Intelligent Information Retrieval

Boolean and Vector Space Retrieval Models

Credits

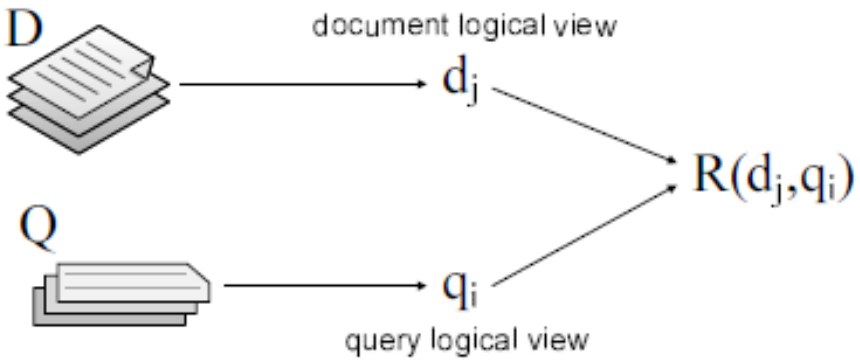
- Christopher Manning
- Prabhakar Raghavan
- Francesco Ricci
- Marco de Gemmis
- Pasquale Lops
- Giovanni Semeraro

Retrieval Models

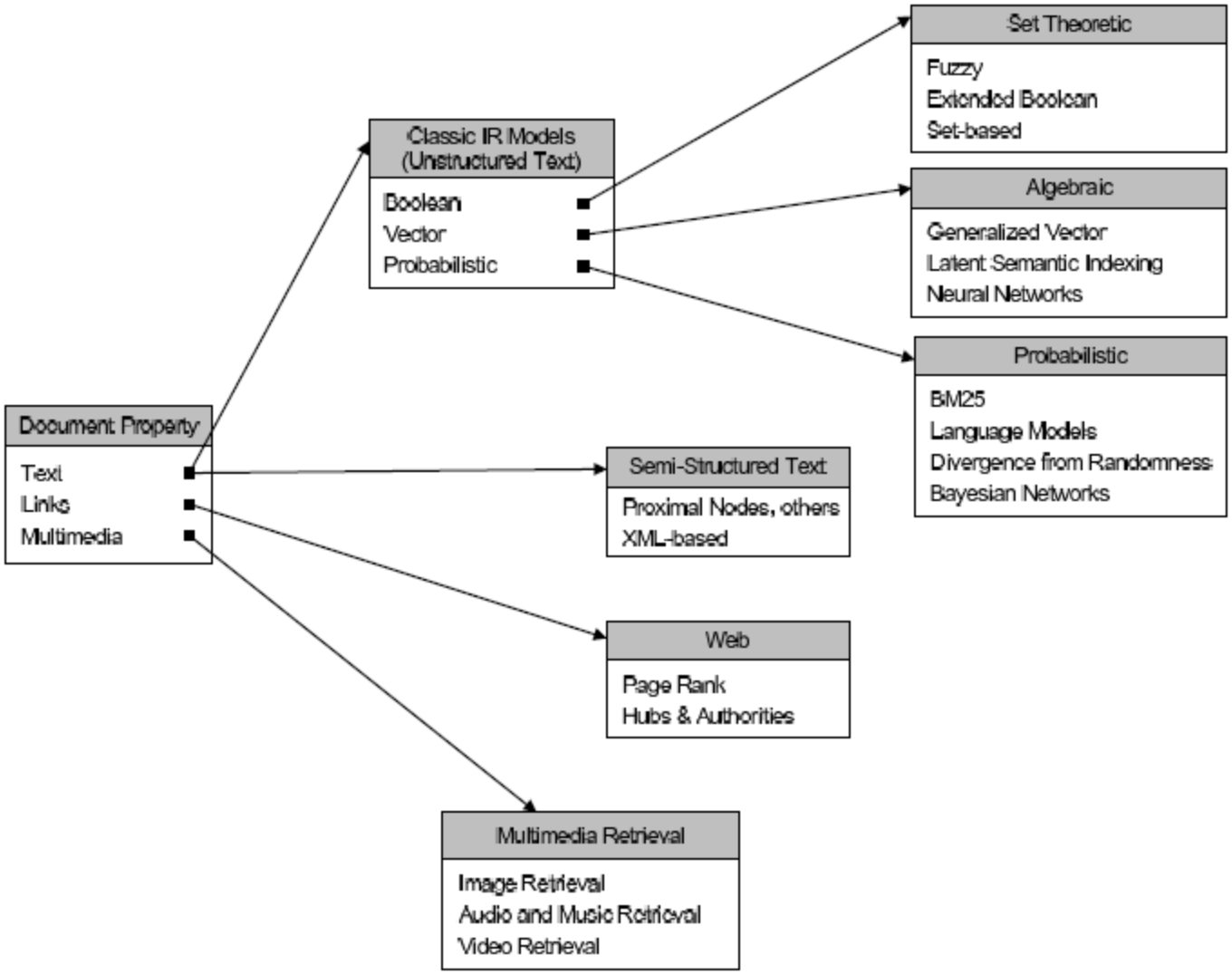
- A retrieval model specifies the details of:
  - ✓ Document representation
  - ✓ Query representation
  - ✓ Retrieval function
- Determines a notion of relevance.
- Notion of relevance can be binary or continuous (i.e. *ranked retrieval*).
- Ranking function: a function that assigns scores to documents with regard to a given query

IR Models

- An IR model is a quadruple [D, Q, *F*, $R(q_i, d_j)$] where
  - ✓ 1. D is a set of logical views for the documents in the collection
  - ✓ 2. Q is a set of logical views for the user queries
  - ✓ 3. *F* is a framework for modeling documents and queries
  - ✓ 4. $R(q_i, d_j)$ is a ranking function



A Taxonomy of IR Models



Classes of Retrieval Models

- Boolean models (set theoretic)
  - ✓ Extended Boolean
- Vector space models (statistical/algebraic)
  - ✓ Generalized VS
  - ✓ Latent Semantic Indexing
- Probabilistic models

Common Preprocessing Steps

- Strip unwanted characters/markup (e.g. HTML tags, punctuation, numbers, etc.).
- Break into *tokens* (keywords) on whitespace.
- Stem tokens to "root" words (retrieval independent of tense, number,…)
  - ✓ computational → comput

  Open problems:
  - ✓ Errors (policies, police → polic)
  - ✓ Loss of context (does → do)
  - ✓ Stem still a word?

Common Preprocessing Steps (cont'd)

References on stemming:

Porter, M., *An algorithm for suffix stripping*, Program, 14(3), 130-137, 1980.

Frakes, W., *Stemming algorithms*, in Frakes, W. & Baeza-Yates, Information Retrieval Data Structures and Algorithms, Prentice-Hall, 1992.

- Remove common stopwords (e.g. a, the, it, etc.).
- Detect common phrases (possibly using a domain specific dictionary).
- Build inverted index (keyword → list of docs containing it).

Boolean Model

- A document is represented as a set of keywords.
- Queries are Boolean expressions of keywords, connected by AND, OR, and NOT, including the use of brackets to indicate scope.
- Output: Document is relevant or not. No partial matches or ranking.

Term-document incidence

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

1 if play contains word, 0 otherwise

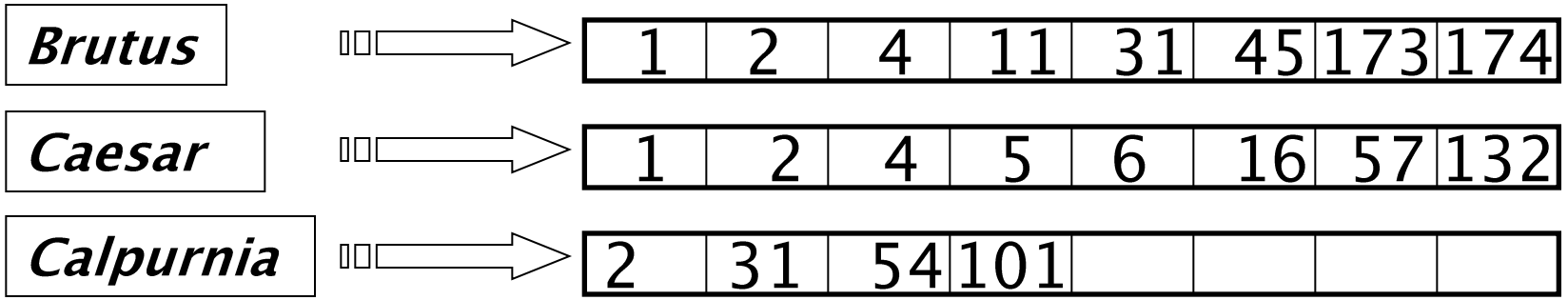
Brutus *AND* ***Caesar*** *BUT NOT* ***Calpurnia***

Incidence vectors

- So we have a 0/1 vector for each term
- To answer query:
  - ✓ ***Brutus, Caesar*** and NOT ***Calpurnia***
  - ✓ take the vectors for
    - ***Brutus*** 110100
    - ***Caesar*** 110111
    - ***Calpurnia*** (complemented) 101111
  - ✓ Bitwise *AND*
    - 110100 *AND* 110111 *AND* 101111 = 100100

Inverted index

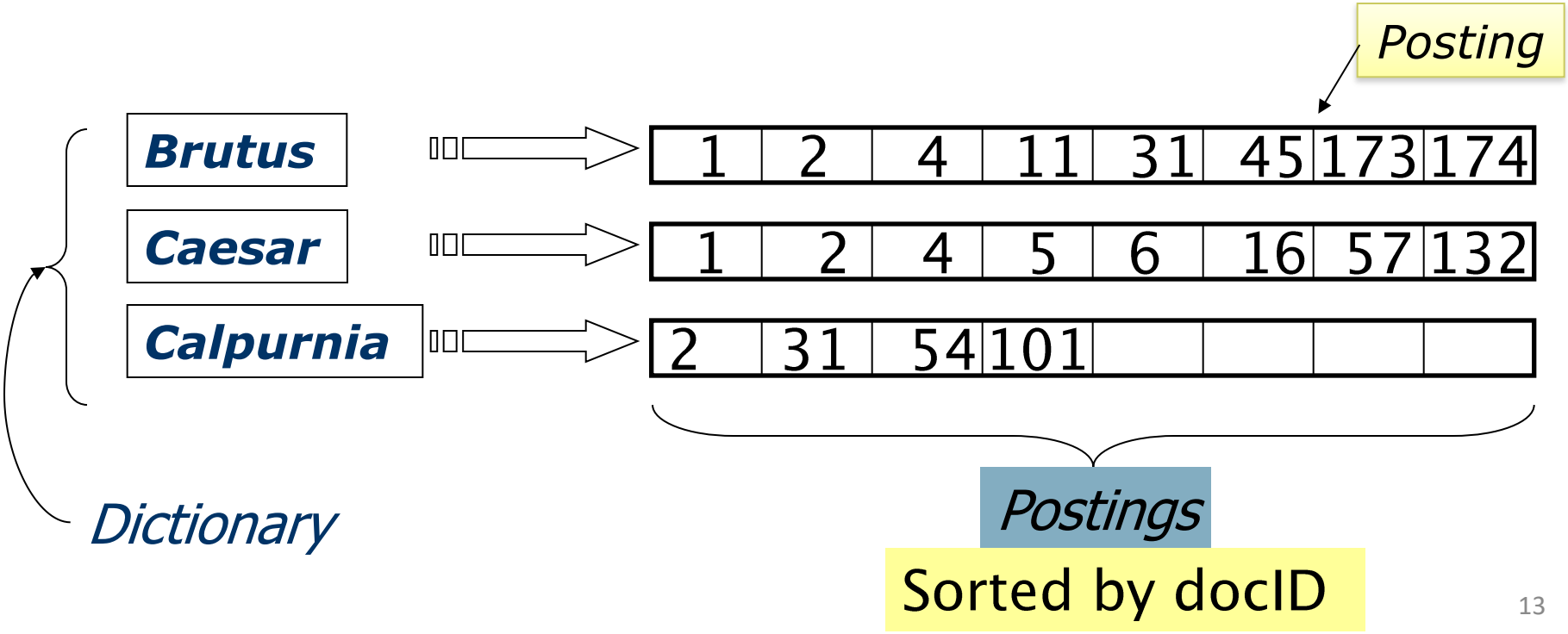
- For each term *t*, we must store a list of all documents that contain *t*
  - ✓ Identify each by a **docID**, a document serial number
- Can we used fixed-size arrays for this?



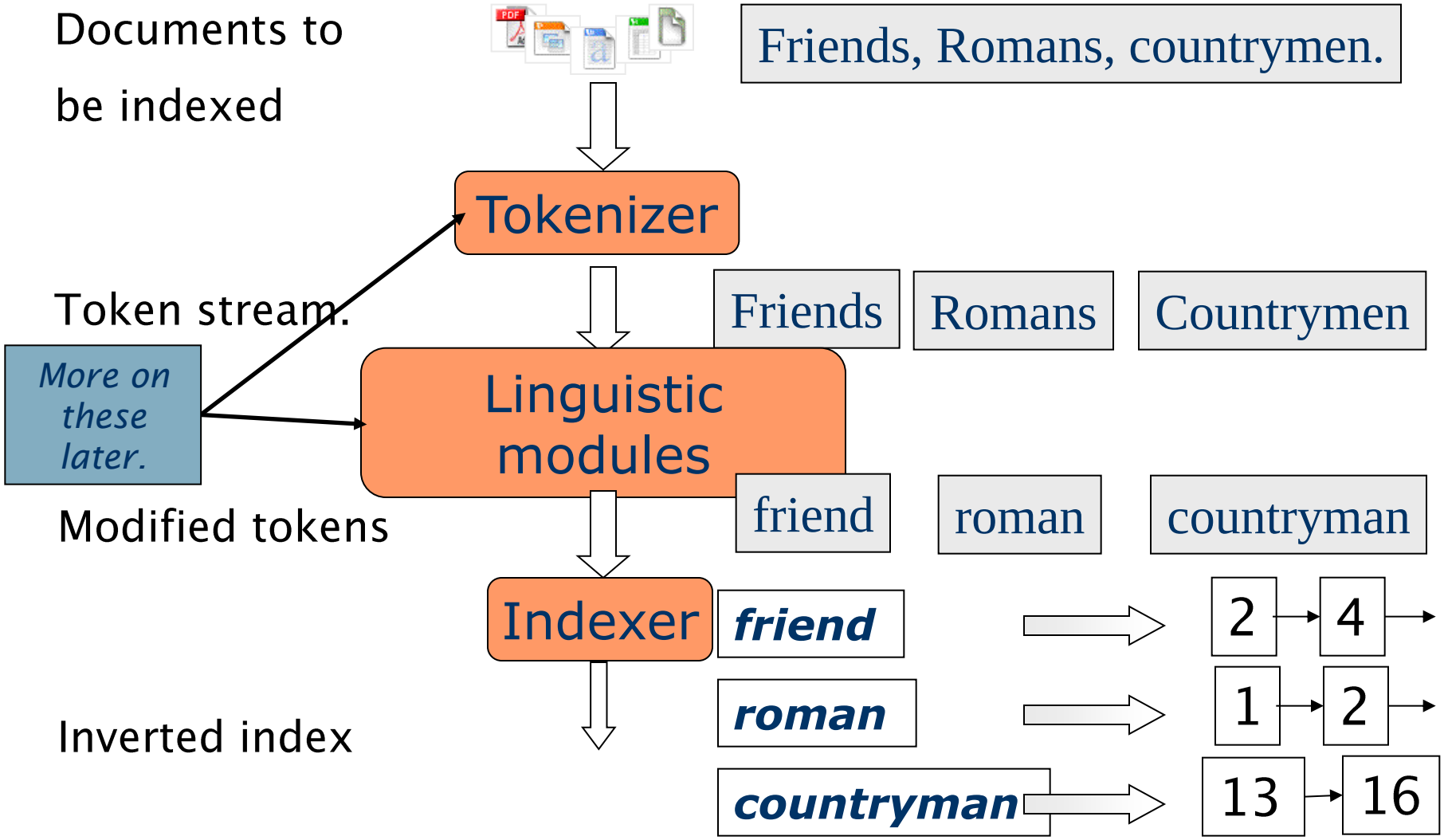
What happens if the word ***Caesar*** is added to document 14?

Inverted index

- We need variable-size postings lists
  - ✓ On disk, a continuous run of postings is normal and best
  - ✓ In memory, can use linked lists or variable length arrays
    - Some tradeoffs in size/ease of insertion



Inverted index construction



Indexer steps: Token sequence

- Sequence of (Modified token, Document ID) pairs.

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

Indexer steps: Sort

- Sort by terms
  - ✓ And then docID

Core indexing step

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

Indexer steps: Dictionary & Postings

- Multiple term entries in a single document are merged
- Split into Dictionary and Postings
- Doc. frequency information is added.

Why frequency?

Will discuss later

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

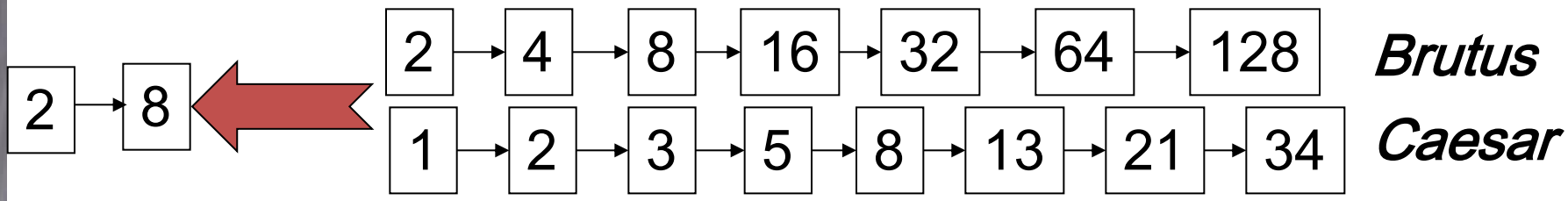
term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

Query processing: AND

- Consider processing the query:

  Brutus *AND* ***Caesar***

  - ✓ Locate ***Brutus*** in the Dictionary
    - Retrieve its postings
  - ✓ Locate ***Caesar*** in the Dictionary
    - Retrieve its postings
  - ✓ "Merge" the two postings



Crucial: postings sorted by docID and processed in order of increasing posting list length

Intersecting two postings lists (a "merge" algorithm)

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

Boolean queries: Exact match

- The Boolean retrieval model is being able to ask a query that is a Boolean expression:
  - ✓ Boolean Queries are queries using *AND, OR* and *NOT* to join query terms
    - Views each document as a set of words
    - Is precise: document matches condition or not.
  - ✓ Perhaps the simplest model to build an IR system on
- Primary commercial retrieval tool for 3 decades
- Many search systems you still use are Boolean:
  - ✓ Email, library catalog, Mac OS X Spotlight.

Boolean Models – Problems

- Very rigid: AND means all (too few results); OR means any (too many results).
- Difficult to express complex user requests
  - Information need to be translated into a Boolean expression
- Difficult to control the number of documents retrieved.
  - *All* matched documents will be returned.
- Difficult to rank output.
  - *All* matched documents logically satisfy the query.
- Difficult to perform relevance feedback.
  - If a document is identified by the user as relevant or irrelevant, how should the query be modified?

PRE-PROCESSING STEPS

Parsing a document

- What format is it in?
  - pdf/word/excel/html?
- What language is it in?
- What character set encoding is in use?
- Each of these is a **classification problem**
- But these tasks are often done heuristically:
  - The classification is predicted with simple rules
  - Example: "if there are many `the' then it is English".

Complications: Format/language

- Documents being indexed can include docs from many different languages
  - A single index may have to contain terms of several languages
- Sometimes a document or its components can contain multiple languages/formats
  - French email with a German pdf attachment
- What is a unit document?
  - A file?
  - An email? (Perhaps one of many in a mbox)
  - An email with 5 attachments?
  - A group of files (PPT or LaTeX as HTML pages).

TOKENS AND TERMS

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?

Tokenization

- Issues in tokenization:
  - ***Finland's capital*** → ***Finland? 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?

Numbers

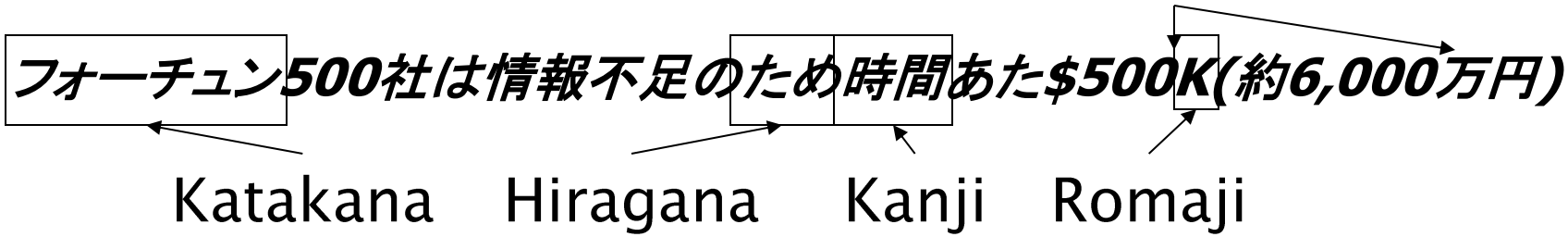
- ***3/20/91 Mar. 12, 1991 20/3/91***
- ***55 B.C.***
- ***B-52***
- ***My PGP key is 324a3df234cb23e***
- ***(800) 234-2333***
  - ✓ Often have embedded spaces
  - ✓ Older IR systems may not index numbers
    - But often very useful: think about things like looking up error codes/stacktraces on the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

Tokenization: language issues

- French
  - ***L'ensemble*** → one token or two?
    - ***L*** ? ***L'*** ? ***Le*** ?
    - Want ***l'ensemble*** to match with ***un ensemble***
      - Until at least 2003, it didn't on Google
        - Internationalization!
- 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.

Tokenization: language issues

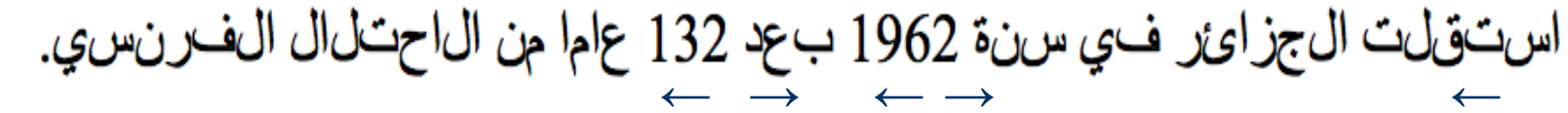
- Chinese and Japanese have no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - Not always guaranteed a unique tokenization
- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats



End-user can express query entirely in hiragana!

Tokenization: language issues

- Arabic (or Hebrew) is basically written right to left, but with certain items like numbers written left to right



start

'Algeria achieved its independence in 1962 after 132 years of French occupation.'

- With Unicode, the surface presentation is complex, but the stored form is straightforward.

Stop words

- With a stop list, you **exclude** from the dictionary entirely **the commonest words**:
  - ✓ 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 means the space for including stopwords in a system is very small
  - ✓ Good query optimization techniques 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 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 implicitly define equivalence classes of terms by, e.g.,
  - ✓ deleting periods to form a term
    - ***U.S.A., USA*** $\in$ ***[USA]***
  - ✓ deleting hyphens to form a term
    - ***anti-discriminatory, antidiscriminatory*** $\in$ ***[antidiscriminatory]***

Equivalence class of a

$[a] = \{x \mid x \sim a\}$

Normalization: other languages

- Accents: e.g., French ***résumé*** vs. ***resume***
- Umlauts: e.g., German: ***Tuebingen*** vs. ***Tübingen***
  - ✓ Should be equivalent
- 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
    - ***Tuebingen, Tübingen, Tubingen*** $\in$ ***[Tubingen]***

Normalization: other languages

- Normalization of things like date forms
  - ✓ ***7月30日 vs. 7/30***
  - ✓ ***Japanese use of kana vs. Chinese characters***

- Tokenization and normalization may depend on the language and so is intertwined with language detection

Morgen will ich in MIT …

Is this German "mit"?

- *Crucial: need to "normalize" indexed text as well as query terms into the same form.*

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…

Federal reserve

Steel Authority of India

- Google example:
  - ✓ Query ***C.A.T.***
  - ✓ #1 result is for Caterpillar Inc., then "usual" cat
- What is the impact of normalization on precision and recall?

Caterpillar: Home
Caterpillar is the world's leading manufacturer of construction and mining equipment, diesel and natural gas engines, industrial gas turbines and a wide and ... Show stock quote for CAT
Caterpillar Products - Machine Specs - Careers - Engine Specs
www.cat.com/ - Cached - Similar

Cat - Wikipedia, the free encyclopedia
The cat (Felis silvestris catus), also known as the domestic cat or housecat to distinguish it from other felines and felids, is a small carnivorous mammal ...
File - Body language - Diet - Intelligence
en.wikipedia.org/wiki/Cat - Cached - Similar

Lolcats 'n' Funny Pictures of Cats - I Can Has Cheezburger?
2 Feb 2010 ... Humorous captioned pictures of felines and other animals. Visitors can submit their own material or add captions to a large archive of ...

Normalization to terms

- An alternative to equivalence classing is to include in the dictionary many variants of a term and then do asymmetric expansion at query time
- An example of where this may be useful
  - ✓ User enters: ***window*** System searches: ***window, windows***
  - ✓ Enter: ***windows*** Search: ***Windows, windows, window***
  - ✓ Enter: ***Windows*** Search: ***Windows***
- Potentially more powerful, but less efficient (Why?)

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

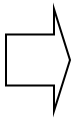
Lemmatization

- Reduce inflectional/variant forms to base form (the one that you search in your English dictionary)
- 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" suggest 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 equival 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**.*
- Girl**s**

"s" is the suffix

"Girl" is the stem

Examples

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

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

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

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

SCORING, TERM WEIGHTING AND VECTOR SPACE MODEL

Content

- Ranked retrieval
- Scoring documents
- Term frequency
- Collection statistics
- Tf-idf
- Weighting schemes
- Vector space scoring

Boolean retrieval

- Thus far, our queries have all been Boolean:
  - ✓ documents either **match** or **don't**
- Good for expert users with precise understanding of their needs and the collection
- Also good for applications: applications can easily consume 1000s of results
- **Not good for the majority of users**
  - ✓ Most users incapable of writing Boolean queries (or they are, but they think it's too much work)
  - ✓ Most users **don't want to wade through 1000s of results**
    - This is particularly true of web search.

Problem with Boolean search:feast or famine

- Boolean queries often result in **either too few** (=0) **or too many** (1000s) **results**
- Query 1: "*standard user dlink 650*" → 200,000 hits
- Query 2: "*standard user dlink 650 no card found*": 0 hits
- It takes a lot of skill to come up with a query that produces a manageable number of hits
- AND gives too few; OR gives too many.

Ranked retrieval models

- Rather than a set of documents satisfying a query expression, in **ranked retrieval models**, the system returns an ordering over the (top) documents in the collection with respect to a query
- **Free text queries:** Rather than a query language of operators and expressions, the user's query is just one or more words in a human language
- In principle, there are two separate choices here – the **query language** and the **retrieval model** - but in practice, ranked retrieval models have normally been associated with free text queries.

Feast or famine: not a problem in ranked retrieval

- When a system produces a ranked result set, **large result sets are not an issue**
  - ✓ Indeed, the size of the result set is not an issue
  - ✓ We just show the top *k* (≈ 10) results
  - ✓ We don't overwhelm the user
  - ✓ ***Premise: the ranking algorithm works***

Do you really agree with that?

Scoring as the basis of ranked retrieval

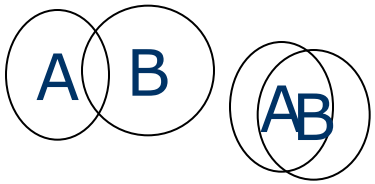
- We wish to return in order the documents most likely to be useful to the searcher
- How can we rank-order the documents in the collection with respect to a query?
- Assign a score – say in [0, 1] – to each document
- This score measures how well document and query "match".

Query-document matching scores

- We need a way of assigning a score to a query/document pair
- Let's start with a one-term query
- If the query term does not occur in the document:
  - The score should be 0
  - *Why? Can we do better?*
- The more frequent the query term in the document, the higher the score (should be)
- We will look at a number of alternatives for this.

Take 1: Jaccard coefficient

- A commonly used measure of overlap of two sets *A* and *B*
- jaccard*(A,B)* = $|A \cap B| \,/\, |A \cup B|$
- jaccard*(A,A)* = 1
- jaccard*(A,B)* = 0 if $A \cap B = 0$
- *A* and *B* don't have to be the same size
- Always assigns a number between 0 and 1



Jaccard coefficient: Scoring example

- What is the query-document match score that the Jaccard coefficient computes for each of the two documents below?
- Query: *ides of march*
- Document 1: *caesar died in march*
- Document 2: *the long march*

- jaccard*(Q,D)* = $|Q \cap D| / |Q \cup D|$

- *jaccard(Query, Document1) = 1/6*
- *jaccard(Query, Document2) = 1/5*

Issues with Jaccard for scoring

- It doesn't consider *term frequency* (how many times a term occurs in a document)
- Rare terms in a collection are more informative than frequent terms - Jaccard doesn't consider this information
- We need also a more sophisticated way of normalizing for length

Recall: Binary term-document incidence matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Each document is represented by a binary vector $\in \{0,1\}^{|V|}$.

Term-document count matrices

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

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

Bag of words model

- Vector representation doesn't consider the ordering of words in a document
- *"John is quicker than Mary"* and *"Mary is quicker than John"* have the same vectors
- This is called the bag of words model
- In a sense, this is a step back: the positional index was able to distinguish these two documents
- We will look at "recovering" positional information later in this course
- For now: bag of words model.

Term frequency tf

- The term frequency $tf_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*
- We want to use *tf* when computing query-document match scores - but how?
- Raw term frequency is not what we want:
  - ✓ A document with 10 occurrences of the term **is more relevant** than a document with 1 occurrence of the term.
  - ✓ **But not 10 times** more relevant
- Relevance does not increase proportionally with term frequency.

NB: *frequency = count* - in IR

Log-frequency weighting

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

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

- $0 \rightarrow 0$, $1 \rightarrow 1$, $2 \rightarrow 1.3$, $10 \rightarrow 2$, $1000 \rightarrow 4$, etc.
- Score for a document-query pair: sum over terms *t* in both *q* and *d*:

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

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

Normal vs. Sublinear tf scaling

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

- The above formula defined the **sublinear tf-scaling**
- The simplest approach (**normal**) is to use the number of occurrences of the term in the document (frequency)
- But as discussed earlier sublinear tf should be better.

Properties of the Logarithms

- $y = \log_a x$ iff $x = a^y$
- $\log_a 1 = 0$
- $\log_a a = 1$
- $\log_a (xy) = \log_a x + \log_a y$
- $\log_a (x/y) = \log_a x - \log_a y$
- $\log_a (x^b) = b \log_a x$
- $\log_b x = \log_a x / \log_a b$
- $\log x$ is typically $\log_{10} x$
- $\ln x$ is typically $\log_e x$ ($e = 2.7182...$ Napier or Euler number) – Natural logarithm.

Document frequency

- Rare terms – in the whole collection - are more informative than frequent terms
  - ✓ Recall stop words
- Consider a term in the query that is rare in the collection (e.g., *arachnocentric*)
- A document containing this term is very likely to be relevant to the query *arachnocentric*
- → We want a high weight for rare terms like *arachnocentric*.

Document frequency, *cont'd*

- Generally frequent terms are **less informative** than rare terms
- Consider a **query** term that is frequent in the collection (e.g., *high*, *increase*, *line*)
- A document containing such a term is more likely to be relevant than a document that doesn't
- **But** consider a query containing two terms – e.g.: high *arachnocentric*
- For a **frequent** term in a document, s.a., high, we want a positive weight but **lower** than for terms that are **rare** in the collection, s.a., *arachnocentric*
- We will use **document frequency** (df) to capture this.

idf weight

- df_t is the document frequency of t: the number of documents that contain t
  - ✓ df_t is an inverse measure of the informativeness of t *(the smaller the better)*
  - ✓ $df_t \leq N$
- We define the idf (**inverse document frequency**) of t by

$$idf_t = \log(N/df_t)$$

Is a function of t only – does not depend on the document

  - ✓ We use log (N/df_t) instead of N/df_t to "dampen" the effect of idf.

idf example, suppose N = 1 million

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

$$\text{idf}_t = \log\left(N/\text{df}_t\right)$$

There is one idf value for each term t in a collection.

Effect of idf on ranking

- Can *idf* have an effect on ranking for one-term queries? E.g. like:
  - iPhone
- idf has no effect on ranking one term queries - since there is one idf value for each term in a collection
  - idf affects the ranking of documents for queries with at least two terms
  - For the query capricious person, idf weighting makes occurrences of capricious count for much more in the final document ranking than occurrences of person.

Collection vs. Document frequency

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

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

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

tf-idf weighting

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

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

- Best known weighting scheme in information retrieval
  - ✓ Note: the "-" in tf-idf is a hyphen, not a minus sign!
  - ✓ Alternative names: tf.idf, tf x idf

1. **Increases** with the **number of occurrences within a document**
2. **Increases** with the **rarity of the term in the collection.**

Final ranking of documents for a query

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

Binary → count → weight matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

dimensions

Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$

Documents as vectors

- So we have a |V|-dimensional vector space
  - ✓ **Terms** are **axes** of the space
  - ✓ **Documents** are **points** or vectors in this space
- Very high-dimensional:
  - ✓ 400,000 in RCV1
  - ✓ tens of millions of dimensions when you apply this to a web search engine
- These are very sparse vectors - most entries are zero.

Queries as vectors

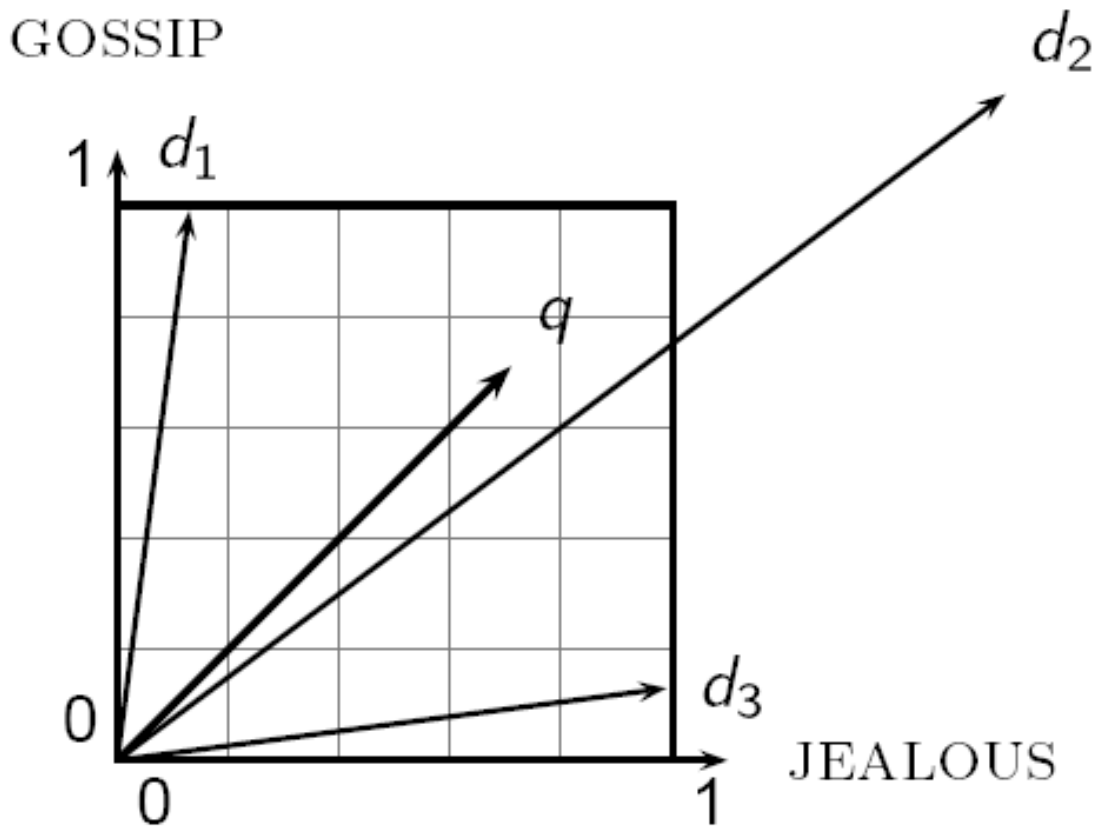
- Key idea 1: Do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity ≈ inverse of distance
- Recall: We do this because we want to get away from the you're-either-in-or-out Boolean model
- Instead: rank more relevant documents higher than less relevant documents.

Formalizing vector space proximity

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

Why distance is a bad idea

The Euclidean distance between q and d_2 is large even though the distribution of terms in the query q and the distribution of terms in the document d_2 are very similar.



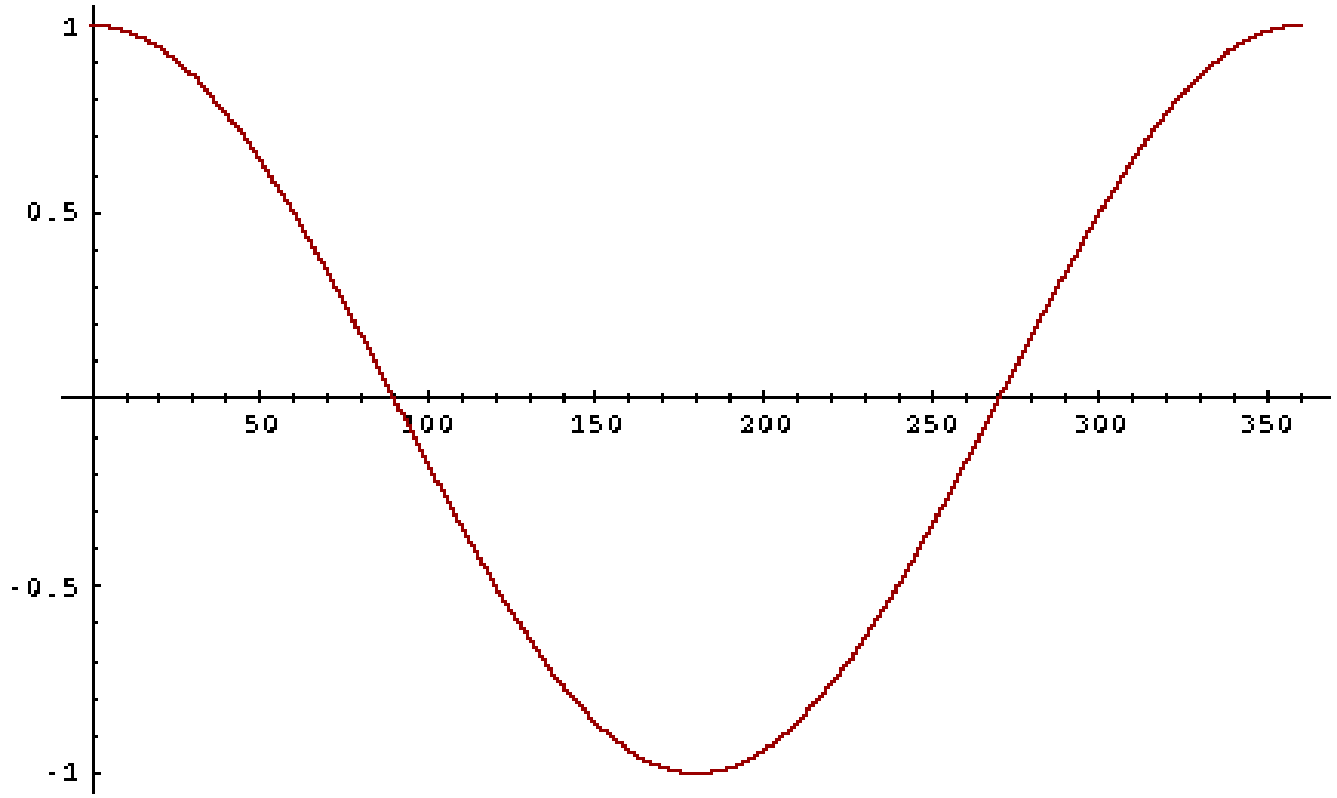
Use angle instead of distance

- Thought experiment: take a document *d* and append it to itself
- Call this document *d′*
- "Semantically" d and d′ have the same content
- The Euclidean distance between the two documents can be quite large
- If the $tf_{t,d}$ representation is used then the angle between the two documents is 0, corresponding to maximal similarity
- Key idea: Rank documents according to angle with query.

From angles to cosines

- The following two notions are equivalent:
  - Rank documents in decreasing order of the angle between query and document
  - Rank documents in increasing order of cosine(query,document)
- Cosine is a monotonically decreasing function for the interval [0^o, 180^o]

From angles to cosines



- But how – *and why* – should we be computing cosines?

Length normalization

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm:

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

- Dividing a vector by its L_2 norm makes it a unit (length) vector (on surface of unit hypersphere)
- Effect on the two documents d and d′ (d appended to itself) from earlier slide: they have identical vectors after length-normalization
  - ✓ Long and short documents now have comparable weights.

cosine(query,document)

Dot product

Unit vectors

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

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

$\cos(\vec{q},\vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$ … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

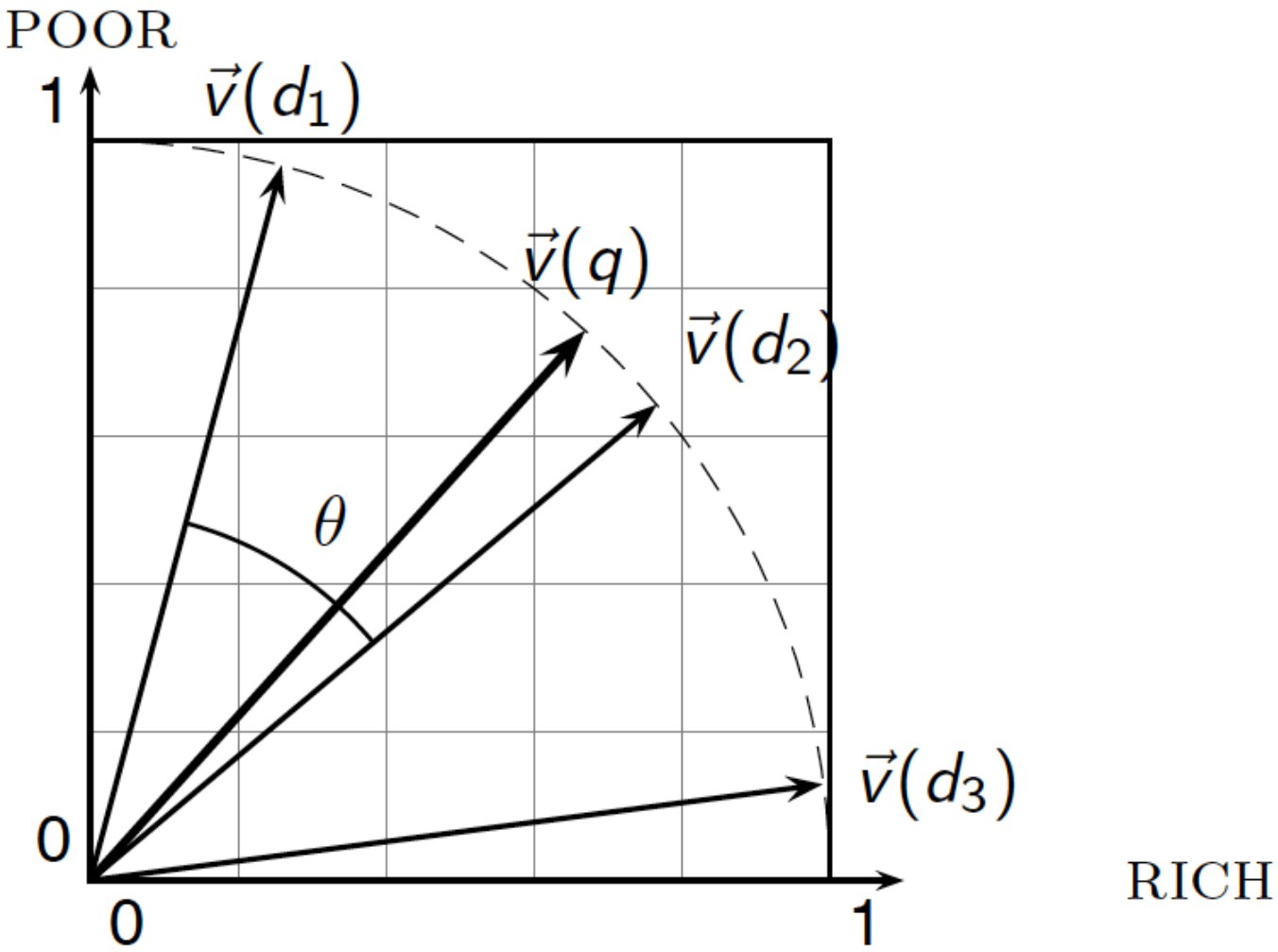
Cosine for length-normalized vectors

- For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):

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

for q, d length-normalized.

Cosine similarity illustrated



Cosine similarity amongst 3 documents

How similar are the novels

SaS: *Sense and Sensibility*

PaP: *Pride and Prejudice*, and

WH: *Wuthering Heights*?

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Term frequencies (counts)

Note: To simplify this example, we don't do idf weighting.

3 documents example cont'd.

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

cos(SaS,PaP) ≈

$0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0 \approx 0.94$

cos(SaS,WH) ≈ 0.79

cos(PaP,WH) ≈ 0.69

Why do we have cos(SaS,PaP) > cos(SaS,WH)?

Computing cosine scores

```
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} × 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[]
```

# of documents

This array contains the vector lengths of all the documents

tf-idf weighting has many variants

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)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

Why is the base of the log in idf immaterial?

Weighting may differ in queries vs documents

- Many search engines allow for different weightings for queries vs. documents
- SMART Notation: denotes the combination in use in an engine, with the notation *ddd.qqq,* using the acronyms from the previous table
- A very standard weighting scheme is: lnc.ltc
- Document: logarithmic tf (l as first character), no idf, and cosine normalization

A bad idea?

- Query: logarithmic tf (l in leftmost column), idf (t in second column), cosine normalization …

tf-idf example: lnc.ltc

ddd.qqq

Document: *car insurance auto insurance*

Query: *best car insurance*

Term	Query						Document				Prod
	tf-raw	tf-wt 1+log tf	df	idf	tf-idf	norm alized	tf-raw	tf-wt 1+ log tf	wt	norm alized	
auto	0	0	5000	2.3	0	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0.34	0	0	0	0	0
car	1	1	10000	2.0	2.0	0.52	1	1	1	0.52	0.27
insurance	1	1	1000	3.0	3.0	0.78	2	1.3	1.3	0.68	0.53

Exercise: We assume *N= 1.000.000* (number of docs)

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

Query length = $\sqrt{0^2+1.3^2+2^2+3^2} \approx 3.83$

Score = 0+0+0.27+0.53 = 0.8

Summary – vector space ranking

- Represents the query as a weighted tf-idf vector
- Represents each document as a weighted tf-idf vector
- Computes the cosine similarity score for the query vector and each document vector
- Ranks documents with respect to the query by score
- Returns the top K (e.g., $K = 10$) to the user.

Intelligent Information Retrieval

Integrazione di conoscenza lessicale: WordNet

WordNet

- Ontologia linguistica che rappresenta in maniera esplicita e formale la conoscenza linguistica umana
- L'idea nasce nel 1985 da un gruppo di linguisti e psicolinguisti dell'università di Princeton
  - ✓ Obiettivo: ricerca concettuale nei dizionari
  - ✓ Risultato: definizione di un database lessicale
  - ✓ Linea di ricerca: memoria lessicale umana
- URL: http://wordnet.princeton.edu/wordnet/about-wordnet/

WordNet

- WordNet è un'ontologia linguistica top-level
- La conoscenza linguistica :
  - ✓ è conoscenza di senso comune
  - ✓ può essere utilizzata in qualsiasi dominio

Utilizzo di WordNet

- Sistemi per Information Retrieval e Text Categorization utilizzano la conoscenza linguistica di WordNet per aggiungere “semantica” al processo di ritrovamento/categorizzazione
  - ✓ Algoritmi di base per l’indicizzazione
  - ✓ Algoritmi avanzati di *word sense disambiguation*

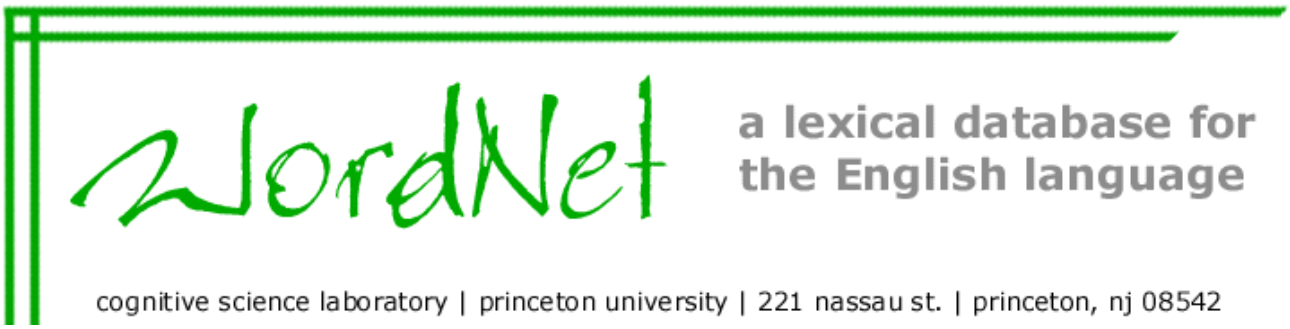
Le quattro categorie lessicali

- La memoria lessicale umana si suddivide in quattro parti ognuna rispettivamente dedicata a: nomi, verbi, aggettivi e avverbi
- Gli ideatori di WordNet, ispirandosi a tale teoria, hanno suddiviso in modo analogo la conoscenza lessicale

Concetto di parola

- PAROLA: un'associazione fra una word form e una word meaning
  - ✓ *word form*: espressione fisica della parola ovvero l'insieme di lettere che la costituisce (stringa)
  - ✓ *word meaning*: concetto lessicale che la word form vuole esprimere ovvero il suo significato sottinteso

WordNet: la matrice lessicale



Word Meanings	***Word Forms***					
	F_1	F_2	F_3	…	…	F_n
M_1	V(1,1)	V(2,1)				
M_2		V(2,2)	V(3,2)			
M_3						
$M_{…}$						
M_m						V(m,n)

Realizza il mapping tra word form e word meaning

Polysemy & Synonymy

- Una word form è polysemous se ad essa possono essere associate più word meaning
- Due word form sono synonym se ad esse è associata la stessa word meaning

Rappresentazione della conoscenza linguistica

- Lo scopo principale di WordNet è quello di riuscire a trasferire ad un computer tutta la conoscenza linguistica
  - le word form, le word meaning e il mapping fra queste due categorie
- La rappresentazione delle word form, in una forma comprensibile ad un calcolatore, non ha suscitato molti problemi

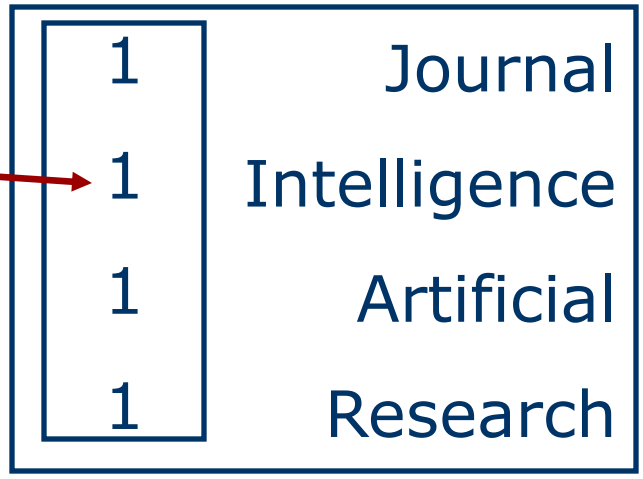
Rappresentazione della conoscenza linguistica

- Ogni word meaning è rappresentata dall'insieme delle word form che possono essere usate per esprimerla: *synset*
- Un synset associato ad una word form consente all'utente di inferire la semantica della word form in esame purché conosca la semantica di almeno una word form elencata nel synset

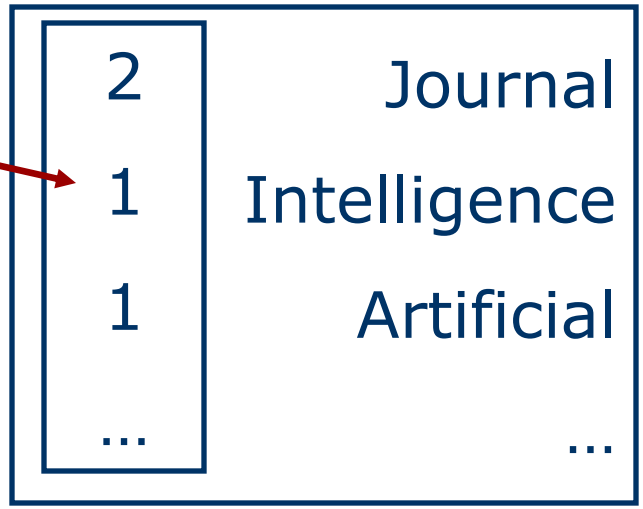
Document representation

Journal of Artificial Intelligence Research

JAIR is a referred journal, covering all areas of Artificial Intelligence, which is distributed free of charge over the Internet. Each volume of the journal is also published by Morgan Kaufman…



Slot "title"



Slot "abstract"

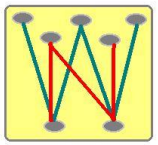
Extended Document Representation

Journal of Artificial Intelligence Research

JAIR is a referred journal, covering all areas of Artificial Intelligence, which is distributed free of charge over the Internet. Each volume of the journal is also published by Morgan Kaufman…

Artificial Intelligence

Slot "title"



WordNet

The Lexical Matrix

WordNet a lexical database for the English language

cognitive science laboratory | princeton university | 221 nassau st. | princeton, nj 08542

Synonym word forms: SYNSET

Word Meanings	*Word Forms*					
	F_1	F_2	F_3	…	…	F_n
M_1	E(1,1)	E(2,1)				
M_2		E(2,2)	E(3,2)			
M_3						
$M_{…}$						
M_m						E(m,n)

Mapping between word forms e word meanings

The Lexical Matrix

the word form is polysemous: WSD needed

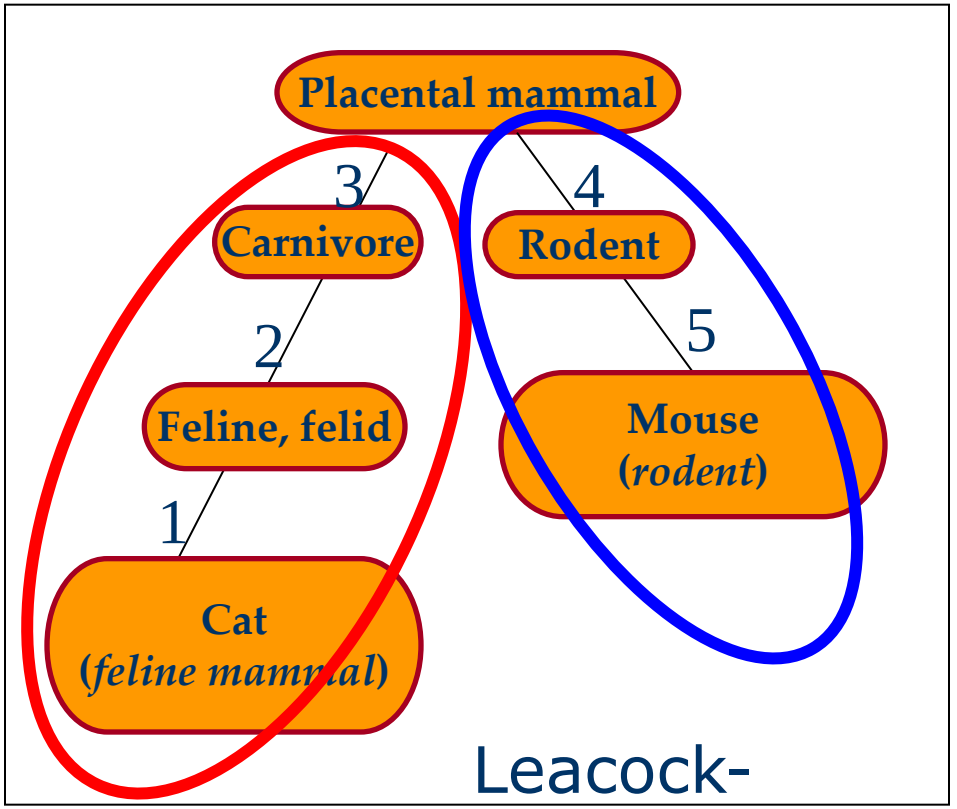
Word Meanings	***Word Forms***					
	F_1	F_2	F_3	…	…	F_n
M_1	E(1,1)	E(2,1)				
M_2		E(2,2)	E(3,2)			
M_3						
$M_{…}$						
M_m						E(m,n)

Mapping between word forms e word meanings

Synset Semantic Similarity

24: **function** SINSIM(a, b) ▷ The similarity of the synsets a and b
25: $N_p \leftarrow$ the number of nodes in path p from a to b
26: $D \leftarrow$ maximum depth of the taxonomy ▷ In WordNet 1.7.1 $D = 16$
27: $r \leftarrow -log(N_p/2D)$
28: **return** r
29: **end function**

SINSIM(cat,mouse) =
-log(5/32)=0.806



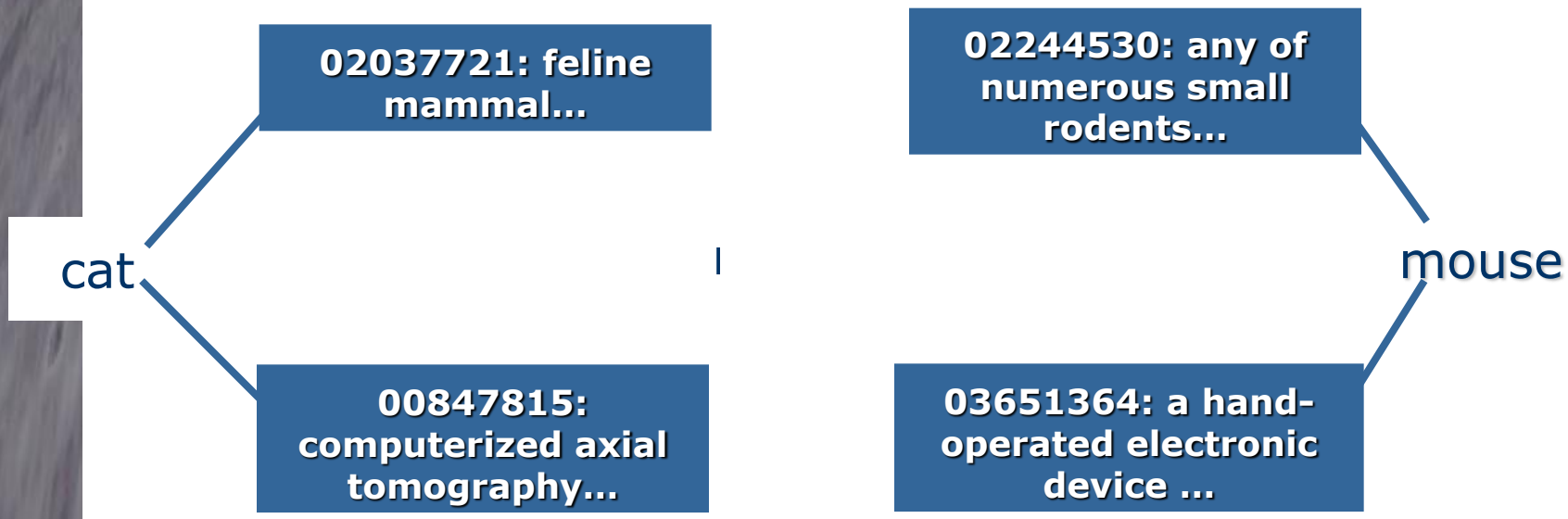
Leacock-Chodorow

Cat-Mouse Disambiguation

"The white cat is hunting the mouse"

w = cat

C = {mouse}

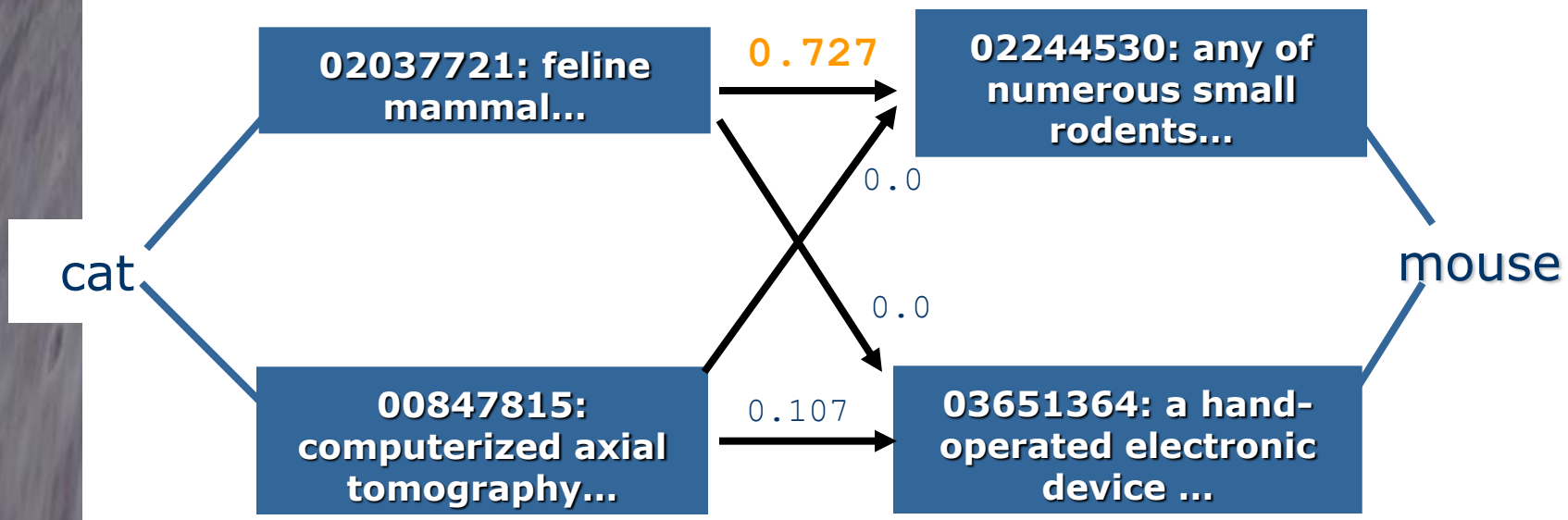


Cat-Mouse Disambiguation

"The white cat is hunting the mouse"

w = cat

C = {mouse}



References

- Baeza-Yates, R.A., Ribeiro-Neto, B.A., "Modern Information Retrieval", ACM Press/Addison-Wesley, 1999.
- R. Feldman and I. Dagan, Knowledge Discovery in Textual Databases (KDT). *Proc. of the 1st Int. Conf. on Knowledge Discovery (KDD-95)*, pp. 112-117, Montreal, 1995.
- Frakes, W., Stemming algorithms, in Frakes, W. & Baeza-Yates, *Information Retrieval Data Structures and Algorithms*, Prentice-Hall, 1992.
- M. Grobelnik, D. Mladenic, and N. Milic-Frayling, *Text Mining as Integration of Several Related Research Areas: Report on KDD'2000 Workshop on Text Mining*, 2000.
- M. Pazzani, *Machine Learning and Information Filtering on the Internet*, IJCAI-97 Tutorial, Nagoya, Japan, Aug 1997.
- Porter, M., An algorithm for suffix stripping, *Program*, 14(3):130-137, 1980.
- Salton, G., & McGill, M. J. "Introduction to Modern Information Retrieval", McGraw-Hill, 1983.
- Salton, G., & Buckley, C., Term weighting approaches in automatic text retrieval, *Information Processing and Management*, 24(5):513-523, 1988.
- Salton, G., & Buckley, C., Improving retrieval performance by relevance feedback, *Journal of the American Society for Information Science*, 41:288-297, 1990.
- vanRijsbergen, C.J., "Information Retrieval", Butterworth & Co., Boston, MA, 1979.