

term frequencies (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 (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	0.79	0.83	0.52
JEALOUS	0.52	0.56	0.47
GOSSIP	0.34	0.0	0.41
WUTHERING	0.0	0.0	0.59

- $\cos(\text{SaS,PaP}) \approx 0.79 * 0.83 + 0.52 * 0.56 + 0.34 * 0.0 + 0.0 * 0.0 \approx 0.94$
- $\cos(\text{SaS,WH}) \approx 0.79$
- $\cos(\text{PaP,WH}) \approx 0.69$
- Why do we have $\cos(\text{SaS,PaP}) > \cos(\text{SAS,WH})$?

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

Best known combination of weighting options

Default: no weighting

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

Evaluation

Main measure: user happiness and relevance

- User happiness equated with relevance of search results to the query.
- But how do you measure relevance?
- Standard methodology in IR consists of three elements:
  1. A benchmark document collection.
  2. A benchmark suite of queries.
  3. An assessment of the relevance of each query-document pair.

Precision and recall (unranked retrieval)

- Precision (P) is the fraction of retrieved documents that are relevant:

$$\text{Precision} = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} = P(\text{relevant}|\text{retrieved})$$

- Recall (R) is the fraction of relevant documents that are retrieved:

$$\text{Recall} = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} = P(\text{retrieved}|\text{relevant})$$

Precision and recall: confusion matrix

	Relevant	Nonrelevant
Retrieved	true positives (TP)	false positives (FP)
Not retrieved	false negatives (FN)	true negatives (TN)

$$P = \frac{TP}{TP + FP} \qquad R = \frac{TP}{TP + FN}$$

Precision/recall tradeoff

- You can increase recall by returning more docs.
- Recall is a non-decreasing function of the number of docs retrieved.
- A system that returns all docs has 100% recall!
- The converse is also true (usually): It's easy to get high precision for very low recall.
- Suppose the document with the largest score is relevant. How can we maximize precision?

A combined measure: F

- The F measure allows us to trade off precision against recall.

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R} \quad \text{where} \quad \beta^2 = \frac{1-\alpha}{\alpha}$$

- $\alpha \in [0, 1]$ and thus $\beta^2 \in [0, \infty]$
- Most frequently used: balanced F with $\beta = 1$ or $\alpha = 0.5$
- This is the harmonic mean of P and R: $\frac{1}{F} = \frac{1}{2}(\frac{1}{P} + \frac{1}{R})$
- What value range of β weights recall higher than precision?

F measure: Example

	relevant	not relevant	
retrieved	20	40	60
not retrieved	60	1,000,000	1,000,060
	80	1,000,040	1,000,120

- $P = 20/(20 + 40) = 1/3$
- $R = 20/(20 + 60) = 1/4$
- $F_1 = 2\frac{1}{\frac{1}{\frac{1}{3}} + \frac{1}{\frac{1}{4}}} = 2/7$

Accuracy

- Why do we use complex measures like precision, recall, and F?
- Why not something simple like accuracy?
- Accuracy is the fraction of correct decisions (relevant/nonrelevant)
- In terms of the contingency table above:

$$A = \frac{TP + TN}{TP + FP + FN + TN}$$

- Why is accuracy not a useful measure for web information retrieval?

Why accuracy is a useless measure in IR

- The number of relavant and non-relevant documents is unbalanced.
- A trick to maximize accuracy in IR: always say no and return nothing.
- You then get 99.99% accuracy on most queries (0.01% docs relevant).
- Searchers on the web (and in IR in general) want to find something and have a certain tolerance for junk.
- It's better to return some bad hits as long as you return something.

→ We use precision, recall, and F for evaluation, not accuracy.

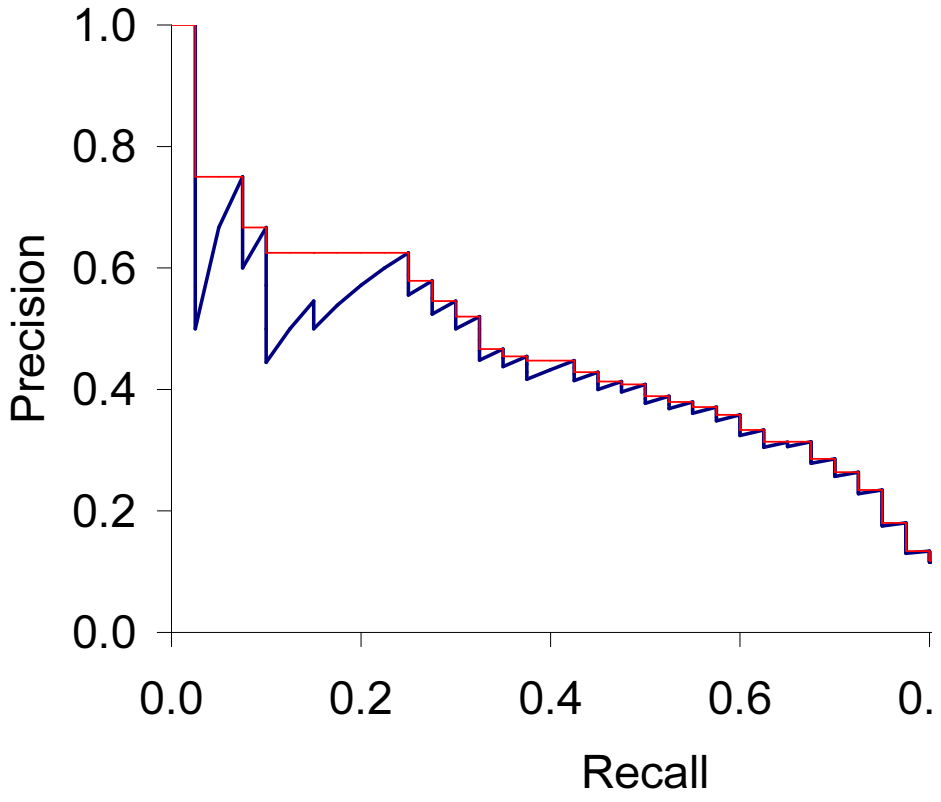
Difficulties in using precision, recall and *F* measure

- We should always average over a large set of queries.
- We need relevance judgments for information-need-document pairs – but they are expensive to produce.
- Alternatives to using precision/recall and having to produce relevance judgments exists (A/B testing).

Precision-recall curve

- Precision/recall/F are measures for unranked sets.
- We can easily turn set measures into measures of ranked lists.
- Just compute the set measure for each "prefix" of the ranked list: the top 1, top 2, top 3, top 4 etc. results.
- Doing this for precision and recall gives you a precision-recall curve.

A precision-recall curve



- Each point corresponds to a result for top k ranked hits (k=1,2,3,...)
- Interpolation (in red): Take maximum of all future points.
- Rationale for interpolation: The user is willing to look at more stuff if both precision and recall get better.

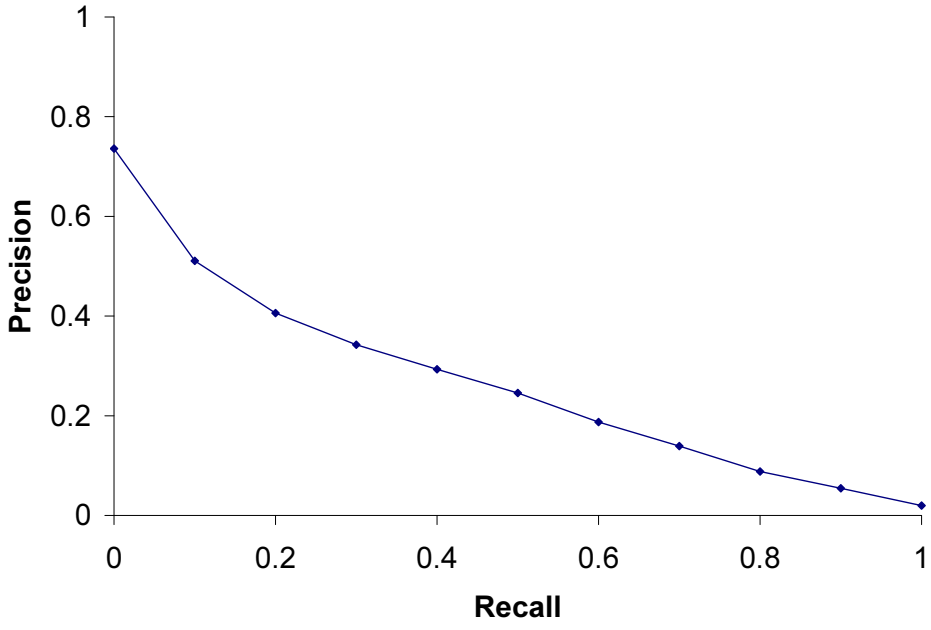
11-point interpolated average precision

Recall	Interpolated Precision
0.0	1.00
0.1	0.67
0.2	0.63
0.3	0.55
0.4	0.45
0.5	0.41
0.6	0.36
0.7	0.29
0.8	0.13
0.9	0.10
1.0	0.08

11-point average: ≈ 0.425

How can precision at 0.0 be > 0?

Averaged 11-point precision/recall graph



- Compute interpolated precision at recall levels 0.0, 0.1, 0.2, ...
- Do this for each of the queries in the evaluation benchmark.
- Average over queries.
- This measure measures performance at all recall levels.

What we need for a benchmark

1. A collection of documents
   - Must be representative of the documents we expect to see in reality.
2. A collection of information needs
   - (which we will often incorrectly refer to as queries)
   - Must be representative of the inform. needs we expect to see in reality.
3. Human relevance assessments
   - We need to hire/pay "judges" or assessors to do this.
   - Expensive, time-consuming.
   - Judges must be representative of the users we expect to see in reality.

Standard relevance benchmark: Cranfield

- Pioneering: first testbed allowing precise quantitative measures of information retrieval effectiveness.
- Late 1950s, UK.
- 1398 abstracts of aerodynamics journal articles, a set of 225 queries, exhaustive relevance judgments of all query-document-pairs.
- Too small, too untypical for serious IR evaluation today.

Standard relevance benchmark: TREC

- TREC = Text Retrieval Conference (TREC)
- Organized by National Institute of Standards and Technology (NIST)
- TREC is actually a set of several different relevance benchmarks.
- Best known: TREC Ad Hoc, used for TREC evaluations in 1992 – 1999
- 1.89 M documents, mainly newswire articles, 450 information needs
- No exhaustive relevance judgments – too expensive
- Rather, NIST assessors' relevance judgments are available only for the documents that were among the top k returned for some system which was entered in the TREC evaluation for which the information need was developed.

Standard relevance benchmarks: Others

- GOV2
  - Another TREC/NIST collection
  - 25 million web pages
  - Used to be largest collection that is easily available
  - But still 3 orders of magnitude smaller than what Google/Bing index
- NTCIR
  - East Asian language and cross-language information retrieval
- Cross Language Evaluation Forum (CLEF)
  - This evaluation series has concentrated on European languages and cross-language information retrieval.
- Many others