

6th IIT Hyderabad Advanced School on Natural Language Processing (IASNLP 2015), July 1 – 15, 2015, Hyderabad, India

Information Retrieval & Extraction

Priya Radhakrishnan,

Search and Information Extraction Lab (SIEL)

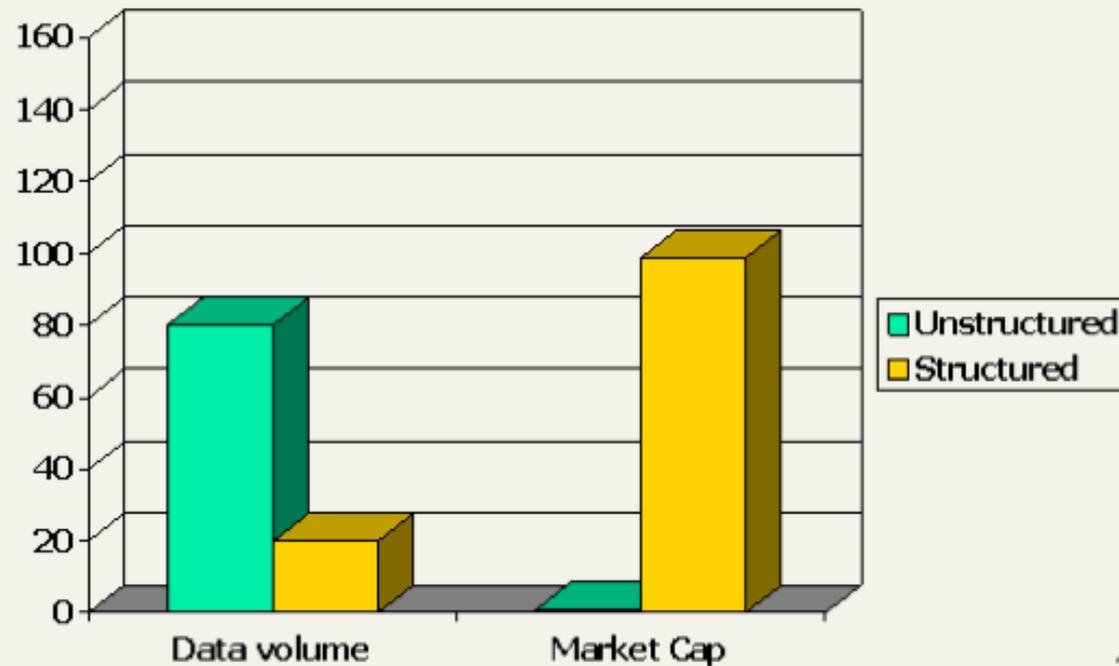
Slide Courtesy : IR - Hinrich Schütze and Christina Lioma
(<http://nlp.stanford.edu/IR-book/newslides.html>)

IE – Alexander Fraser (http://www.cis.uni-muenchen.de/~fraser/information_extraction_2013_lecture/)

Outline

- **Information Retrieval**
 - Definition and Motivation
 - Inverted Index
 - A simple Boolean information retrieval system
 - Term vocabulary and postings list
 - Dictionaries and tolerant retrieval
 - Index construction and storage
 - Scoring, term weighting and vector-space model (exercise)
 - Ranking and Cosine Similarity
- Information Extraction
 - Entity Extraction & Relation Extraction

Unstructured (text) vs. structured (database) data in 1996



2

Unstructured (text) vs. structured (database) data in 2006



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).

Outline

- Information Retrieval
 - Definition and Motivation
 - **Inverted Index**
 - A simple Boolean information retrieval system
 - Term vocabulary and postings list
 - Dictionaries and tolerant retrieval
 - Index construction and storage
 - Scoring, term weighting and vector-space model (exercise)
 - Ranking and Cosine Similarity
- Information Extraction
 - Entity Extraction & Relation Extraction

Initial approaches

- GREP
- Index – large collection, flexible matching, ranked retrieval
- Incidence matrix
- 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.

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
...						

Unstructured data in 1650 : Shakespeare

- Which plays of Shakespeare contain the words BRUTUS AND CAESAR, but not CALPURNIA?
- One could grep all of Shakespeare's plays for BRUTUS 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 NOT CALPURNIA
 - Complement the vector of CALPURNIA
 - Do a (bitwise) and on the three vectors
 - $110100 \text{ AND } 110111 \text{ AND } 101111 = 100100$

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

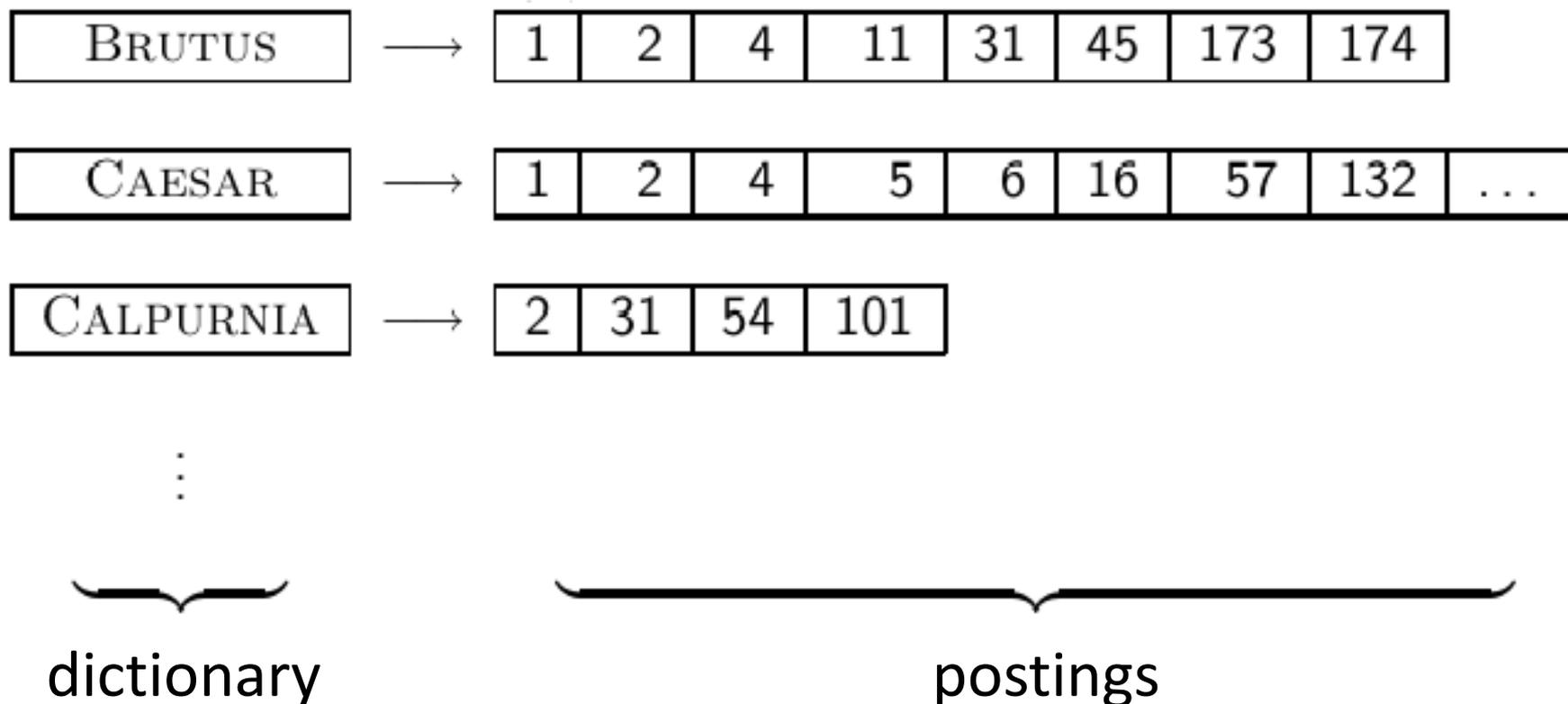
- Consider $N = 10^6$ documents, each with about 1000 tokens
- \Rightarrow total of 10^9 tokens
- On average 6 bytes per token, including spaces and punctuation \Rightarrow size of document collection is about $6 \cdot 10^9 = 6$ GB
- Assume there are $M = 500,000$ distinct terms in the collection
- (Notice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

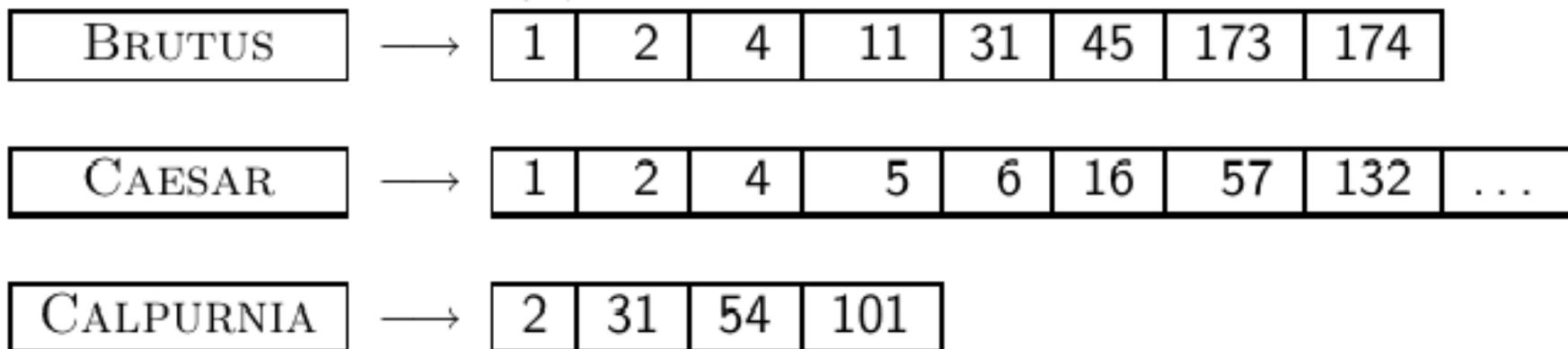
Inverted Index

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



Inverted Index

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



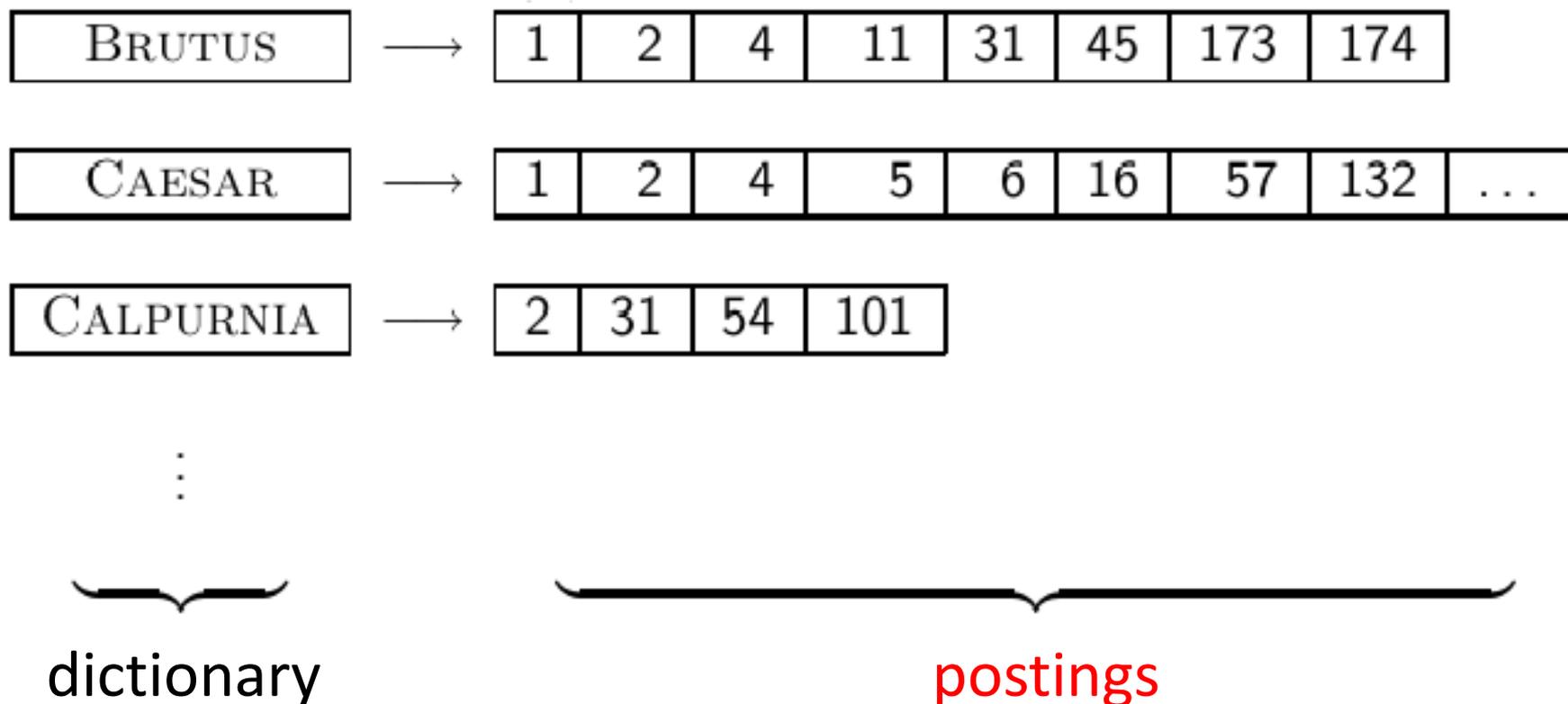
⋮

dictionary

postings

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.

Tokenizing 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 posting

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

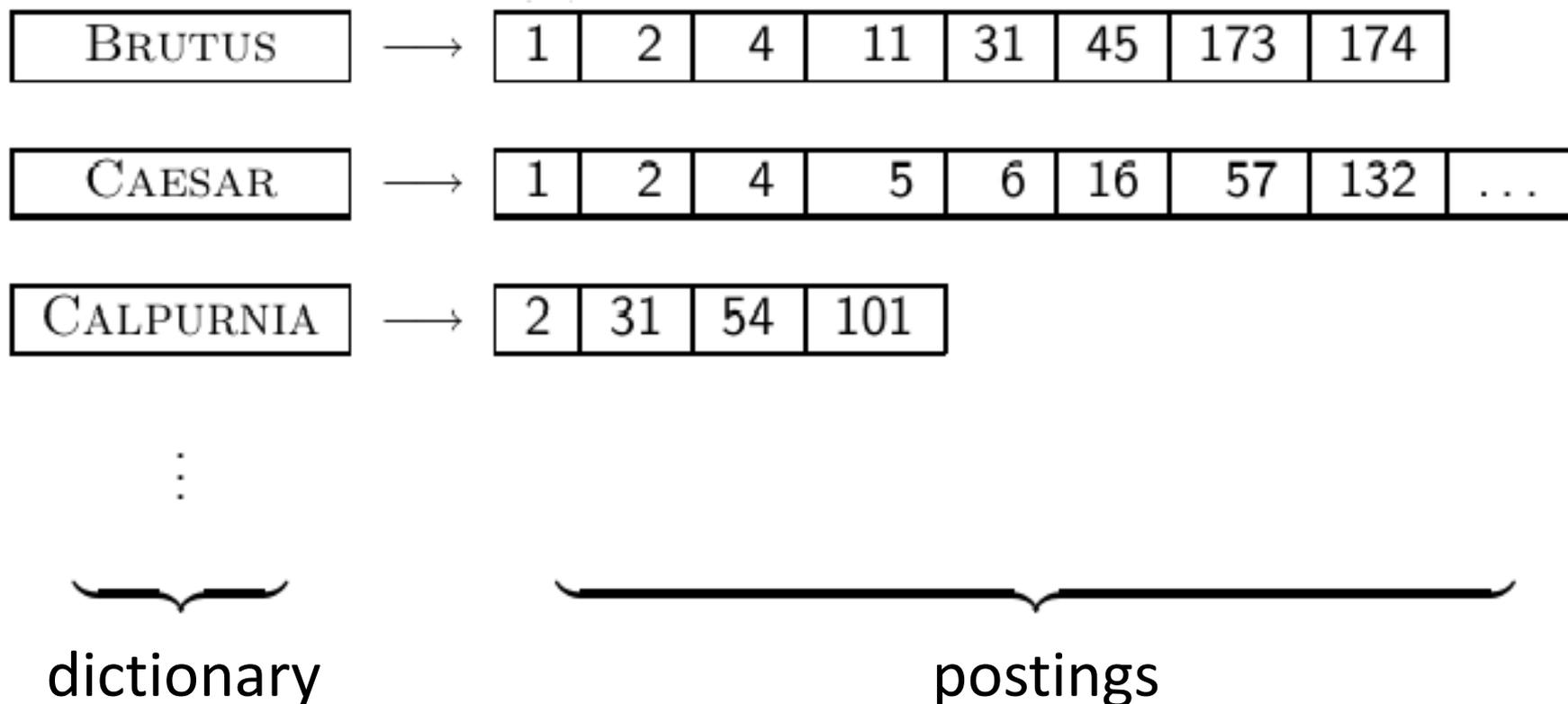
term	docID		term	docID
i	1		ambitious	2
did	1		be	2
enact	1		brutus	1
julius	1		brutus	2
caesar	1		capitol	1
i	1		caesar	1
was	1		caesar	2
killed	1		caesar	2
i'	1		did	1
the	1		enact	1
capitol	1		hath	1
brutus	1		i	1
killed	1		i	1
me	1	⇒	i'	1
so	2		it	2
let	2		julius	1
it	2		killed	1
be	2		killed	1
with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



Positional indexes

- Postings lists in a **nonpositional** index: each posting is just a docID
- Postings lists in a **positional** index: each posting is a docID and **a list of positions**
- Example query: “ $to_1 be_2 or_3 not_4 to_5 be_6$ ”

TO, 993427:

1: <7, 18, 33, 72, 86, 231>;

2: <1, 17, 74, 222, 255>;

4: <8, 16, 190, 429, 433>;

5: <363, 367>;

7: <13, 23, 191>; . . . >

BE, 178239:

1: <17, 25>;

4: <17, 191, 291, 430, 434>;

5: <14, 19, 101>; . . . > Document 4 is a match!

Positional indexes

- With a positional index, we can answer **phrase queries**.
- With a positional index, we can answer **proximity queries**.

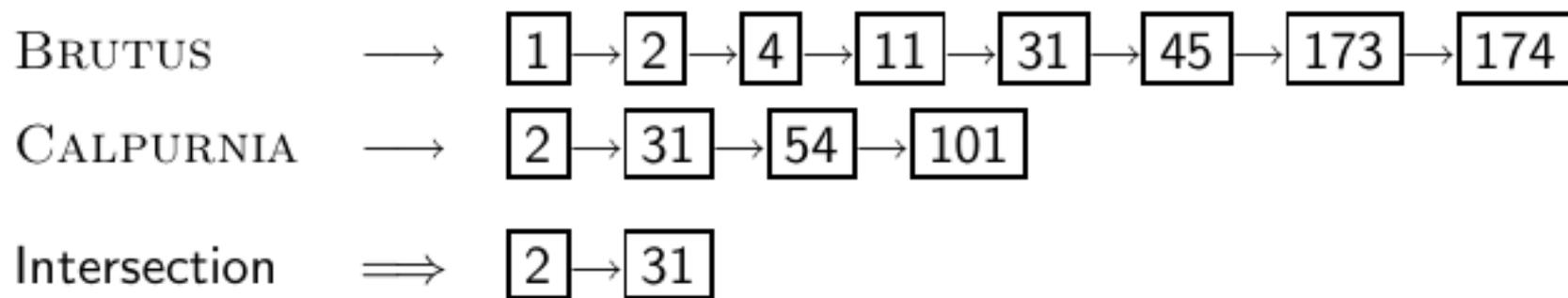
Outline

- Information Retrieval
 - Definition and Motivation
 - Inverted Index
 - A simple Boolean information retrieval system
 - Term vocabulary and postings list
 - Dictionaries and tolerant retrieval
 - Index construction and storage
 - Scoring, term weighting and vector-space model (exercise)
 - Ranking and Cosine Similarity
- Information Extraction
 - Entity Extraction & Relation Extraction

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 posting lists

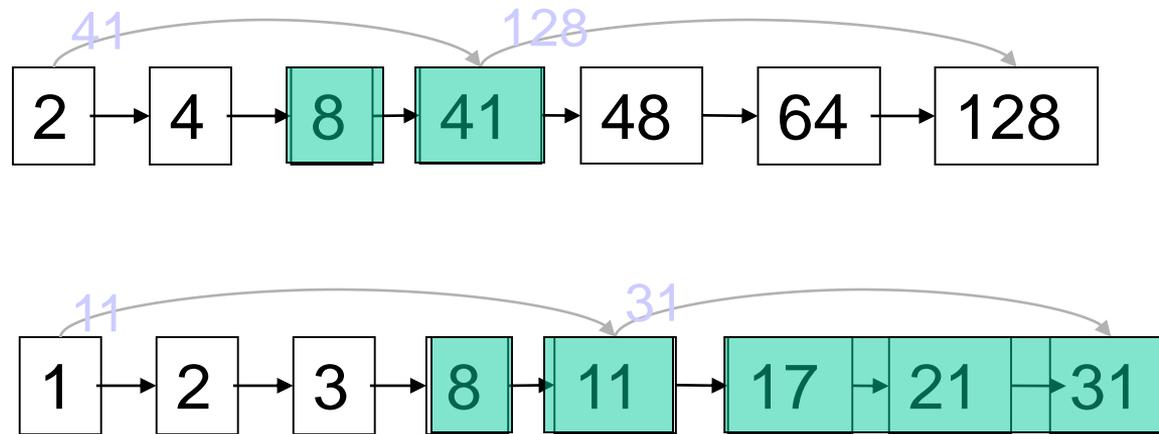


- This is linear in the length of the postings lists.
- Note: This only works if postings lists are sorted.

Intersecting two posting 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  $\text{docID}(p_1) = \text{docID}(p_2)$ 
4      then  $\text{ADD}(answer, \text{docID}(p_1))$ 
5           $p_1 \leftarrow \text{next}(p_1)$ 
6           $p_2 \leftarrow \text{next}(p_2)$ 
7      else if  $\text{docID}(p_1) < \text{docID}(p_2)$ 
8          then  $p_1 \leftarrow \text{next}(p_1)$ 
9          else  $p_2 \leftarrow \text{next}(p_2)$ 
10 return  $answer$ 
```

Query processing with skip pointers



Suppose we've stepped through the lists until we process **8** on each list. We match it and advance.

We then have **41** and **11** on the lower. **11** is smaller.

But the skip successor of **11** on the lower list is **31**, so we can skip ahead past the intervening postings.

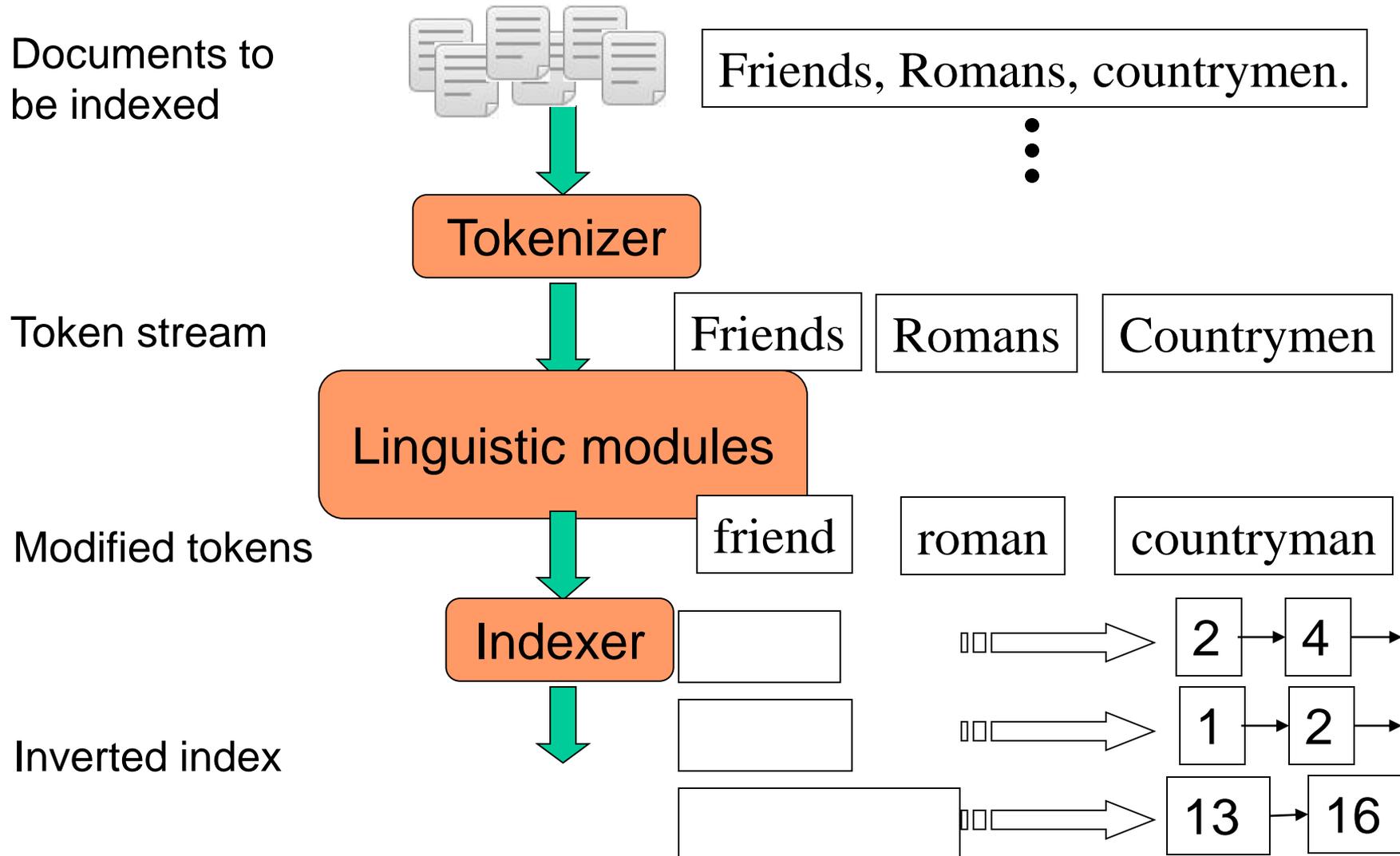
Boolean queries

- 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.

Outline

- Information Retrieval
 - Definition and Motivation
 - Inverted Index
 - A simple Boolean information retrieval system
 - Term vocabulary and postings list
 - Dictionaries and tolerant retrieval
 - Index construction and storage
 - Scoring, term weighting and vector-space model (exercise)
 - Ranking and Cosine Similarity
- Information Extraction
 - Entity Extraction & Relation Extraction

The basic indexing pipeline

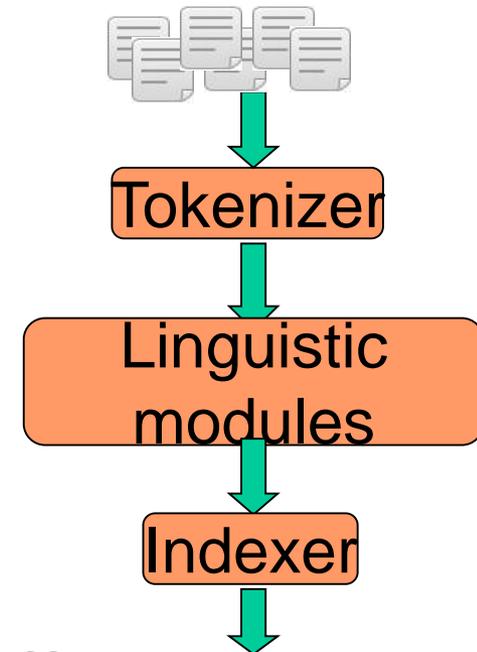


Parsing a document

- What format is it in?
 - pdf/word/excel/html?
- What language is it in?
- What character set is in use?
 - (CP1252, UTF-8, ...)

Each of these is a classification problem.

But these tasks are often done heuristically ...



Complications: What is a document?

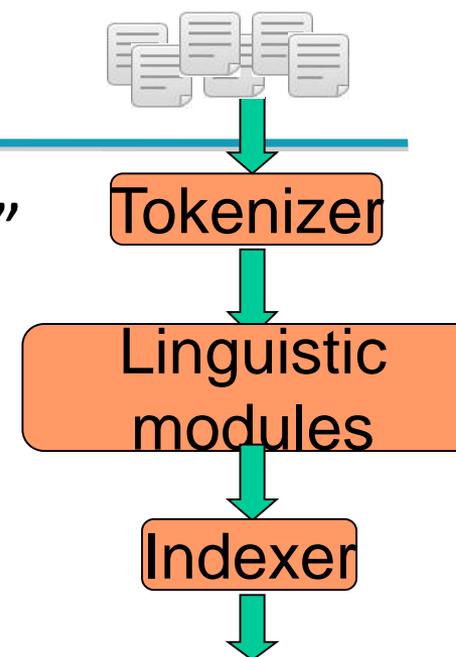
We return from our query “documents” but there are often interesting questions of grain size:

What is a unit document?

- A file?
- An email? (Perhaps one of many in a single mbox file)
 - What about an email with 5 attachments?
- A group of files (e.g., PPT or LaTeX split over HTML pages)

Tokenization

- Input: “*Friends, Romans and Countrymen*”
- Output: Tokens
 - *Friends*
 - *Romans*
 - *Countrymen*
- A **token** is an instance of a sequence of characters
- Each such token is now a candidate for an index entry, after further processing
 - Described below
- But what are valid tokens to emit?



Challenges :Tokenization

- Issues in tokenization:
 - ***Finland's capital*** →
Finland AND ***s***? ***Finlands***? ***Finland's***?
 - ***Hewlett-Packard*** → ***Hewlett*** and ***Packard*** as two tokens?
 - ***state-of-the-art***: break up hyphenated sequence.
 - ***co-education***
 - ***lowercase, lower-case, lower case*** ?
 - It can be effective to get the user to put in possible hyphens
 - ***San Francisco***: one token or two?
 - How do you decide it is one token?

Tokenization Challenges: Numbers

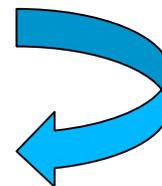
- *3/20/91* *Mar. 12, 1991* *20/3/91*
- *55 B.C.*
- *B-52*
- *My PGP key is 324a3df234cb23e*
- *(800) 234-2333*
 - Often have embedded spaces
 - Older IR systems may not index numbers
 - But often very useful: think about things like looking up error codes/stacktraces on the web
 - (One answer is using n-grams: IIR ch. 3)
 - Will often index “meta-data” separately
 - Creation date, format, etc.

Tokenization: language issues

- Agglutination in Dravidian languages
 - “ ഞാനറിഞ്ഞില്ല ” → one sentence or one word or one token?
- **जाइए जाईए जाइये जयिये**
 - Should be equivalent
- German noun compounds are not segmented
 - **Lebensversicherungsgesellschaftsangestellter**
 - ‘life insurance company employee’
 - German retrieval systems benefit greatly from a **compound splitter** module
 - Can give a 15% performance boost for German

Stop words

- With a stop list, you exclude from the dictionary entirely the commonest words. Intuition:
 - They have little semantic content: *the, a, and, to, be*
 - There are a lot of them: ~30% of postings for top 30 words
- But the trend is away from doing this:
 - Good compression techniques (IIR 5) means the space for including stop words in a system is very small
 - Good query optimization techniques (IIR 7) mean you pay little at query time for including stop words.
 - You need them for:
 - Phrase queries: “King of Denmark”
 - Various song titles, etc.: “Let it be”, “To be or not to be”
 - “Relational” queries: “flights to London”



Normalization to terms

- We may need to “normalize” words in indexed text as well as query words into the same form
 - We want to match ***U.S.A.*** and ***USA***
- Result is terms: a **term** is a (normalized) word type, which is an entry in our IR system dictionary
- We most commonly implicitly define equivalence classes of terms by, e.g.,
 - deleting periods to form a term
 - ***U.S.A., USA*** (***USA***
 - deleting hyphens to form a term
 - ***anti-discriminatory, antidiscriminatory*** (***antidiscriminatory***

Normalization: other languages

- Accents and Umlaut
- Most important criterion:
 - How are your users like to write their queries for these words?
- Even in languages that standardly have accents, users often may not type them
 - Often best to normalize to a de-accented term

Case folding

- Reduce all letters to lower case
 - exception: upper case in mid-sentence?
 - e.g., General Motors
 - Fed vs. fed
 - SAIL vs. sail
 - Often best to lower case everything, since users will use lowercase regardless of ‘correct’ capitalization...
- Longstanding Google example: [fixed in 2011...]
 - Query C.A.T.
 - #1 result is for “cats” (well, Lolcats) not Caterpillar Inc.

Normalization to terms

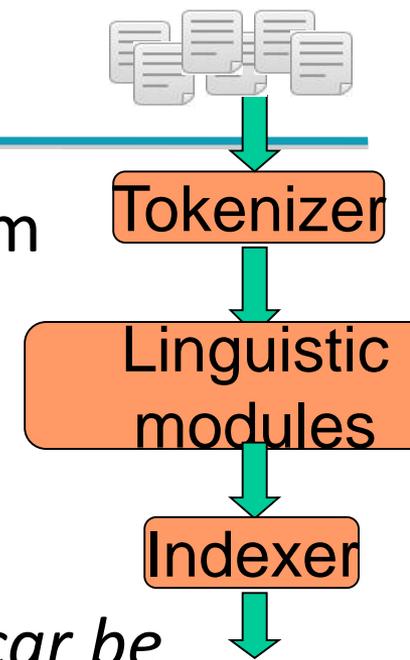
- An alternative to equivalence classing is to do asymmetric expansion
- An example of where this may be useful
 - Enter: *window* Search: *window, windows*
 - Enter: *windows* Search: *Windows, windows, window*
 - Enter: *Windows* Search: *Windows*
- Potentially more powerful, but less efficient

Thesauri and soundex

- Do we handle synonyms and homonyms?
 - E.g., by hand-constructed equivalence classes
 - *car = automobile* *color = colour*
 - We can rewrite to form equivalence-class terms
 - When the document contains *automobile*, index it under *car-automobile* (and vice-versa)
 - Or we can expand a query
 - When the query contains *automobile*, look under *car* as well
- What about spelling mistakes?
 - One approach is Soundex, which forms equivalence classes of words based on phonetic heuristics

Lemmatization

- Reduce inflectional/variant forms to base form
- E.g.,
 - *am, are, is* → *be*
 - *car, cars, car's, cars'* → *car*
- *the boy's cars are different colors* → *the boy car be different color*
- Lemmatization implies doing “proper” reduction to dictionary headword form



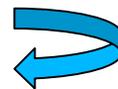
Stemming

- Reduce terms to their “roots” before indexing
- “Stemming” suggests crude affix chopping
 - language dependent
 - e.g., *automate(s)*, *automatic*, *automation* all reduced to *automat*.

for example compressed and compression are both accepted as equivalent to compress.



for exampl compress and compress ar both accept as equal to compress



Porter's algorithm

- Commonest algorithm for stemming English
 - Results suggest it's at least as good as other stemming options
- Conventions + 5 phases of reductions
 - phases applied sequentially
 - each phase consists of a set of commands
 - sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*

Other stemmers

- Other stemmers exist:
 - Lovins stemmer
 - <http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm>
 - Single-pass, longest suffix removal (about 250 rules)
 - Paice/Husk stemmer
 - Snowball
- Full morphological analysis (lemmatization)
 - At most modest benefits for retrieval

Language-specificity

- The above methods embody transformations that are
 - Language-specific, and often
 - Application-specific
- These are “plug-in” addenda to the indexing process
- Both open source and commercial plug-ins are available for handling these

Does stemming help?

- English: very mixed results. Helps recall for some queries but harms precision on others
 - E.g., operative (dentistry) \Rightarrow oper
- Definitely useful for Spanish, German, Finnish, ...
 - 30% performance gains for Finnish!

Outline

- Information Retrieval
 - Definition and Motivation
 - **Inverted Index**
 - A simple Boolean information retrieval system
 - Term vocabulary and postings list
 - **Dictionaries and tolerant retrieval**
 - Index construction and storage
 - Scoring, term weighting and vector-space model (exercise)
 - Ranking and Cosine Similarity
- Information Extraction
 - Entity Extraction & Relation Extraction

Dictionaries

- **Term vocabulary**: the data
- **Dictionary**: the **data structure** for storing the term vocabulary
- 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

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

space needed: 20 bytes 4 bytes 4 bytes

How do we look up a query term q_i in this array at query time?

That is: 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:
 - Is there a fixed number of terms or will it keep growing?
 - What are the relative frequencies with which various keys will be accessed?
 - How many terms are we likely to have?

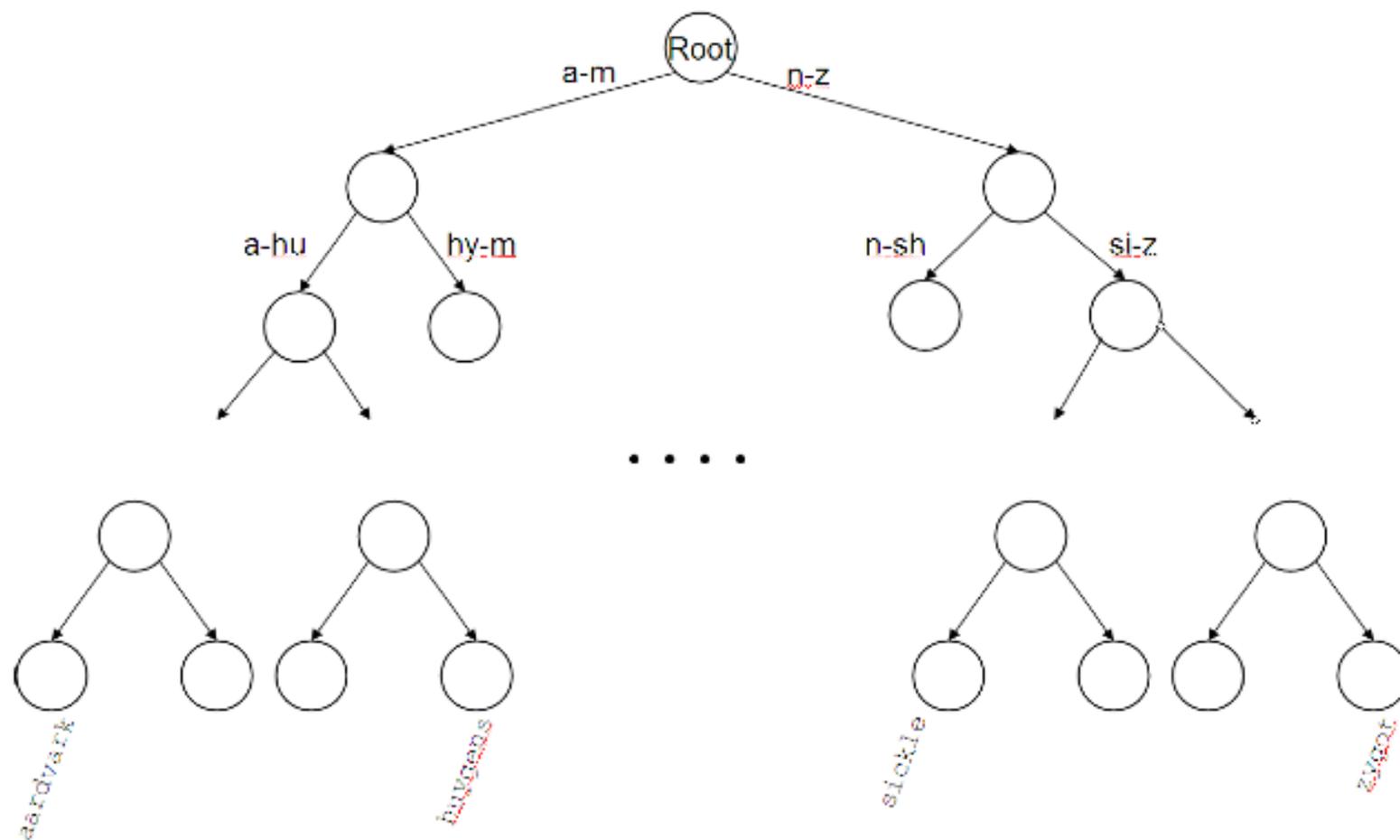
Hashes

- 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: Lookup in a hash is faster than lookup in a tree.
 - Lookup time is constant.
- Cons
 - no way to find minor variants (*resume* vs. *résumé*)
 - no prefix search (all terms starting with *automat*)
 - need to rehash everything periodically if vocabulary keeps growing

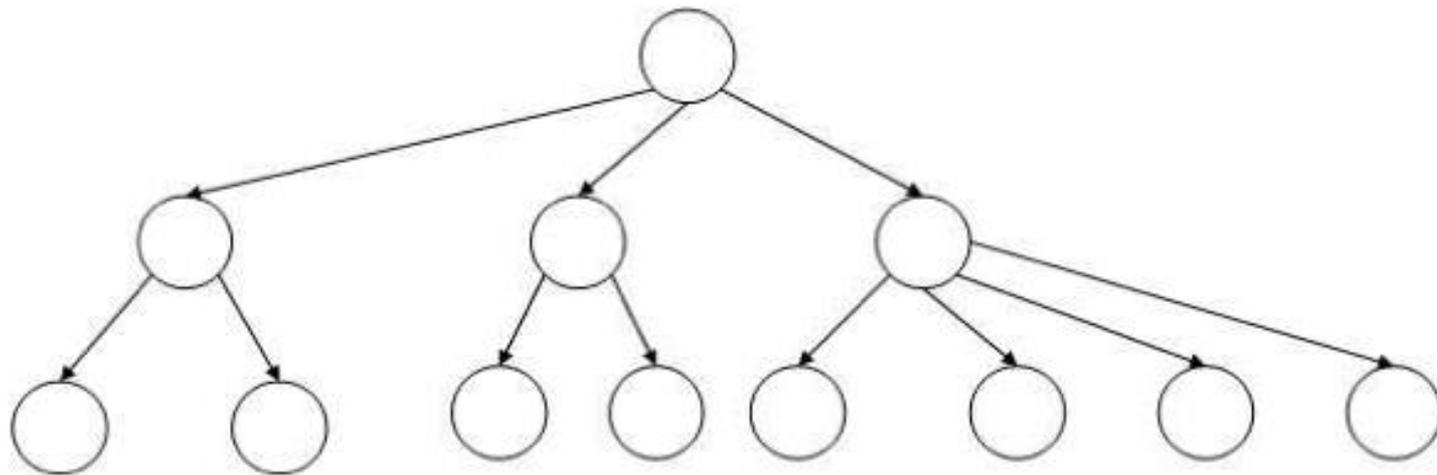
Trees

- Trees solve the prefix problem (find all terms starting with *automat*).
- Simplest tree: binary tree
- 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 binary trees 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., $[2, 4]$.

Binary tree



B-tree



Tolerant Retrieval

- **Tolerant retrieval**: What to do if there is no exact match between query term and document term
- Wildcard queries
- Spelling correction

Wildcard queries

- hyd^* : find all docs containing any term beginning with *mon*
- Easy with B-tree dictionary: retrieve all terms t in the range: $hyd \leq t < hye$
- $*bad$: find all docs containing any term ending with *bad*
 - Maintain an additional tree for terms *backwards*
 - Then retrieve all terms t in the range: $dab \leq t < dac$
- Result: A set of terms that are matches for wildcard query
- Then retrieve documents that contain any of these terms

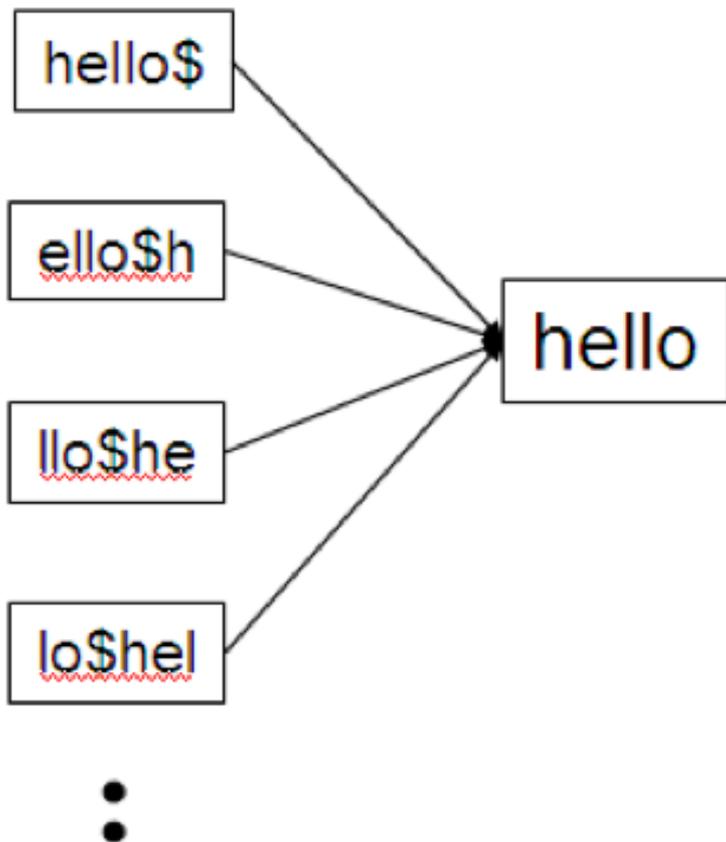
How to handle * in the middle of a term

- Example: hyd*bad
- We could look up hyd* and *bad in the B-tree and intersect the two term sets.
- Expensive
- Alternative: [permuterm](#) index
- Basic idea: Rotate every wildcard query, so that the * occurs at the end.
- Store each of these rotations in the dictionary, say, in a B-tree

Permuterm index

- For term HELLO: add *hello\$, ello\$h, llo\$he, lo\$hel, and o\$hell* to the B-tree where \$ is a special symbol

Permuterm \rightarrow term mapping



k -gram indexes

- More space-efficient than permuterm index
- Enumerate all character k -grams (sequence of k characters) occurring in a term
- 2-grams are called **bigrams**.
- Example :
- $re*ve \rightarrow \$re \text{ AND } ve\$$
- Look up a 2-gram list. Yields a list of matching terms like relive, remove, retrieve, revive. Look up these terms in the inverted index.

Processing wildcard queries in the term-document index

- Problem 1: we must potentially execute a large number of Boolean queries.
- Most straightforward semantics: Conjunction of disjunctions
- For [gen* universit*]: geneva university OR geneva université OR genève university OR genève université OR general universities OR . . .
- Very expensive
- Problem 2: Users hate to type.
- If abbreviated queries like [pyth* theo*] for [pythagoras' theorem] are allowed, users will use them a lot.
- This would significantly increase the cost of answering queries.
- Somewhat alleviated by Google Suggest

Spelling correction

- Two principal uses
 - Correcting documents being indexed
 - Correcting user queries
- Two different methods for spelling correction
- **Isolated word** spelling correction
 - Check each word on its own for misspelling
 - Will not catch typos resulting in correctly spelled words, e.g., *an asteroid that fell **form** the sky*
- **Context-sensitive** spelling correction
 - Look at surrounding words
 - Can correct *form/from* error above

Distance between misspelled word and “correct” word

- We will study several alternatives.
- Edit distance and Levenshtein distance
- Weighted edit distance
- k -gram overlap

Edit distance

- The edit distance between string s_1 and string s_2 is the minimum number of basic operations that convert s_1 to s_2 .
- Levenshtein distance: The admissible basic operations are insert, delete, and replace
- Levenshtein distance *dog-do*: 1
- Levenshtein distance *cat-cart*: 1
- Levenshtein distance *cat-cut*: 1
- Levenshtein distance *cat-act*: 2
- Damerau-Levenshtein distance *cat-act*: 1
- Damerau-Levenshtein includes transposition as a fourth possible operation.

Levenshtein distance: Algorithm

LEVENSHTEINDISTANCE(s_1, s_2)

```
1  for  $i \leftarrow 0$  to  $|s_1|$ 
2  do  $m[i, 0] = i$ 
3  for  $j \leftarrow 0$  to  $|s_2|$ 
4  do  $m[0, j] = j$ 
5  for  $i \leftarrow 1$  to  $|s_1|$ 
6  do for  $j \leftarrow 1$  to  $|s_2|$ 
7     do if  $s_1[i] = s_2[j]$ 
8         then  $m[i, j] = \min\{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]\}$ 
9         else  $m[i, j] = \min\{m[i-1, j]+1, m[i, j-1]+1, m[i-1, j-1]+1\}$ 
10 return  $m[|s_1|, |s_2|]$ 
```

Operations: insert (cost 1), delete (cost 1), replace (cost 1), copy (cost 0)

Weighted edit distance

- As above, but weight of an operation depends on the characters involved.
- Meant to capture keyboard errors, e.g., m more likely to be mistyped as n than as q .
- Therefore, replacing m by n is a smaller edit distance than by q .
- We now require a weight matrix as input.
- Modify dynamic programming to handle weights

Using edit distance for spelling correction

- Given query, first enumerate all character sequences within a preset (possibly weighted) edit distance
- Intersect this set with our list of “correct” words
- Then suggest terms in the intersection to the user.

Context-sensitive spelling correction

- Our example was: *an asteroid that fell **form** the sky*
- How can we correct *form* here?
- One idea: **hit-based** spelling correction
 - Retrieve “correct” terms close to each query term
 - *for flew form munich: flea for flew, from for form, munch for*
 - *munich*
 - Now try all possible resulting phrases as queries with one word “fixed” at a time
 - Try query “*flea form munich*”
 - Try query “*flew from munich*”
 - Try query “*flew form munch*”
 - The correct query “*flew from munich*” has the most hits.
- Suppose we have 7 alternatives for *flew*, 20 for *form* and 3 for *munich*, how many “corrected” phrases will we enumerate?

General issues in spelling correction

- User interface
 - automatic vs. suggested correction
 - *Did you mean* only works for one suggestion.
 - What about multiple possible corrections?
 - Tradeoff: simple vs. powerful UI
- Cost
 - Spelling correction is potentially expensive.
 - Avoid running on every query?
 - Maybe just on queries that match few documents.
 - Guess: Spelling correction of major search engines is efficient enough to be run on every query.

Soundex

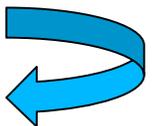
- Soundex is the basis for finding **phonetic** (as opposed to orthographic) alternatives.
- Example: *chebyshev / tchebyscheff*
- Algorithm:
 - Turn every token to be indexed into a 4-character reduced form
 - Do the same with query terms
 - Build and search an index on the reduced forms

Outline

- Information Retrieval
 - Definition and Motivation
 - **Inverted Index**
 - A simple Boolean information retrieval system
 - Term vocabulary and postings list
 - Dictionaries and tolerant retrieval
 - **Index construction and storage**
 - Scoring, term weighting and vector-space model (exercise)
 - Ranking and Cosine Similarity
- **Information Extraction**
 - Entity Extraction & Relation Extraction

Index Construction & Storage

- Two index construction algorithms: **BSBI** (simple) and **SPIMI** (more realistic)
- **Distributed** index construction: MapReduce
- **Dynamic** index construction: how to keep the index up-to-date as the collection changes



RCV1 collection

- Shakespeare's collected works are not large enough for demonstrating many of the points in this course.
- As an example for applying scalable index construction algorithms, we will use the [Reuters RCV1](#) collection.
- English newswire articles sent over the wire in 1995 and 1996 (one year).

A Reuters RCV1 document



You are here: [Home](#) > [News](#) > [Science](#) > [Article](#)

Go to a Section: [U.S.](#) [International](#) [Business](#) [Markets](#) [Politics](#) [Entertainment](#) [Technology](#) [Sports](#) [Oddly En](#)

Extreme conditions create rare Antarctic clouds

Tue Aug 1, 2006 3:20am ET

[Email This Article](#) | [Print This Article](#) | [Reprin](#)

[\[-\] Text \[-\]](#)



SYDNEY (Reuters) - Rare, mother-of-pearl colored clouds caused by extreme weather conditions above Antarctica are a possible indication of global warming, Australian scientists said on Tuesday.

Known as nacreous clouds, the spectacular formations showing delicate wisps of colors were photographed in the sky over an Australian

Reuters RCV1 statistics

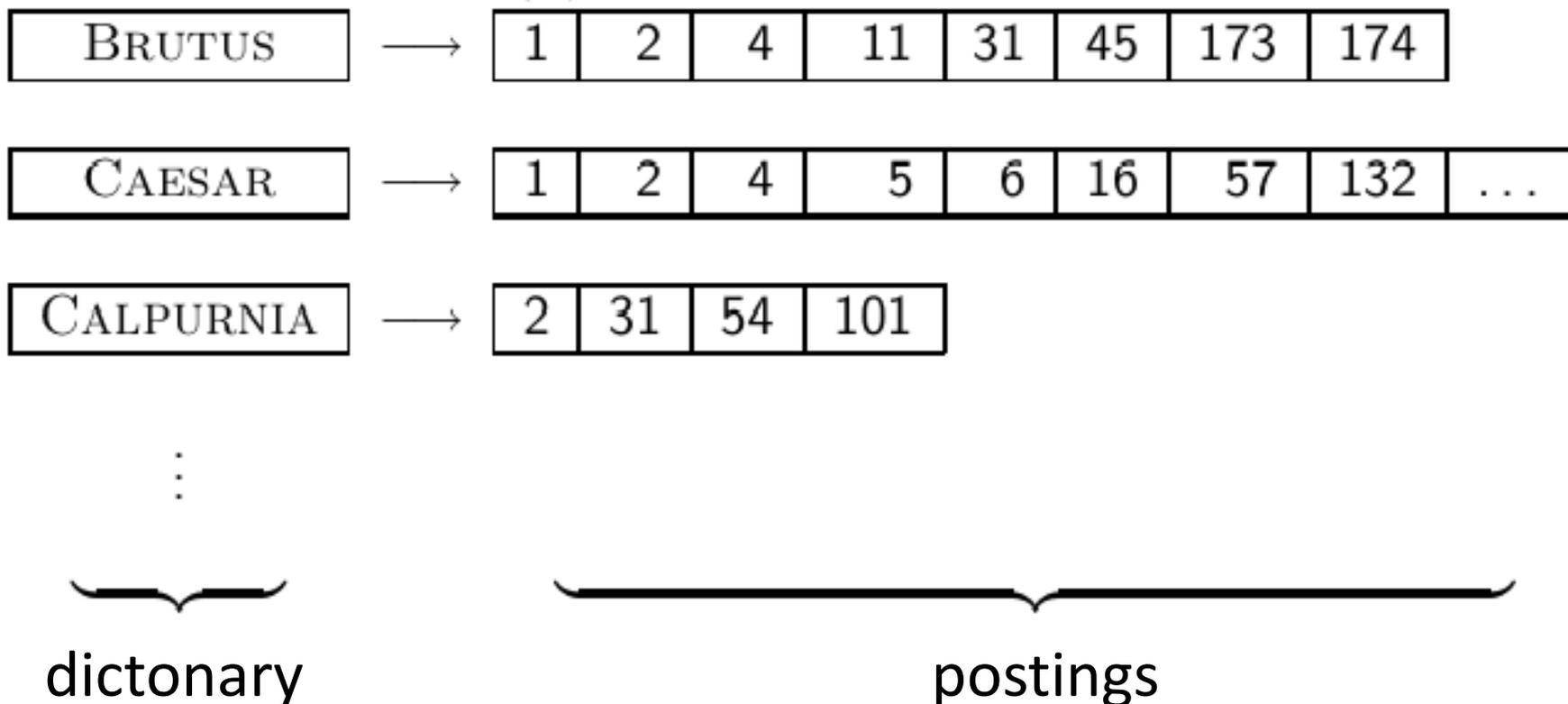
N	documents	800,000
L	tokens per document	200
M	terms (= word types)	400,000
	bytes per token (incl. spaces/punct.)	6
	bytes per token (without spaces/punct.)	4.5
	bytes per term (= word type)	7.5
T	non-positional postings	100,000,000

Exercise: Average frequency of a term (how many tokens)? 4.5

bytes per word token vs. 7.5 bytes per word type: why the difference? How many positional postings?

BSBI Algorithm

Goal: construct the inverted Index



Sort-based index construction

- As we build index, we parse docs one at a time.
- The final postings for any term are incomplete until the end.
- Can we keep all postings in memory and then do the sort in-memory at the end?
- No, not for large collections
- At 10–12 bytes per postings entry, we need a lot of space for large collections.
- $T = 100,000,000$ in the case of RCV1: we can do this in memory on a typical machine in 2010.
- But in-memory index construction does not scale for large collections.
- Thus: We need to store intermediate results on disk.

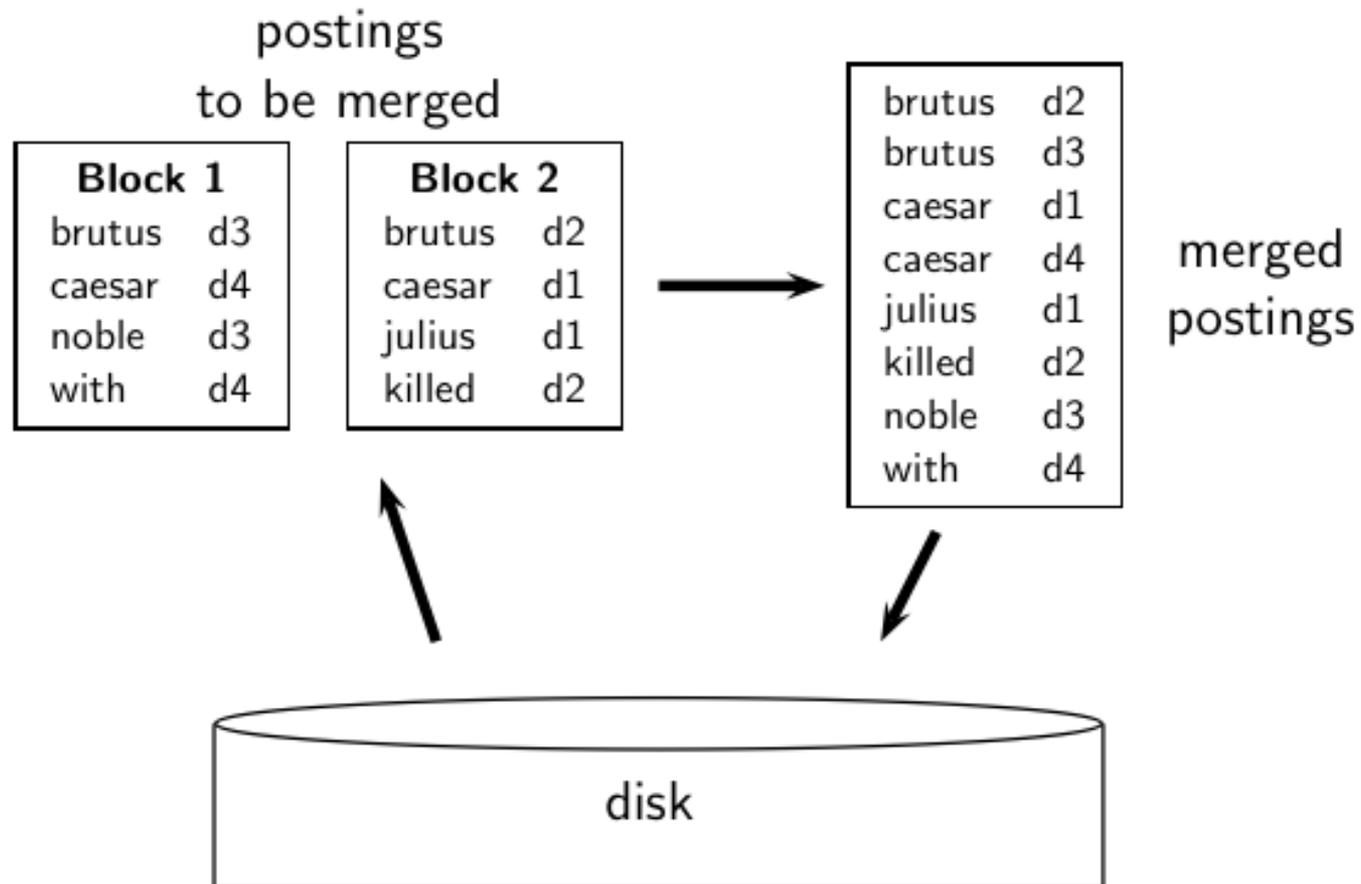
Same algorithm for disk?

- Can we use the same index construction algorithm for larger collections, but by using disk instead of memory?
- No: Sorting $T = 100,000,000$ records on disk is too slow – too many disk seeks.
- We need an **external** sorting algorithm.

“External” sorting algorithm (using few disk seeks)

- We must sort $T = 100,000,000$ non-positional postings.
 - Each posting has size 12 bytes (4+4+4: termID, docID, document frequency).
- Define a **block** to consist of 10,000,000 such postings
 - We can easily fit that many postings into memory.
 - We will have 10 such blocks for RCV1.
- Basic idea of algorithm:
 - For each block: (i) accumulate postings, (ii) sort in memory, (iii) write to disk
 - Then merge the blocks into one long sorted order.

Merging two blocks



Blocked Sort-Based Indexing (BSBI)

BSBINDEXCONSTRUCTION()

```
1   $n \leftarrow 0$ 
2  while (all documents have not been processed)
3  do  $n \leftarrow n + 1$ 
4      $block \leftarrow \text{PARSENEXTBLOCK}()$ 
5     BSBI-INVERT( $block$ )
6     WRITEBLOCKTODISK( $block, f_n$ )
7  MERGEBLOCKS( $f_1, \dots, f_n; f_{\text{merged}}$ )
```

- Key decision: What is the size of one block?

Problem with sort-based algorithm

- Our assumption was: we can keep the dictionary in memory.
- We need the dictionary (which grows dynamically) in order to implement a term to termID mapping.
- Actually, we could work with term,docID postings instead of termID,docID postings . . .
- . . . but then intermediate files become very large. (We would end up with a scalable, but very slow index construction method.)

Single-pass in-memory indexing(SPIMI)

- Abbreviation: SPIMI
- Key idea 1: Generate separate dictionaries for each block – no need to maintain term-termID mapping across blocks.
- Key idea 2: Don't sort. Accumulate postings in postings lists as they occur.
- With these two ideas we can generate a complete inverted index for each block.
- These separate indexes can then be merged into one big index.

SPIMI-Invert

```
SPIMI-INVERT(token_stream)
1  output_file ← NEWFILE()
2  dictionary ← NEWHASH()
3  while (free memory available)
4  do token ← next(token_stream)
5     if term(token) ∉ dictionary
6     then postings_list ← ADDTODICTIONARY(dictionary, term(token))
7     else postings_list ← GETPOSTINGSLIST(dictionary, term(token))
8     if full(postings_list)
9     then postings_list ← DOUBLEPOSTINGSLIST(dictionary, term(token))
10    ADDTODICTIONARY(dictionary, postings_list, docID(token))
11  sorted_terms ← SORTTERMS(dictionary)
12  WRITEBLOCKTODISK(sorted_terms, dictionary, output_file)
13  return output_file
```

Merging of blocks is analogous to BSBI.

Distributed indexing

- For web-scale indexing (don't try this at home!): must use a distributed computer cluster
- Individual machines are fault-prone.
 - Can unpredictably slow down or fail.
- How do we exploit such a pool of machines?

Distributed indexing

- Maintain a **master** machine directing the indexing job – considered “safe”
- Break up indexing into sets of parallel tasks
- Master machine assigns each task to an idle machine from a pool.

MapReduce

- The index construction algorithm can be implemented with MapReduce.
- MapReduce is a robust and conceptually simple framework for distributed computing . . .
- . . .without having to write code for the distribution part.
- The Google indexing system (ca. 2002) consisted of a number of phases, each implemented in MapReduce.
- Index construction was just one phase.
- Another phase: transform term-partitioned into document-partitioned index.

Dynamic indexing

- Up to now, we have assumed that collections are **static**.
- They rarely are: Documents are inserted, deleted and modified.
- This means that the dictionary and postings lists have to be **dynamically** modified.

Dynamic indexing: Simplest approach

- Maintain big **main index on disk**
- New docs go into **small auxiliary index in memory**.
- Search across both, merge results
- Periodically, merge auxiliary index into big index
- Deletions:
 - Invalidation bit-vector for deleted docs
 - Filter docs returned by index using this bit-vector

Issue with auxiliary and main index

- Frequent merges
- Poor search performance during index merge
- Actually:
 - Merging of the auxiliary index into the main index is not that costly if we keep a separate file for each postings list.
 - Merge is the same as a simple append.
 - But then we would need a lot of files – inefficient.
- Assumption for the rest of the lecture: The index is one big file.
- In reality: Use a scheme somewhere in between (e.g., split very large postings lists into several files, collect small postings lists in one file etc.)

Logarithmic merge

- Logarithmic merging amortizes the cost of merging indexes over time.
 - → Users see smaller effect on response times.
- Maintain a series of indexes, each twice as large as the previous one.
- Keep smallest (Z_0) in memory
- Larger ones (I_0, I_1, \dots) on disk
- If Z_0 gets too big ($> n$), write to disk as I_0
- ... or merge with I_0 (if I_0 already exists) and write merger to I_1 etc.

Why compression

- An index compression ratio of 1:4 cuts cost of storage by 75%
- Increased use of cache
- Faster transfer of data from disk to memory

Outline

- Information Retrieval
 - Definition and Motivation
 - **Inverted Index**
 - A simple Boolean information retrieval system
 - Term vocabulary and postings list
 - Dictionaries and tolerant retrieval
 - Index construction and storage
 - Scoring, term weighting and vector-space model (exercise)
 - Ranking and Cosine Similarity
- **Information Extraction**
 - Entity Extraction
 - Relation Extraction

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

- 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”.

Take 1: Jaccard coefficient

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

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

$(A \neq \emptyset \text{ or } B \neq \emptyset)$

- $\text{JACCARD}(A, A) = 1$
- $\text{JACCARD}(A, B) = 0$ 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”
 - $JACCARD(q, d) = 1/6$

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.

See for yourself

- 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”

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.

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

Binary incidence 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 $\in \mathbb{N}^{|V|}$.

Not 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.

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.

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:

$$idf_t = \log_{10} \frac{N}{df_t}$$

(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: $idf_t = \log_{10} \frac{1,000,000}{df_t}$

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
- Note: the “-” in tf-idf is a hyphen, not a minus sign!
- 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)

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

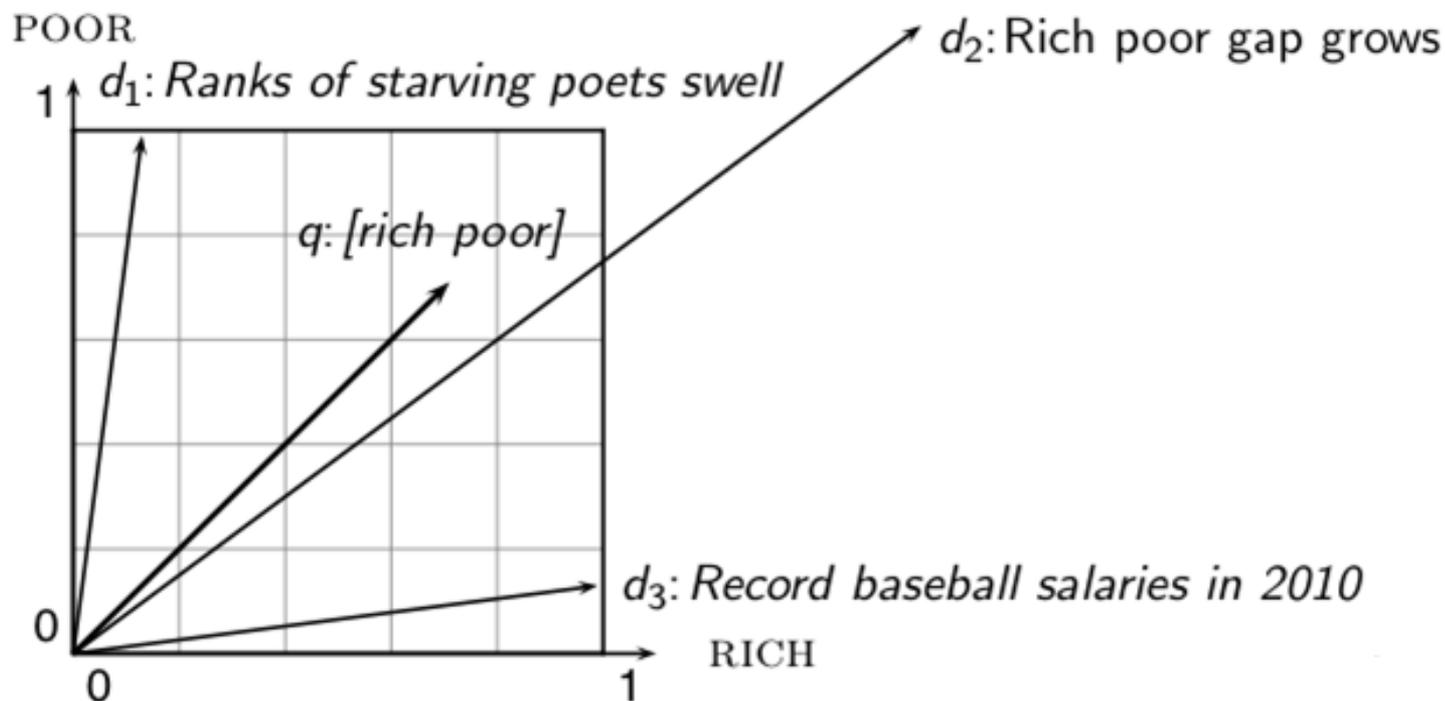
Documents as vectors

- 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

Euclidean distance : a bad idea?



The Euclidean distance of \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?

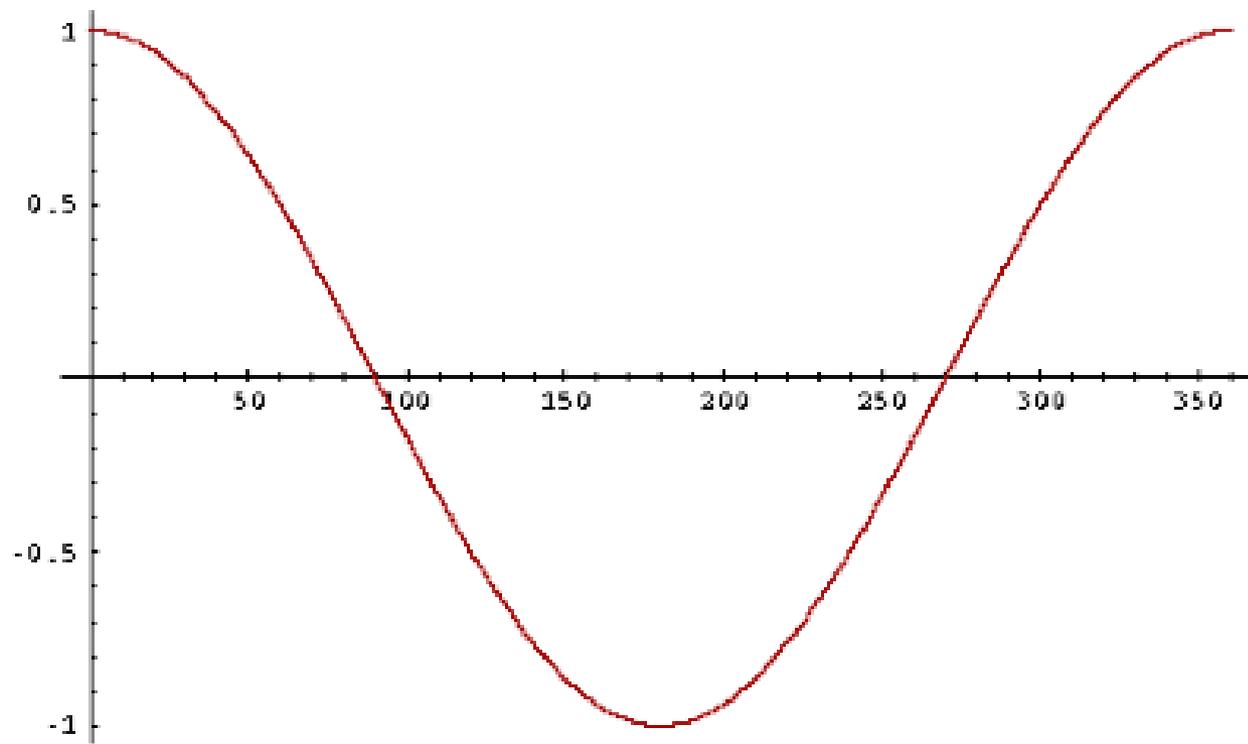
Use angle instead of distance

- 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} .

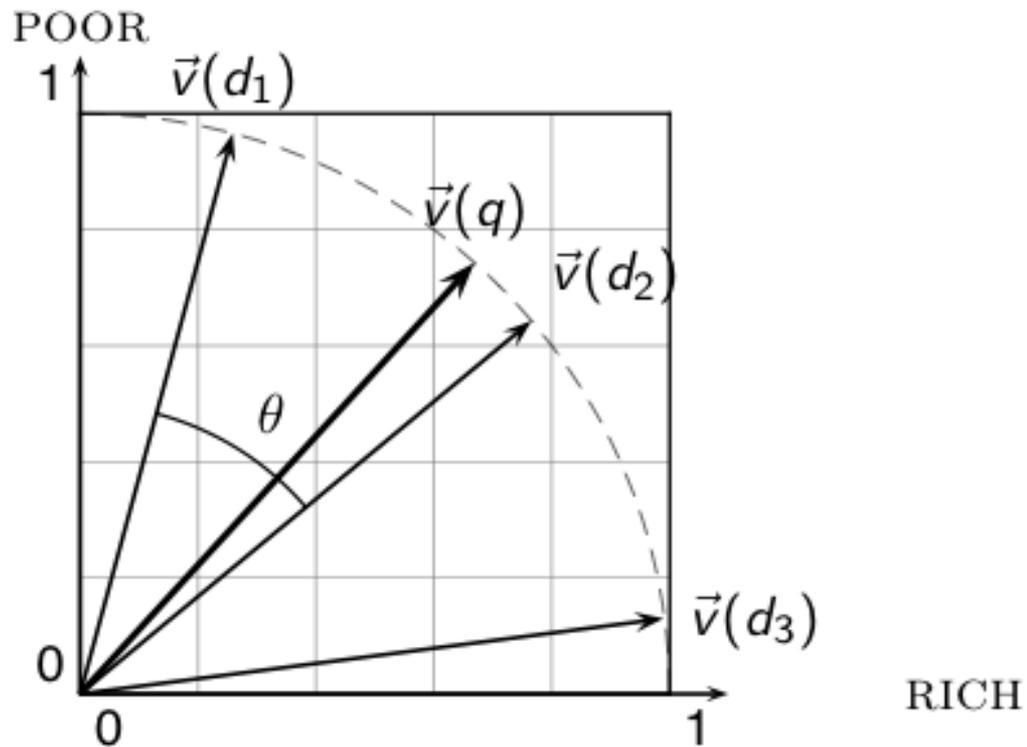
Cosine for normalized vectors

- 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)

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

How similar are these novels?

SaS: Sense and Sensibility

PaP: Pride and Prejudice

WH: Wuthering Heights

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 $\cos(\text{SaS}, \text{PaP}) > \cos(\text{SAS}, \text{WH})$?

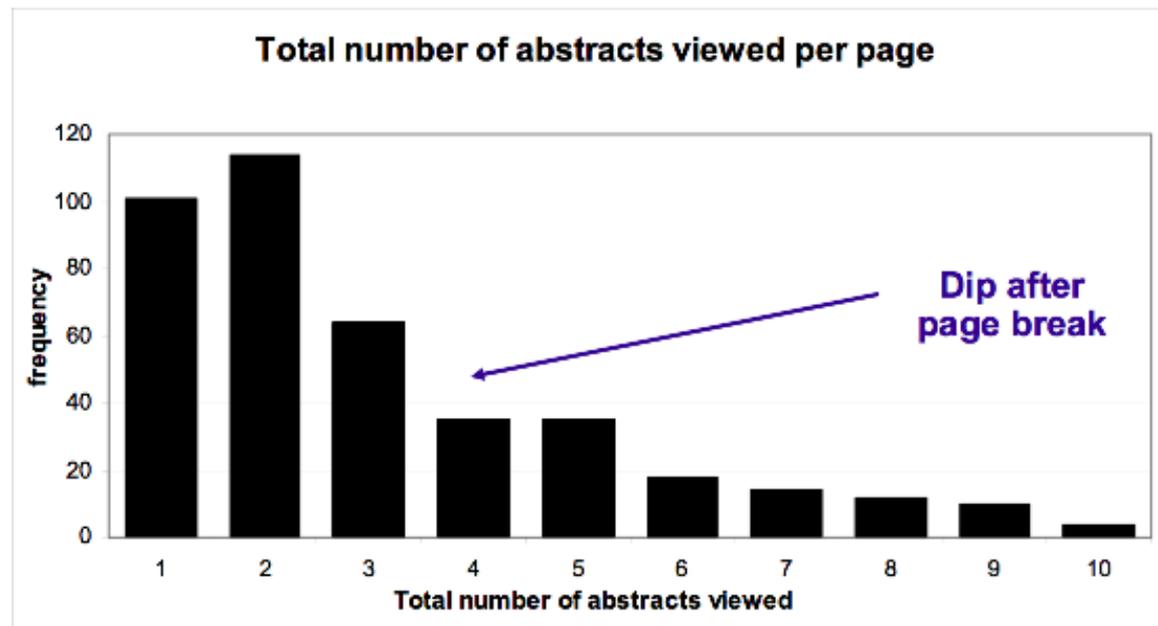
Summary: Ranked retrieval in the vector space model

- 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

Outline

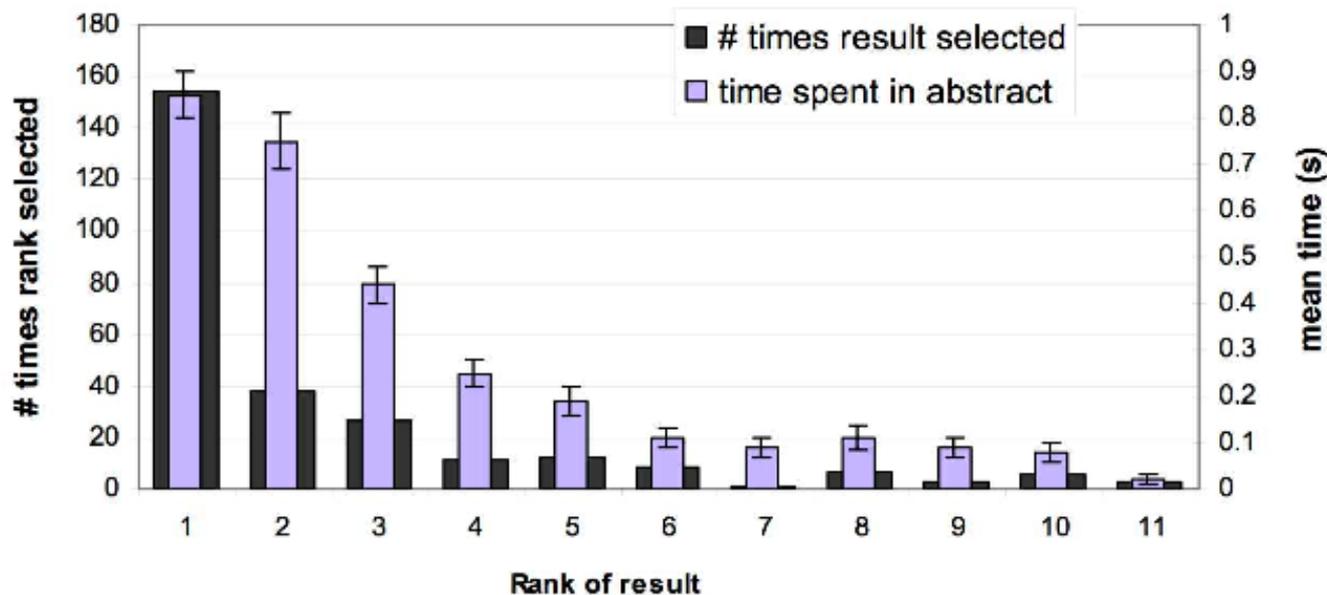
- Information Retrieval
 - Definition and Motivation
 - **Inverted Index**
 - A simple Boolean information retrieval system
 - Term vocabulary and postings list
 - Dictionaries and tolerant retrieval
 - Index construction and storage
 - Scoring, term weighting and vector-space model
 - **Ranking and Cosine Similarity**
- Information Extraction
 - Entity Extraction & Relation Extraction

How many links do users view?



Mean: 3.07 Median/Mode: 2.00

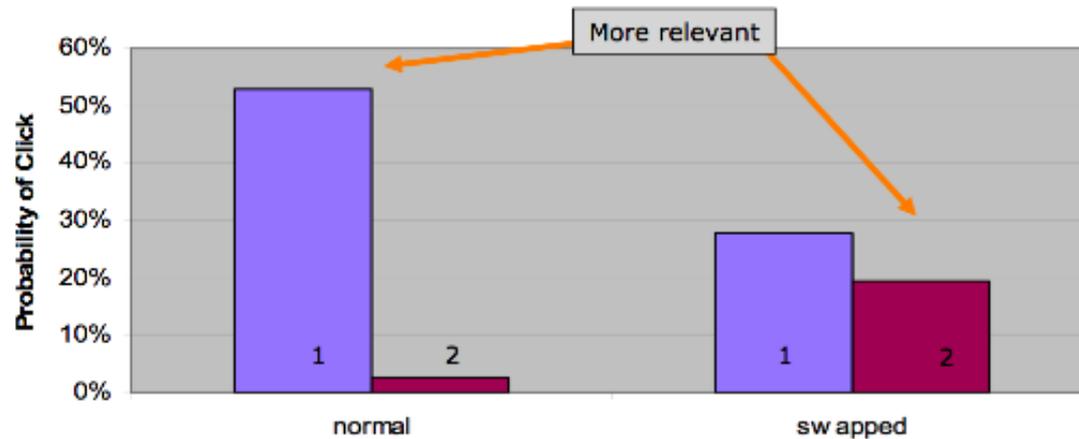
Looking vs. Clicking



- 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.



Ranking Implementation:

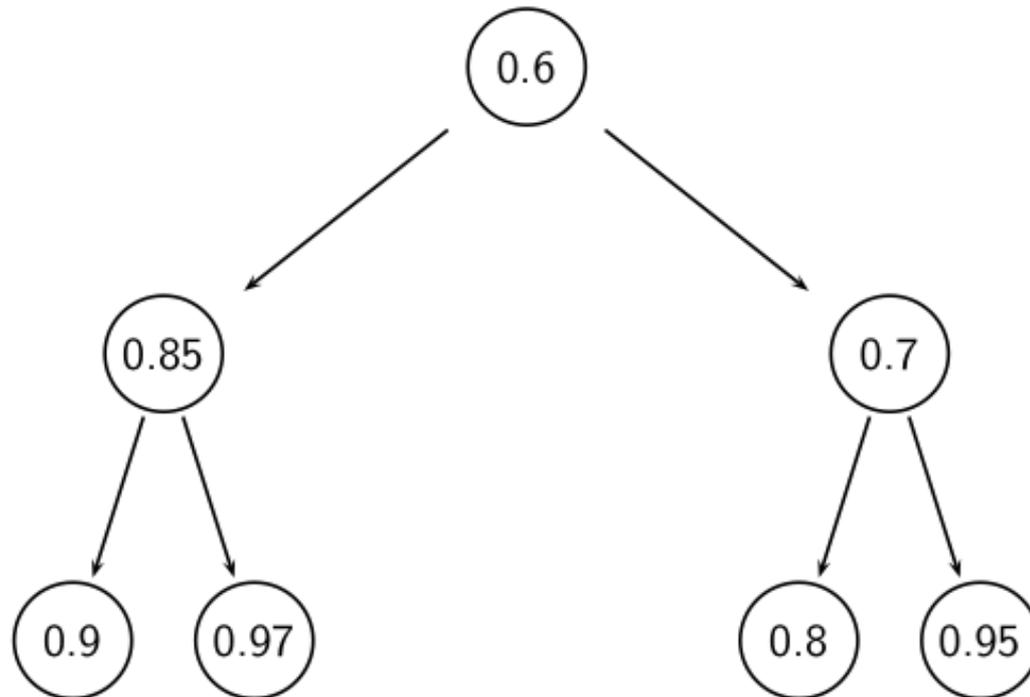
How do we compute the top k in ranking?

- In many applications, we don't need a complete ranking.
- We just need the top k for a small k (e.g., $k = 100$).
- If we don't need a complete ranking, is there an efficient way of computing just the top k ?
- Naive:
 - Compute scores for all N documents
 - Sort
 - Return the top k
- What's bad about this?
- Alternative?

Use min heap for selecting top k out of N

- Use a binary min heap
- A binary min heap is a binary tree in which each node's value is less than the values of its children.
- Takes $O(N \log k)$ operations to construct (where N is the number of documents) . . .
- . . . then read off k winners in $O(k \log k)$ steps

Binary min heap



Even more efficient computation of top k ?

- Ranking has time complexity $O(N)$ where N is the number of documents.
- Optimizations reduce the constant factor, but they are still $O(N)$, $N > 10^{10}$
- Are there sublinear algorithms?
- What we're doing in effect: solving the k -nearest neighbor (kNN) problem for the query vector (= query point).
- There are no general solutions to this problem that are sublinear.

More efficient computation of top k: Heuristics

- Idea 1: Reorder postings lists
 - Instead of ordering according to docID . . .
 - . . . order according to some measure of “expected relevance”.
- Idea 2: Heuristics to prune the search space
 - Not guaranteed to be correct . . .
 - . . . but fails rarely.
 - In practice, close to constant time.
 - For this, we’ll need the concepts of document-at-a-time processing and term-at-a-time processing.

Non-docID ordering of postings lists

- So far: postings lists have been ordered according to docID.
- Alternative: a query-independent measure of “goodness” of a page
- Example: **PageRank** $g(d)$ of page d , a measure of how many “good” pages hyperlink to d (chapter 21)
- Order documents in postings lists according to PageRank:
 $g(d_1) > g(d_2) > g(d_3) > \dots$
- Define composite score of a document:
$$\text{net-score}(q, d) = g(d) + \cos(q, d)$$
- This scheme supports early termination: We do not have to process postings lists in their entirety to find top k .

Document-at-a-time processing

- Both docID-ordering and PageRank-ordering impose a consistent ordering on documents in postings lists.
- Computing cosines in this scheme is **document-at-a-time**.
- We complete computation of the query-document similarity score of document d_i before starting to compute the query-document similarity score of d_{i+1} .
- Alternative: term-at-a-time processing

Weight-sorted postings lists

- Idea: don't process postings that contribute little to final score
- Order documents in postings list according to **weight**
- Simplest case: normalized tf-idf weight (rarely done: hard to compress)
- Documents in the top k are likely to occur early in these ordered lists.
- → Early termination while processing postings lists is unlikely to change the top k .
- But:
 - We no longer have a consistent ordering of documents in postings lists.
 - We no longer can employ document-at-a-time processing.

Term-at-a-time processing

- Simplest case: completely process the postings list of the first query term
- Create an accumulator for each docID you encounter
- Then completely process the postings list of the second query term
- . . . and so forth

Term-at-a-time processing

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  $Scores[]$ 
```

The elements of the array "Scores" are called **accumulators**.

Computing cosine scores

- For the web (20 billion documents), an array of accumulators A in memory is infeasible.
- Thus: Only create accumulators for docs occurring in postings lists
- This is equivalent to: Do not create accumulators for docs with zero scores (i.e., docs that do not contain any of the query terms)

Accumulators: Example

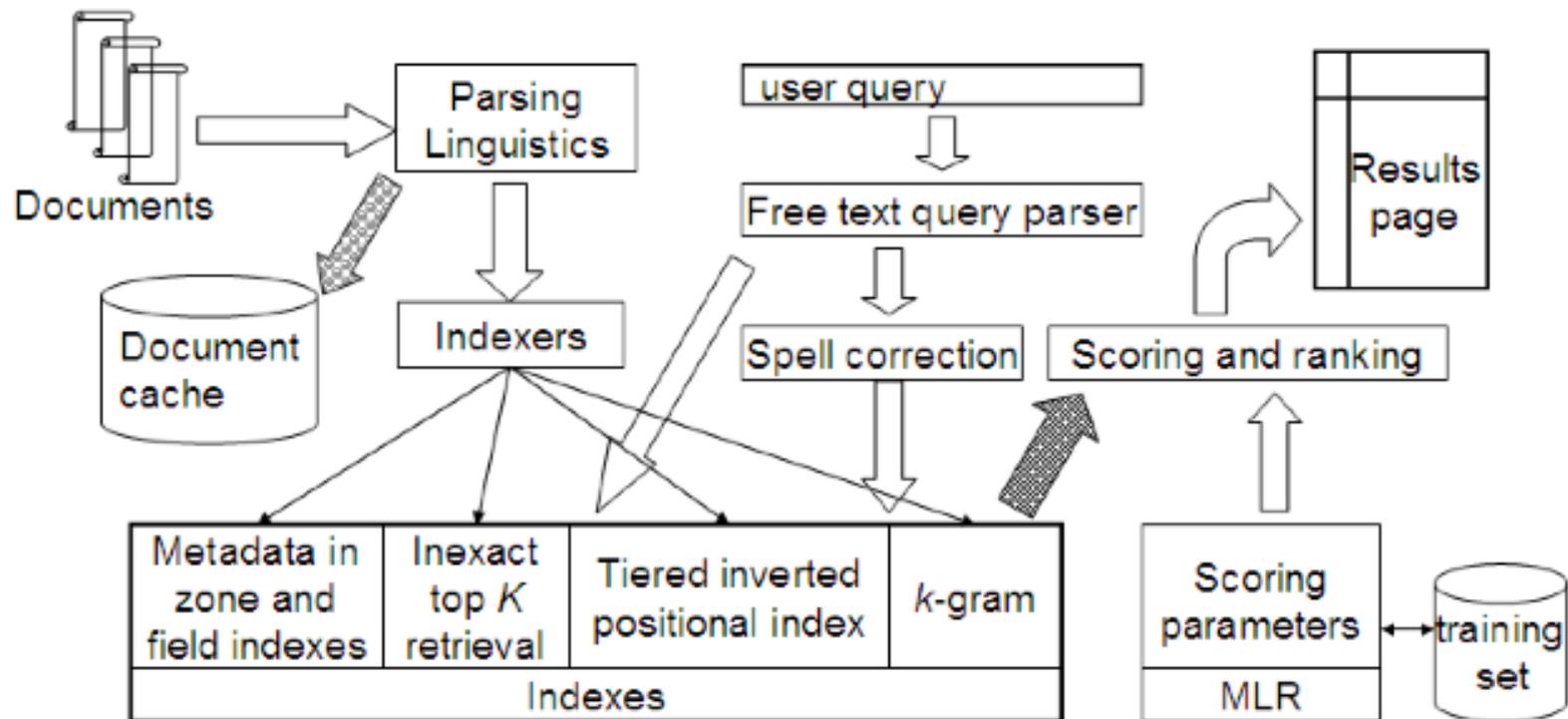
BRUTUS → 1 ,2 | 7 ,3 | 83 ,1 | 87 ,2 | ...

CAESAR → 1 ,1 | 5 ,1 | 13 ,1 | 17 ,1 | ...

CALPURNIA → 7 ,1 | 8 ,2 | 40 ,1 | 97 ,3

- For query: [Brutus Caesar]:
- Only need accumulators for 1, 5, 7, 13, 17, 83, 87
- Don't need accumulators for 8, 40, 85

Complete search system



Outline

- Information Retrieval
 - Definition and Motivation
 - **Inverted Index**
 - A simple Boolean information retrieval system
 - Term vocabulary and postings list
 - Dictionaries and tolerant retrieval
 - Index construction and storage
 - Scoring, term weighting and vector-space model
 - Ranking and Cosine Similarity
- Information Extraction
 - Entity Extraction & Relation Extraction

A problem

baker job opening - Google Search - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://www.google.com/search?ie=UTF-8&oe=UTF-8&sourceid=gd&q=baker+job+op

Google Web Images Video^{New!} News Maps Desktop Moma more »

baker job opening Search Advanced Search Preferences

Web Results 1 - 10 of about 6,150,000 for **baker job opening**. (0.09 seconds)

Mt Baker School District
You may also call 360-383-2075 for a voice message concerning our **job** listings. Our district applications may be downloaded from each **job** category site. ...
www.mtbaker.wednet.edu/jobs/ - 3k - [Cached](#) - [Similar pages](#) - [Filter](#)

CGI : Job Opening
Job Seekers, Faculty & Other Researchers, Students, Journalists, Policy Makers ... Baker Institute for Animal Health, College of Veterinary Medicine ...
www.genomics.cornell.edu/jobs/view_job.cfm?id=47 - 15k - [Cached](#) - [Similar pages](#) - [Filter](#)

Baker Hostetler - Staff Job Openings
law business employee benefits employment intellectual property international legislative regulatory litigation private wealth real estate tax automotive ...
www.bakerlaw.com/Careers.aspx?Abs_WVP_ID=26a8ff33-0471-4c5e-b5b7-6abdcfce0326-19k - [Cached](#) - [Similar pages](#) - [Filter](#)

Baker & McKenzie || Careers || Current Openings ||
We are always looking for talented, internationally minded people interested in building their careers with a truly global law firm.
www.bakernet.com/BakerNet/Careers/Current+Openings/ - 64k - [Cached](#) - [Similar pages](#) - [Filter](#)

Current Job Opening Search
Click the search button to see all **job openings**. ... Apprentice Baker, Architect - Production, Architectural Drafting Intern, Architectural Project Leader ...
hyveenet.hy-vee.com/applynow/ - 75k - [Cached](#) - [Similar pages](#) - [Filter](#)

Law Enforcement Job Submission
Advertise Your **Job Openings** ... -Mia Baker, Human Resources Officer, Amtrak ... You can announce your **job opening** to thousands of potential applicants at a ...
www.policeemployment.com/joblisting/ - 10k - [Cached](#) - [Similar pages](#) - [Filter](#)

<http://www.bakernet.com/BakerNet/Careers/Current+Openings/>

Sponsored Links

Baker Job
Search Thousands of Job Listings for Opportunities in Your Area
Jobs.AOL.com

Find Bakers Jobs
CareerBuilder® Has More Jobs In Hospitality Than Any Other Site!
CareerBuilder.com/Baker_Jobs

i Hire Chefs
Chef Jobs -Find a Culinary Arts Job Nationwide Employment Opportunities
www.iHireChefs.com

Mt. Baker, the school district

Baker Hostetler, the company

Baker, a job opening

FlipDog.com
Fetch Your Next Job Here™

Home Find Jobs Your Account Resource Center

Return to Results | Modify Search | New Search

The University Alliance
A BISK EDUCATION NETWORK
Degrees Online

Learn While You Earn
MBA, BA, AA Degrees
Online & Project Mgt.

Click here to e-mail your resume to 1000's
of Head Hunters with
ResumeZapper.com

how to easily
DOUBLE
your chances
when applying
FOR JOBS!

Breakthrough ebook
shows why most people
are WRONG about how
to apply for jobs.

1 - 25 of 47 jobs shown below 1 2 Next >

Search these results for: [Search tips](#) Show Jobs Posted:

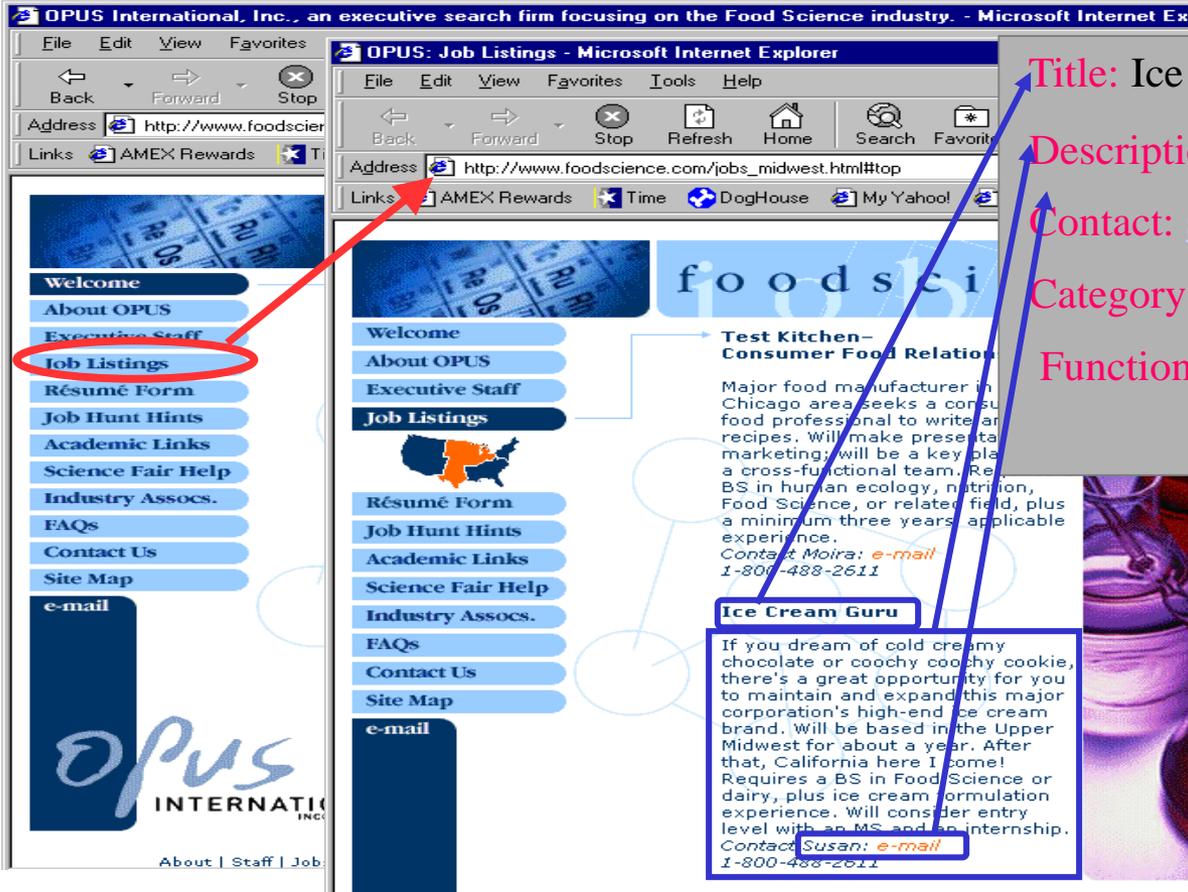
View: [Brief](#) | [Detailed](#)

Web Jobs: FlipDog technology has found these jobs on thousands of employer Web sites.

Food Pantry Workers at Lutheran Social Services	October 11, 2002	Archbold, OH
Cooks at Lutheran Social Services	October 11, 2002	Archbold, OH
Bakers Assistants at Fine Catering by Russell Morin	October 11, 2002	Attleboro, MA
Baker's Helper at Bird-in-Hand	October 11, 2002	United States
Assistant Baker at Gourmet To Go	October 11, 2002	Maryland Heights, MO
	October 10, 2002	Beaverton, OR
	October 10, 2002	Alta, UT
	October 10, 2002	Huntsville, UT
School District	October 10, 2002	Garden Grove, CA
	October 10, 2002	Houma, LA
	October 10, 2002	Nisswa, MN
	October 10, 2002	Big Sky, MT
	October 08, 2002	Willowbrook, IL
Cake Decorator/Baker at Mandalay Bay Hotel and Casino	October 08, 2002	Las Vegas, NV
Shift Supervisors at Brueggers Bagels	October 08, 2002	Minneapolis, MN

Job Openings:
Category = Food Services
Keyword = Baker
Location = Continental U.S.

Extracting Job Openings from the Web



Title: Ice Cream Guru

Description: If you dream of cold creamy...

Contact: susan@foodscience.com

Category: Travel/Hospitality

Function: Food Services



Ice Cream Guru
If you dream of cold creamy chocolate or gooey gooey cookie, there's a great opportunity for you to maintain and expand this major corporation's high-end ice cream brand. Will be based in the Upper Midwest for about a year. After that, California here I come! Requires a BS in Food Science or dairy, plus ice cream formulation experience. Will consider entry level with an MS and an internship.
Contact Susan: e-mail 1-800-488-2611

Definition of IE

Information Extraction (IE) is the process of extracting structured information (e.g., database tables) from unstructured machine-readable documents (e.g., Web documents).

Elvis Presley was a famous rock singer.

...

Mary once remarked that the only attractive thing about the painter Elvis Hunter was his first name.

**Information
Extraction**

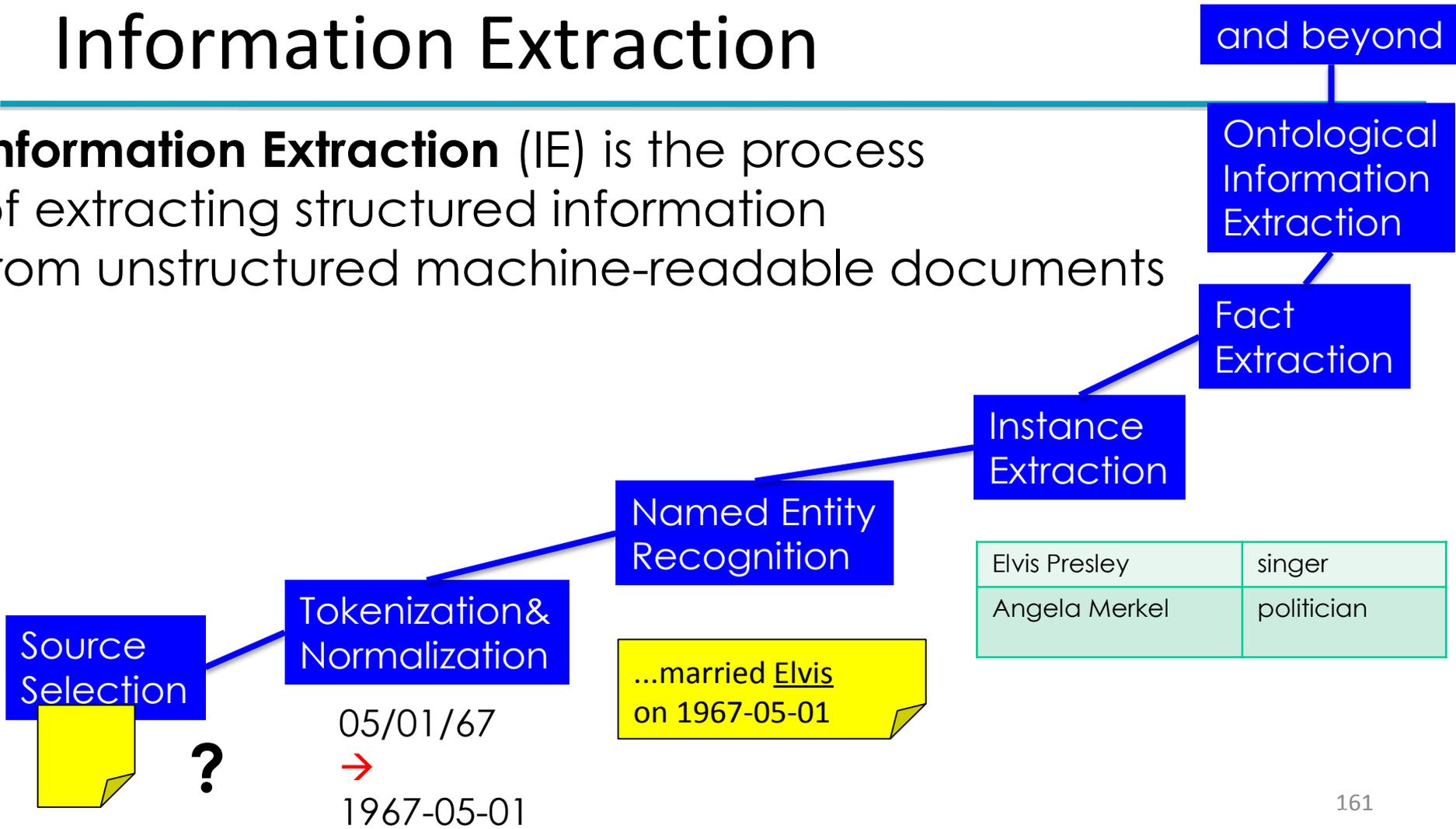


GName	FName	Occupation
Elvis	Presley	singer
Elvis	Hunter	painter
...	...	

“Seeing the Web as a table”

Information Extraction

Information Extraction (IE) is the process of extracting structured information from unstructured machine-readable documents



Information Extraction

Traditional definition: Recovering structured data from text

What are some of the sub-problems/challenges?

Management Team
Board of Directors
Our Firm & WOMMA
FAQs
Contact Us
Careers

Board Members

- **Itzhak Fisher**
Chairman of Nielsen BuzzMetrics
- **Thom Mastrelli**
Executive Vice President/Corporate Development, VNU
- **Jonathan Carson**
CEO of Nielsen BuzzMetrics
- **Mahendra Vora**
CEO and Owner, Vora Technology Park
- **Ori Levy**
President of Nielsen BuzzMetrics Israel
- **Ron Schneier**
Senior Vice President and General Manager, Nielsen Ventures
- **James O'Hara**
Senior Vice President and Chief Financial Officer, VNU's Media Measurement and Information Group

Information Extraction?

- Recovering structured data from text
 - Identifying fields (e.g. named entity recognition)

Management Team
Board of Directors
Our Firm & WOMMA
FAQs
Contact Us
Careers

Board Members

◦ Itzhak Fisher Chairman of Nielsen BuzzMetrics	◦ Ori Levy President of Nielsen BuzzMetrics Israel
◦ Thom Mastrelli Executive Vice President/Corporate Development, VNU	◦ Ron Schneier Senior Vice President and General Manager, Nielsen Ventures
◦ Jonathan Carson CEO of Nielsen BuzzMetrics	◦ James O'Hara Senior Vice President and Chief Financial Officer, VNU's Media Measurement and Information Group
◦ Mahendra Vora CEO and Owner, Vora Technology Park	

Information Extraction?

- Recovering structured data from text
 - Identifying fields (e.g. named entity recognition)
 - Understanding relations between fields (e.g. record association)

Management Team
Board of Directors
Our Firm & WOMMA
FAQs
Contact Us
Careers

Board Members

◦ Itzhak Fisher Chairman of Nielsen BuzzMetrics	◦ Ori Levy President of Nielsen BuzzMetrics Israel
◦ Thom Mastrelli Executive Vice President/Corporate Development, VNU	◦ Ron Schneier Senior Vice President and General Manager, Nielsen Ventures
◦ Jonathan Carson CEO of Nielsen BuzzMetrics	◦ James O'Hara Senior Vice President and Chief Financial Officer, VNU's Media Measurement and Information Group
◦ Mahendra Vora CEO and Owner, Vora Technology Park	

Information Extraction?

- Recovering structured data from text
 - Identifying fields (e.g. named entity recognition)
 - Understanding relations between fields (e.g. record association)
 - Normalization and deduplication

James O'Hara (I)



Date of birth (location)
[11 September 1927](#)
[Dublin, Ireland](#)

Date of death (details)
[3 December 1992](#)
 Glendale, California, USA.

Trivia
 Brother of [Maureen O'Hara](#)

Sometimes Credited As:
 James Lilburn / Jim O'Hara

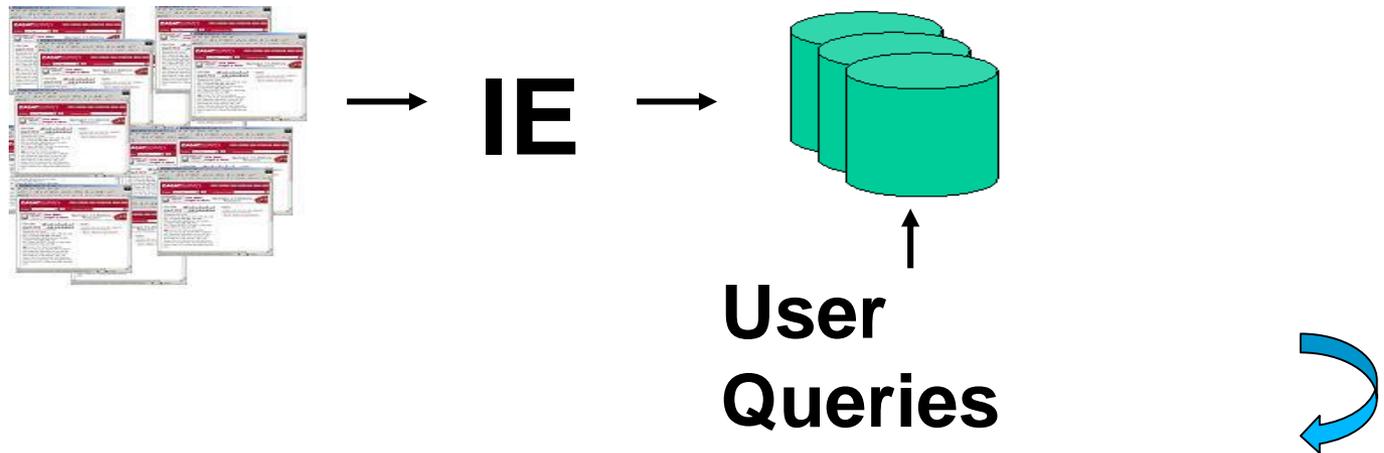
[IMDbPro Details](#) [Add IMDb Resume](#)

- **James O'Hara**
 Senior Vice President and Chief Financial Officer, VNU's Media Measurement and Information Group

Herkovic	Jane is a member of the Nielsen senior leadership team and a senior member of the VNU MMI Finance team. She is based in New York and reports to both Susan Whiting, president and CEO of Nielsen Media Research, and Jim O'Hara , senior vice president and chief financial officer for VNU Media Measurement and Information.
Susan D. Whiting	
Douglas Darfield	
Paul J. Donato	
Sara Erichson	
Dave Harkness	
Jack Loftus	

Information extraction

- Input: Text Document
 - Various sources: web, e-mail, journals, ...
- Output: Relevant fragments of text and relations possibly to be processed later in some automated way



Not all documents are created equal...



- Varying regularity in document collections
- Natural or unstructured
 - Little obvious structural information
- Partially structured
 - Contain some canonical formatting
- Highly structured
 - Often, automatically generated

Natural Text: MEDLINE

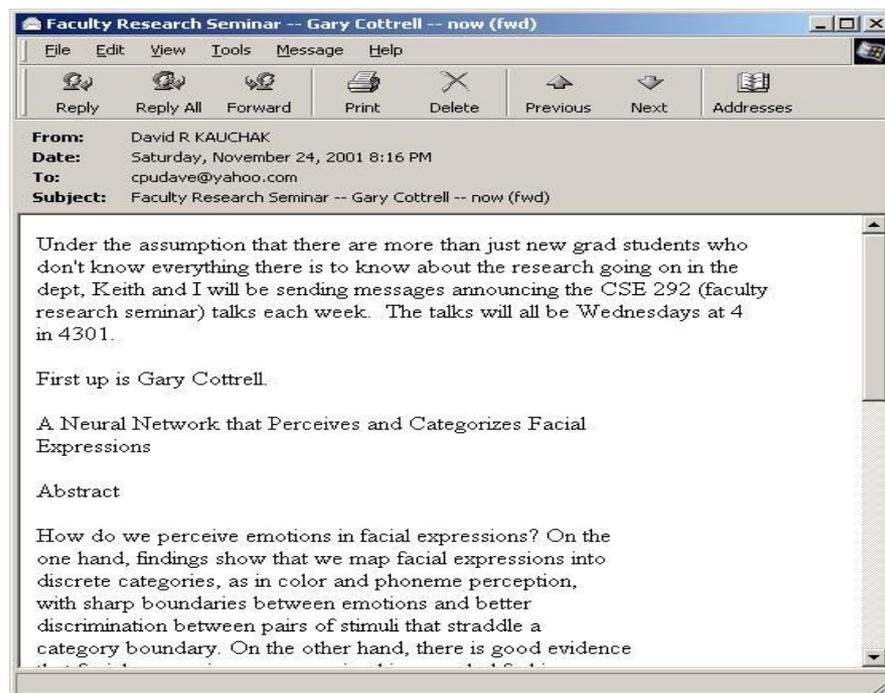
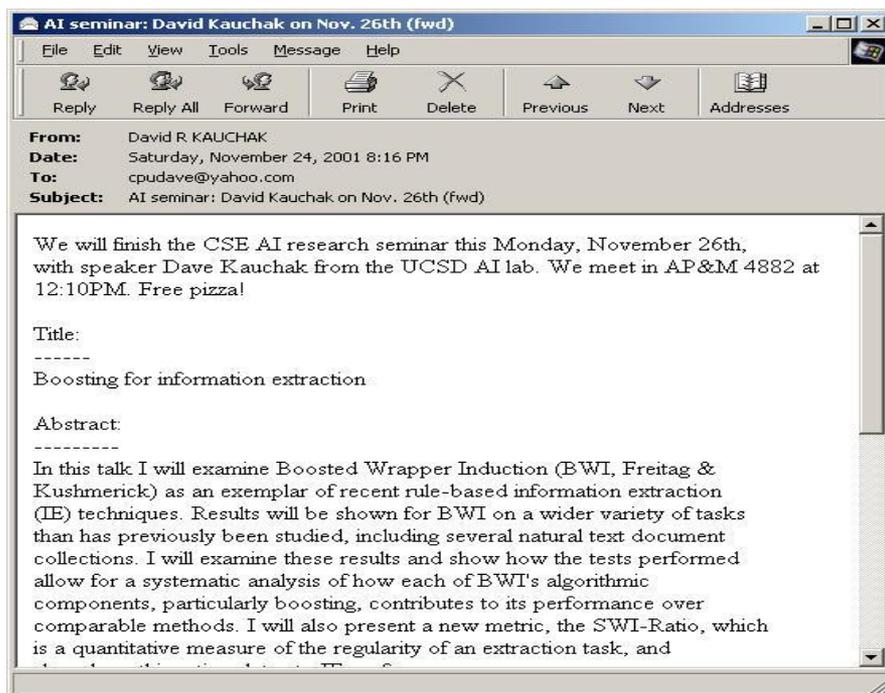
Journal Abstracts

Extract number of subjects, type of study, conditions, etc.

BACKGROUND: The most challenging aspect of revision hip surgery is the management of bone loss. A reliable and valid measure of bone loss is important since it will aid in future studies of hip revisions and in preoperative planning. We developed a measure of femoral and acetabular bone loss associated with failed total hip arthroplasty. The purpose of the present study was to **measure the reliability and the intraoperative validity of this measure** and to determine how it may be useful in preoperative planning. **METHODS:** From July 1997 to December 1998, **forty-five consecutive patients** with a failed hip prosthesis in need of revision surgery were prospectively followed. Three general orthopaedic surgeons were taught the radiographic classification system, and two of them classified standardized preoperative anteroposterior and lateral hip radiographs with use of the system. Interobserver testing was carried out in a **blinded fashion**. These results were then compared with the intraoperative findings of the third surgeon, who was blinded to the preoperative ratings. Kappa statistics (unweighted and weighted) were used to assess correlation. Interobserver reliability was assessed by examining the agreement between the two preoperative raters. Prognostic validity was assessed by examining the agreement between the assessment by either Rater 1 or Rater 2 and the intraoperative assessment (reference standard). **RESULTS:** With regard to the assessments of both the femur and the acetabulum, there was significant agreement ($p < 0.0001$) between the preoperative raters (reliability), with weighted kappa values of >0.75 . There was also significant agreement ($p < 0.0001$) between each rater's assessment and the intraoperative assessment (validity) of both the femur and the acetabulum, with weighted kappa values of >0.75 . **CONCLUSIONS:** With use of the newly developed classification system, preoperative radiographs are reliable and valid for assessment of the severity of bone loss that will be found intraoperatively.

Partially Structured: Seminar Announcements

Extract time, location, speaker, etc.



Information extraction pipeline

For years, [Microsoft Corporation](#) [CEO Bill Gates](#) was against open source. But today he appears to have changed his mind. "We can be open source. We love the concept of shared source," said [Bill Veghte](#), a [Microsoft VP](#). "That's a super-important shift for us in terms of code access."

[Richard Stallman](#), [founder](#) of the [Free Software Foundation](#), countered saying...

Name	Title	Organization
Bill Gates	CEO	Microsoft
Bill Veghte	VP	Microsoft
Richard Stallman	Founder	Free Soft..

The Full Task of Information Extraction

As a family of techniques:

Information Extraction =
segmentation + classification + association + clustering

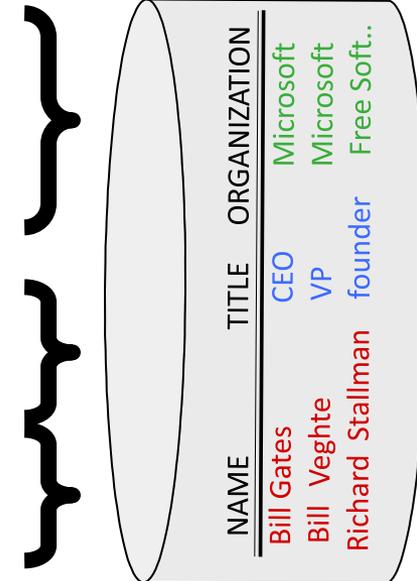
For years, [Microsoft Corporation](#) CEO [Bill Gates](#) railed against the economic philosophy of open-source software with Orwellian fervor, denouncing its communal licensing as a "cancer" that stifled technological innovation.

Now [Gates](#) himself says [Microsoft](#) will gladly disclose its crown jewels--the coveted code behind the Windows operating system--to select customers.

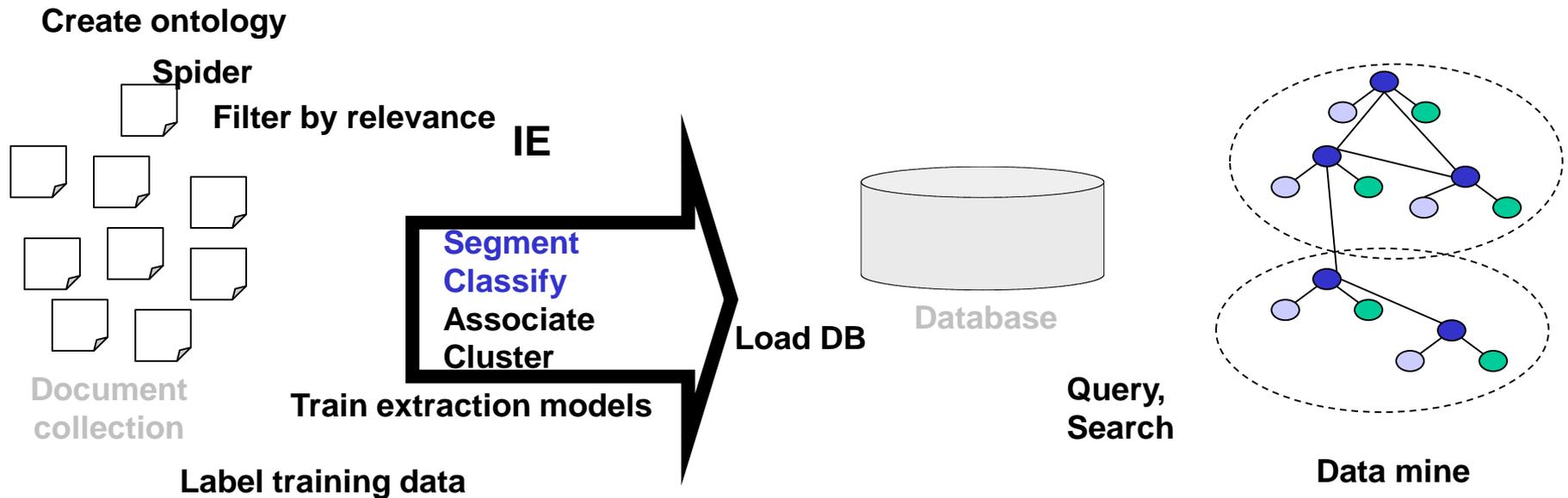
"We can be open source. We love the concept of shared source," said [Bill Veghte](#), a [Microsoft](#) VP. "That's a super-important shift for us in terms of code access."

[Richard Stallman](#), founder of the [Free Software Foundation](#), countered saying...

Microsoft Corporation CEO Bill Gates
Gates Microsoft
Bill Veghte Microsoft VP
Richard Stallman founder Free Software Foundation



An Even Broader View



Landscape of IE Tasks:

Document Formatting

Text paragraphs without formatting

Astro Teller is the CEO and co-founder of BodyMedia. Astro holds a Ph.D. in Artificial Intelligence from Carnegie Mellon University, where he was inducted as a national Hertz fellow. His M.S. in symbolic and heuristic computation and B.S. in computer science are from Stanford University.

Non-grammatical snippets, rich formatting & links

Barto, Andrew G. Professor. Computational neuroscience, reinforcement learning, adaptive motor control, artificial neural networks, adaptive and learning control, motor development.	(413) 545-2109	barto@cs.umass.edu	CS276	 
Berger, Emery D. Assistant Professor.	(413) 577-4211	emery@cs.umass.edu	CS344	 
Brock, Oliver Assistant Professor.	(413) 577-0334	oli@cs.umass.edu	CS246	 
Clarke, Lori A. Professor. Software verification, testing, and analysis; software architecture and design.	(413) 545-1328	clarke@cs.umass.edu	CS304	 

Grammatical sentences and some formatting & links

Dr. Steven Minton - Founder/CTO
Dr. Minton is a fellow of the American Association of Artificial Intelligence and was the founder of the Journal of Artificial Intelligence Research. Prior to founding Fetch, Minton was a faculty member at USC and a project leader at USC's Information Sciences Institute. A graduate of Yale University and Carnegie Mellon University, Minton has been a Principal Investigator at NASA Ames and taught at Stanford, UC Berkeley and USC.

- Press
- Contact**
- General information
- Directions maps

Tables

8:30 - 9:30 AM	Invited Talk: Plausibility Measures: A General Approach for Representing Uncertainty <i>Joseph Y. Halpern, Cornell University</i>				
9:30 - 10:00 AM	Coffee Break				
10:00 - 11:30 AM	Technical Paper Sessions:				
Cognitive Robotics	Logic Programming	Natural Language Generation	Complexity Analysis	Neural Networks	Games
739: A Logical Account of Causal and Topological Maps <i>Emilio Remolina and Benjamin Kuipers</i>	116: A-System: Problem Solving through Abduction <i>Marc Denecker, Antonis Kakas, and Bert Van Nuffelen</i>	758: Title Generation for Machine-Translated Documents <i>Rong Jin and Alexander G. Hauptmann</i>	417: Let's go Nats: Complexity of Nested Circumscription and Abnormality Theories <i>Marco Cadoli, Thomas Eiter, and Georg Gottlob</i>	179: Knowledge Extraction and Comparison from Local Function Networks <i>Kenneth McGarry, Stefan Wermter, and John MacIntyre</i>	71: Iterative Widening <i>Tristan Cazenave</i>
549: Online-Execution of ccGolog Plans <i>Henrik Grosskreutz and Gerhard Lakemeyer</i>	131: A Comparative Study of Logic Programs with Preference <i>Torsten Schaub and Kewen</i>	246: Dealing with Dependencies between Content Planning and Surface Realisation in a Pipeline Generation	470: A Perspective on Knowledge Compilation <i>Adnan Darwiche and Pierre Marquis</i>	258: Violation-Guided Learning for Constrained Formulations in Neural-Network Time-Series	353: Temporal Difference Learning Applied to a High Performance Game-Playing

Landscape of IE Tasks

Intended Breadth of Coverage

Web site specific

Formatting

Amazon.com Book Pages

The screenshot shows the Amazon.com interface for the book 'Learning in Graphical Models' by Michael Irwin Jordan. It features a search bar, navigation tabs (WELCOME, YOUR STORE, BOOKS, ELECTRONICS, DVD, TOYS & GAMES), and a 'NEW Super Saver Shipping FREE' banner. The book cover is visible with a 'LOOK INSIDE!' feature. Below the book title, it lists the list price as \$60.00 and the current price as \$60.00. There is a 'Buy Now' button and a 'Great Buy' section suggesting a bundle with 'Probabilistic Reasoning in Intelligent Systems' for a total price of \$128.95.

Genre specific

Layout

Resumes

The screenshot shows a resume for Jason D. M. Rennie. It includes contact information for MIT AI Lab, research interests in automated analysis of data, and a detailed work history. The work history includes positions at Carnegie Mellon University (Wean Hall, School of Computer Science) and the Technical University of Berlin, as well as an Exchange Fellowship at the University of Michigan. It also lists education (Ph.D. in Computer Science, M.S. in Computer Science) and research experience at Carnegie Mellon University from 1994 to the present.

Wide, non-specific

Language

University Names

The screenshot shows a conference schedule and contact information. The schedule includes an invited talk by Joseph Y. Halpern at 8:30-9:30 AM, a coffee break at 9:30-10:00 AM, and technical paper sessions from 10:00-11:30 AM. The sessions are categorized into Cognitive Robotics, Logic Programming, Natural Language Generation, and Complexity Analysis. Below the schedule, there is a contact section for Dr. Steven Minton, Founder/CTO, with a list of contact options: Press, General information, and Directions maps.

8:30 - 9:30 AM	Invited Talk: Plausibility Measures: A General Approach Joseph Y. Halpern, Cornell University		
9:30 - 10:00 AM	Coffee Break		
10:00 - 11:30 AM	Technical Paper Sessions:		
Cognitive Robotics	Logic Programming	Natural Language Generation	Complexity Analysis
739: A Logical Account of Causal and Topological Maps <i>Emilio Remolina and Benjamin Kuipers</i>	116: A-System: Solving Problem Abduction through Marc Denecker,	758: Title Generation for Machine-Translated Documents <i>Rong Jin and Alexander G. Hauptmann</i>	417: Let's go Nats: Complexity of Nested Circumscription and Abnormality

Dr. Steven Minton - Founder/CTO
Dr. Minton is a fellow of the American Association of Artificial Intelligence and was the founder of the Journal of Artificial Intelligence Research. Prior to founding Fetch, Minton was a faculty member at USC and a project leader at USC's Information Sciences Institute. A graduate of Yale University and Carnegie Mellon University, Minton has been a Principal Investigator at NASA Ames and taught at Stanford, UC Berkeley and USC.

Frank Huybrechts - COO
Mr. Huybrechts has over 20 years of

- Press
- General information
- Directions maps

Landscape of IE Tasks :

Complexity of entities/relations

Closed set

U.S. states

He was born in Alabama...

The big Wyoming sky...

Regular set

U.S. phone numbers

Phone: (413) 545-1323

The CALD main office is 412-268-1299

Complex pattern

U.S. postal addresses

University of Arkansas

P.O. Box 140

Hope, Al

Headquarters:

1128 Main Street, 4th Floor

Cincinnati, Ohio 45210

Ambiguous patterns, needing context and many sources of evidence

Person names

...was among the six houses sold by Hope Feldman that year.

Pawel Opalinski, Software Engineer at WhizBang Labs.

Landscape of IE Tasks:

Arity of relation

Jack Welch will retire as CEO of General Electric tomorrow. The top role at the Connecticut company will be filled by Jeffrey Immelt.

Single entity

Person: Jack Welch

Person: Jeffrey Immelt

Location: Connecticut

Binary relationship

Relation: Person-Title

Person: Jack Welch

Title: CEO

Relation: Company-Location

Company: General Electric

Location: Connecticut

N-ary record

Relation: Succession

Company: General Electric

Title: CEO

Out: Jack Welsh

In: Jeffrey Immelt

"Named entity" extraction

Resolving coreference (both within and across documents)

John Fitzgerald Kennedy was born at 83 Beals Street in Brookline, Massachusetts on Tue 29, 1917, at 3:00 pm,[7] the second son of Joseph P. Kennedy, Sr., and Rose Fitzgerald; R turn, was the eldest child of John "Honey Fitz" Fitzgerald, a prominent Boston political fi was the city's mayor and a three-term member of Congress. Kennedy lived in Brookline years and attended Edward Devotion School, Noble and Greenough Lower School, and the Dexter School, through 4th grade. In 1927, the family moved to 5040 Independence Avenue in Bronx, New York City; two years later, they moved to 294 Pondfield Road in Bronxville, N where Kennedy was a member of Scout Troop 2 (and was the first Boy Scout to become President).[8] Kennedy spent summers with his family at their home in Hyannisport, Massachusetts, and Christmas and Easter holidays with his family at their winter home in Beach, Florida. For the 5th through 7th grade, Kennedy attended Riverdale Country School, a private school for boys. For 8th grade in September 1930, the 13-year old Kennedy attended Canterbury School in New Milford, Connecticut.



Association task = Relation Extraction

- Checking if groupings of entities are instances of a relation
 1. Manually engineered rules
 - Rules defined over words/entities: “<company> located in <location>”
 - Rules defined over parsed text:
 - “((Obj <company>) (Verb located) (*) (Subj <location>))”
 2. Machine Learning-based
 - Supervised: Learn relation classifier from examples
 - Partially-supervised: bootstrap rules/patterns from “seed” examples

Rough Accuracy of Information Extraction

Information type	Accuracy
Entities	90-98%
Attributes	80%
Relations	60-70%
Events	50-60%

- Errors cascade (error in entity tag → error in relation extraction)
- These are very rough, actually optimistic, numbers
 - Hold for well-established tasks, but lower for many specific/novel IE tasks

TEXT BOOKS

1. Christopher D. Manning, Prabhakar Raghavan & Hinrich Schütze. **Introduction to Information Retrieval**, Cambridge University Press, 2008
2. Sunita Sarawagi. **Information Extraction**. *Foundations and Trends in Databases*, 1(3):261–377, 2008

THANKYOU

Slides of this presentation is available at
<http://priyaradhakrishnan.weebly.com/>