CS3245

Information Retrieval

10

Lecture 10:
Relevance Feedback and Query Expansion, and XML IR

Last Time

Search engine evaluation

- Benchmark
  - Measures: Precision / Recall / F-measure, Precision-recall graph and single number summaries
  - Documents, queries and relevance judgments
    - Kappa Measure
- A/B Testing
  - Overall evaluation criterion (OEC)

Today

Chapter 9

1. Relevance Feedback

   Document Level

   - Explicit RF – Rocchio (1971)
     - When does it work?
   - Variants: Implicit and Blind

2. Query Expansion

   Term Level

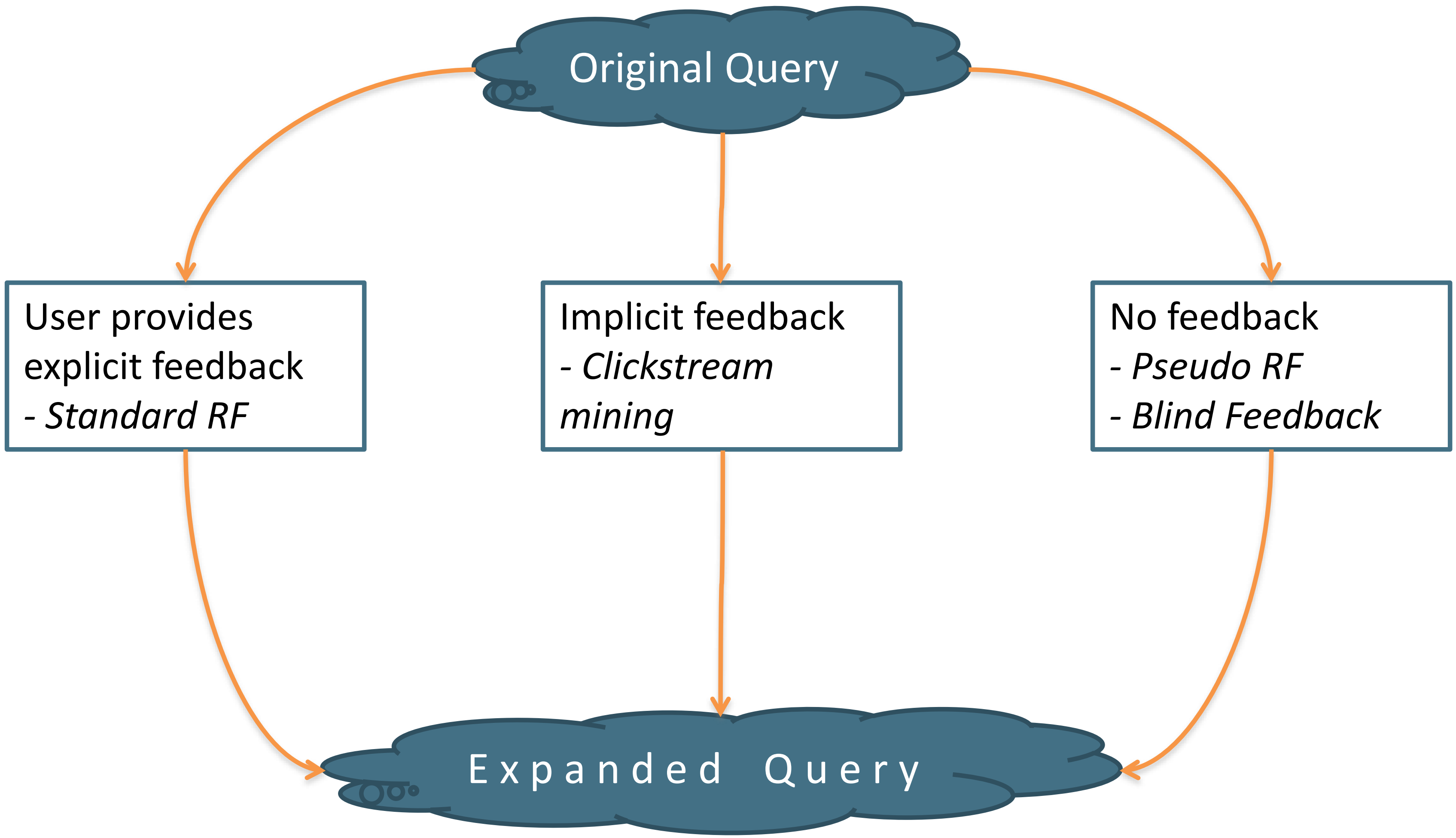
   - Manual thesaurus
   - Automatic Thesaurus Generation

Chapter 10

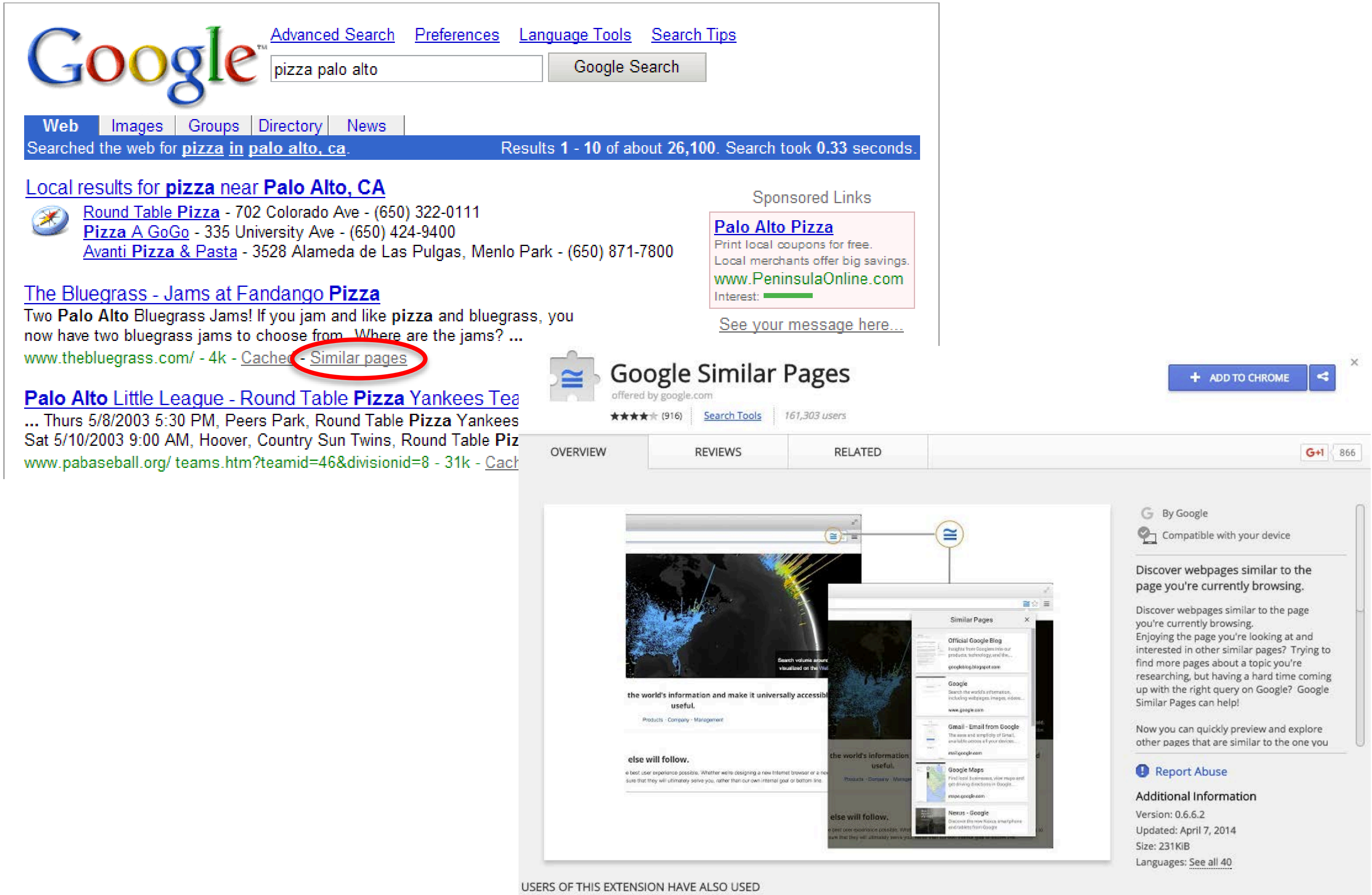
3. XML IR
   - Basic XML concepts
   - Challenges in XML IR
   - Vector space model for XML IR
   - Evaluation of XML IR

RELEVANCE FEEDBACK

Relevance Feedback

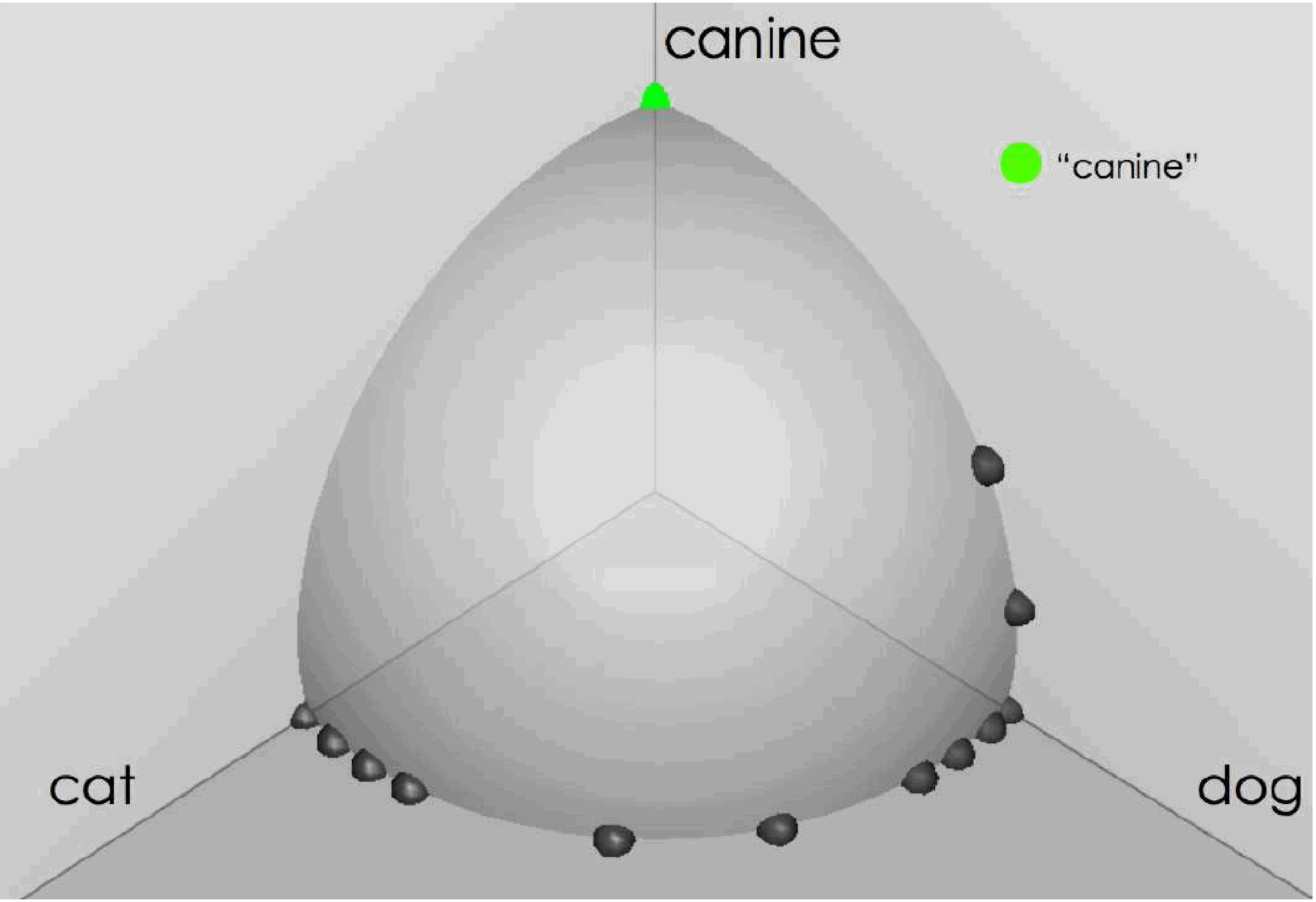


Explicit Feedback



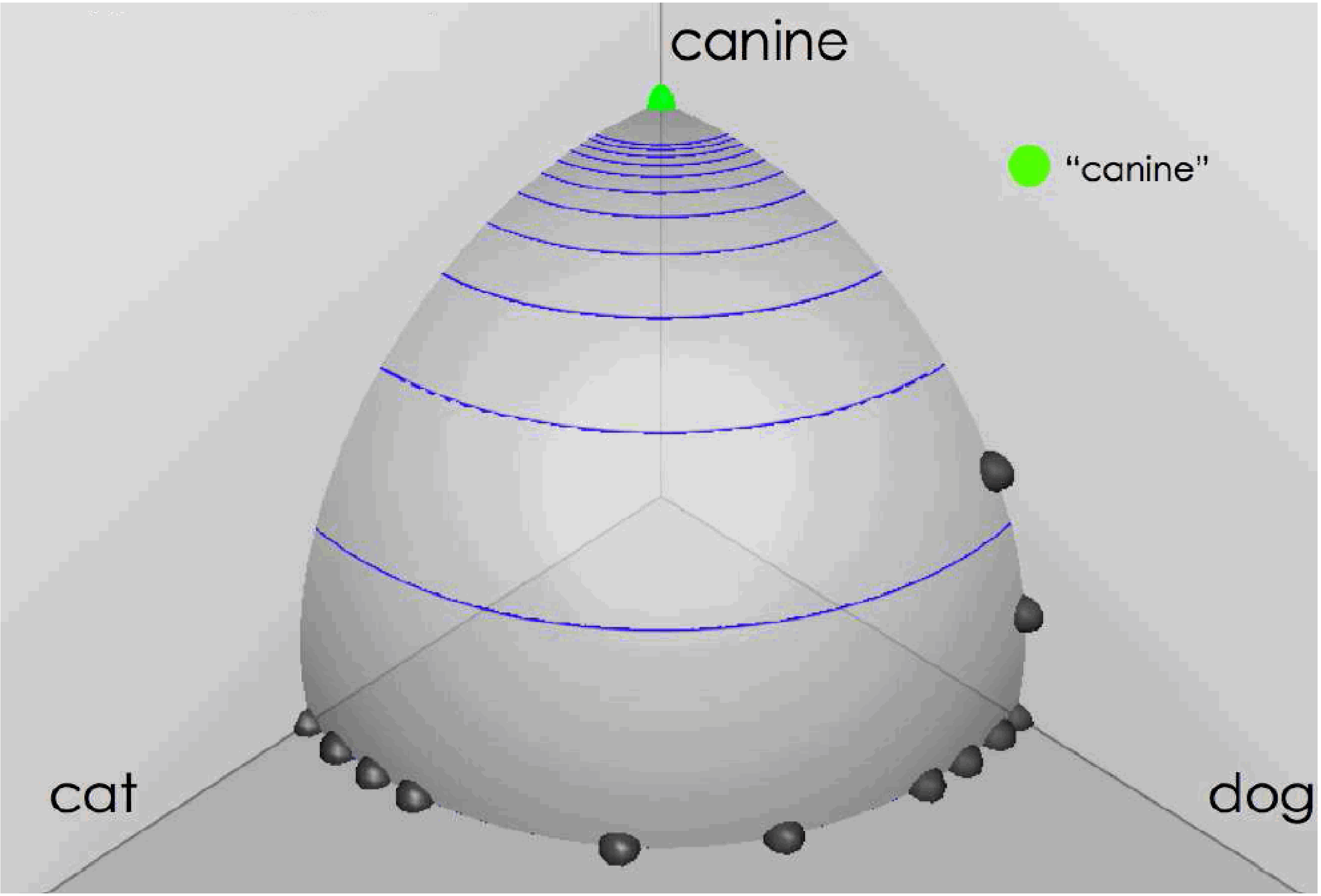
Initial results for query *canine*

source: Fernando Diaz



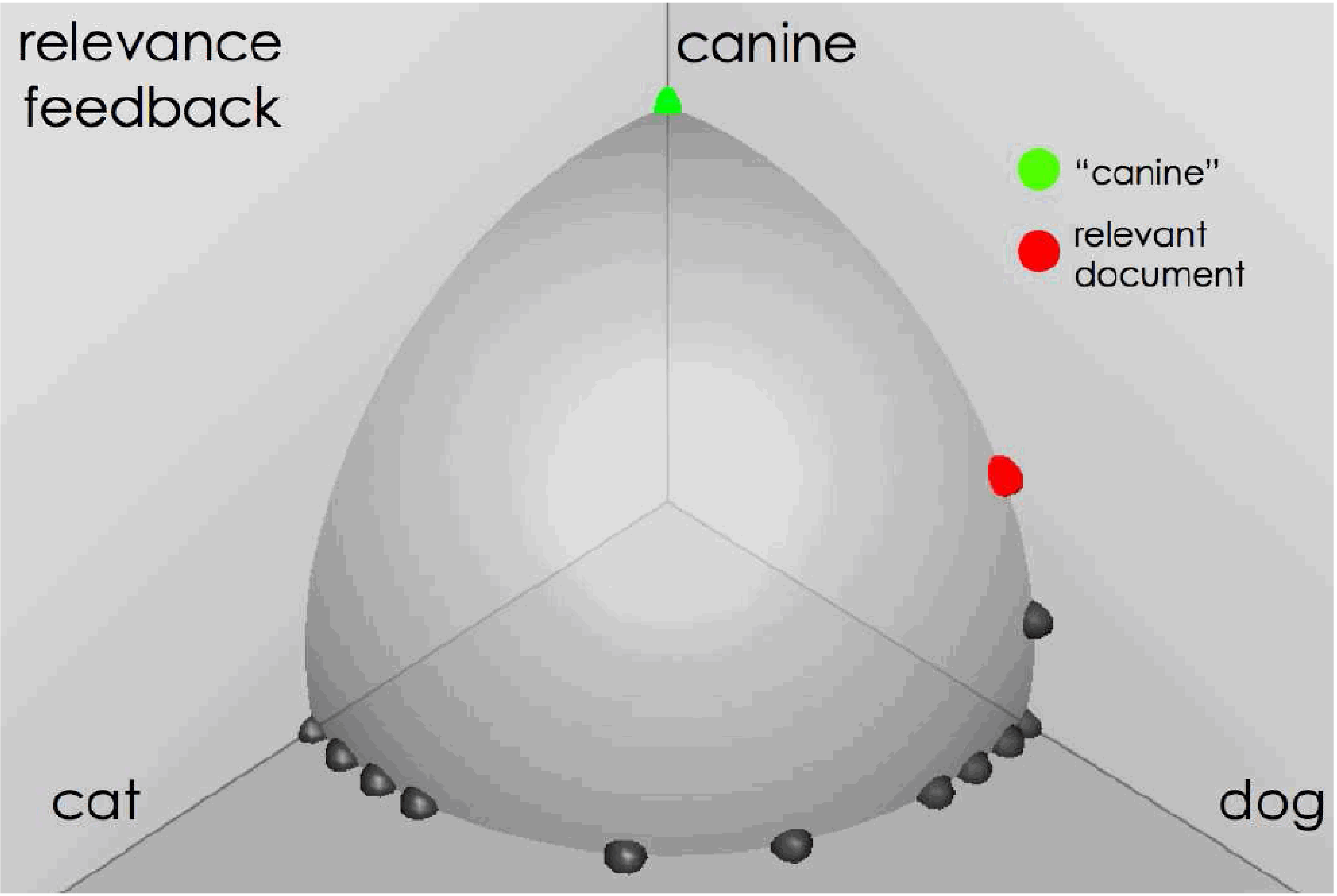
Initial results for query *canine*

source: Fernando Diaz



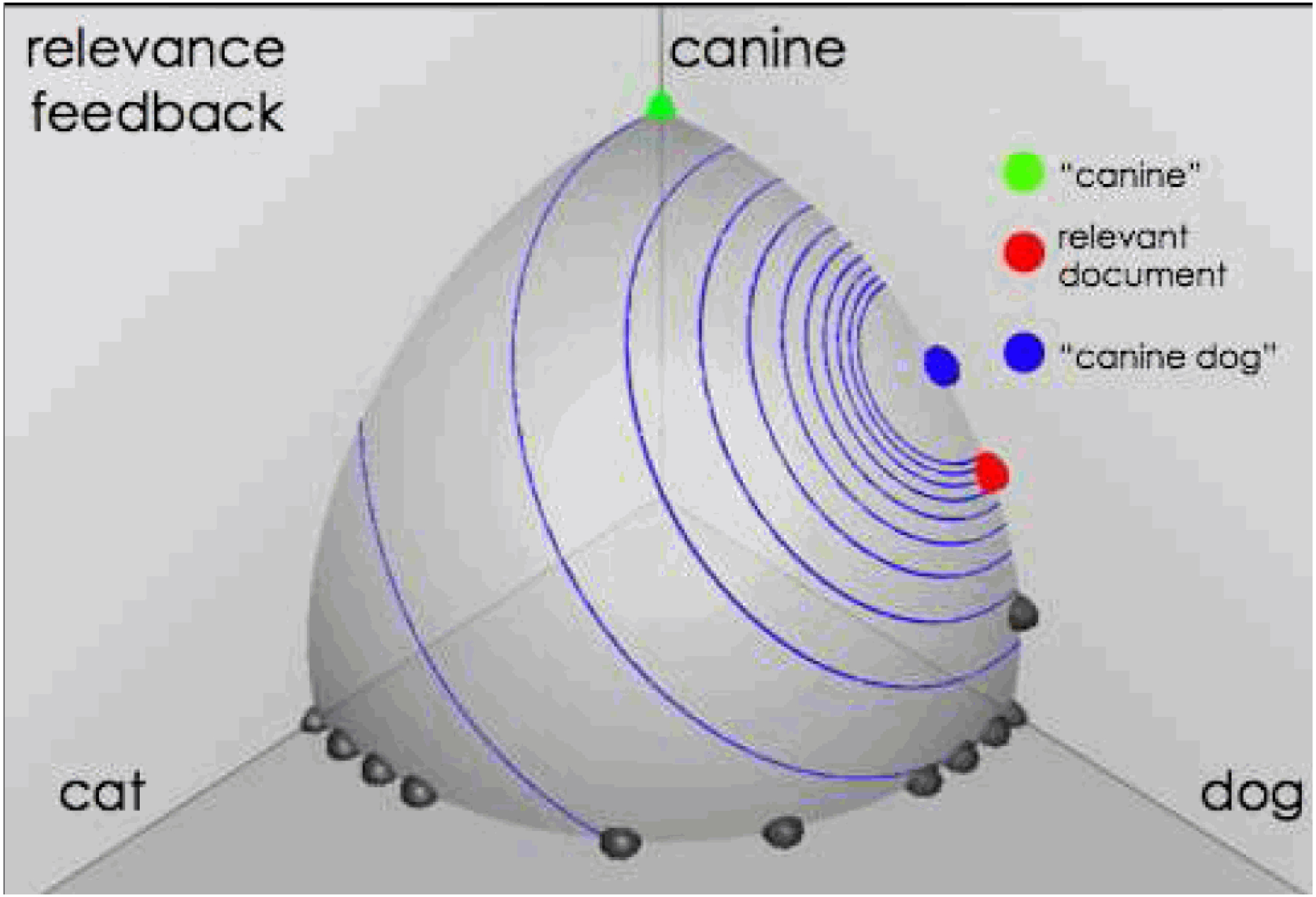
User feedback: Select what is relevant

source: Fernando Diaz



Results after relevance feedback

source: Fernando Diaz



Initial query/results

Initial query: *New space satellite applications*

User marks relevant items

\+ 1. 0.539, 08/13/91, NASA Hasn't Scrapped Imaging Spectrometer
\+ 2. 0.533, 07/09/91, NASA Scratches Environment Gear From Satellite Plan
– 3. 0.528, 04/04/90, Science Panel Backs NASA Satellite Plan, But Urges Launches of Smaller Probes
– 4. 0.526, 09/09/91, A NASA Satellite Project Accomplishes Incredible Feat: Staying Within Budget
– 5. 0.525, 07/24/90, Scientist Who Exposed Global Warming Proposes Satellites for Climate
– 6. 0.524, 08/22/90, Report Provides Support for the Critics Of Using Big Satellites to Study Climate
– 7. 0.516, 04/13/87, Arianespace Receives Satellite Launch Pact From Telesat Canada
\+ 8. 0.509, 12/02/87, Telecommunications Tale of Two Companies

Assume others as nonrelevant

Expanded query after relevance feedback

2.074 new	15.10 space
30.81 satellite	5.660 application
5.991 nasa	5.196 eos
4.196 launch	3.972 aster
3.516 instrument	3.446 arianespace
3.004 bundespost	2.806 ss
2.790 rocket	2.053 scientist
2.003 broadcast	1.172 earth
0.836 oil	0.646 measure

Results for the expanded query

2 1. 0.513, 07/09/91, NASA Scratches Environment Gear From Satellite Plan
1 2. 0.500, 08/13/91, NASA Hasn't Scrapped Imaging Spectrometer
3. 0.493, 08/07/89, When the Pentagon Launches a Secret Satellite, Space Sleuths Do Some Spy Work of Their Own
4. 0.493, 07/31/89, NASA Uses 'Warm' Superconductors For Fast Circuit
8 5. 0.492, 12/02/87, Telecommunications Tale of Two Companies
6. 0.491, 07/09/91, Soviets May Adapt Parts of SS-20 Missile For Commercial Use
7. 0.490, 07/12/88, Gaping Gap: Pentagon Lags in Race To Match the Soviets In Rocket Launchers
8. 0.490, 06/14/90, Rescue of Satellite By Space Agency To Cost $90 Million

Originally Marked Relevant Documents

Key concept: Centroid

- The **centroid** is the center of mass of a set of points.

Definition: Centroid $$\vec{\mu}(D) = \frac{1}{|D|} \sum_{d \in D} \vec{d}$$

Where D is a set of documents.

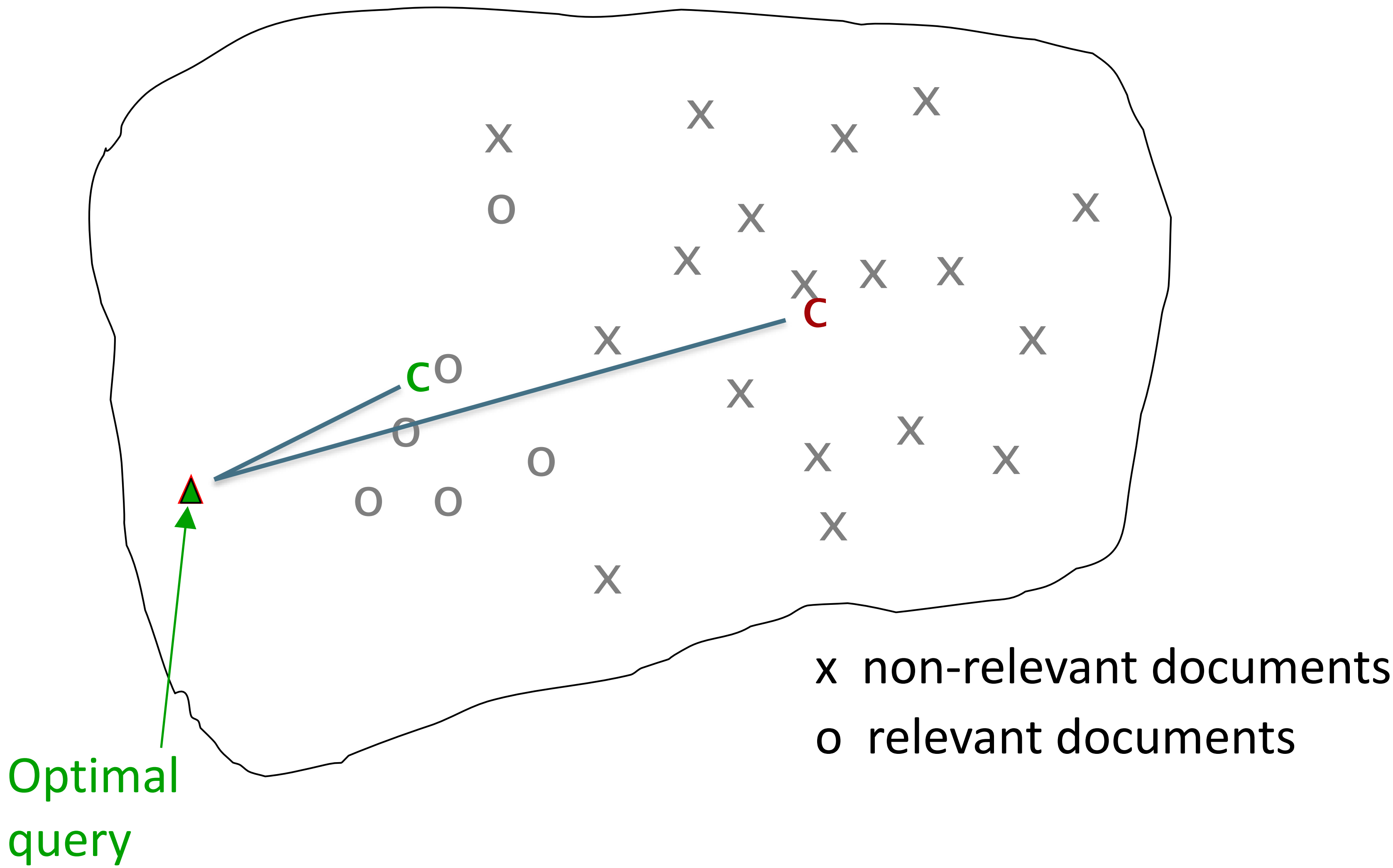
Rocchio Algorithm

- Intuitively, we want to separate docs marked as relevant and non-relevant from each other

- The Rocchio algorithm uses the vector space model to pick a new query

$$\vec{q}_{opt} = \arg\max_{\vec{q}} [\cos(\vec{q}, \vec{\mu}(C_r)) - \cos(\vec{q}, \vec{\mu}(C_{nr}))]$$

The Theoretically Best Query



Rocchio (1971)

Popularized in the SMART system (Salton)

In practice:

$$\vec{q}_m = \alpha \vec{q}_0 + \beta \frac{1}{|D_r|} \sum_{\vec{d}_j \in D_r} \vec{d}_j - \gamma \frac{1}{|D_{nr}|} \sum_{\vec{d}_j \in D_{nr}} \vec{d}_j$$

- D_r = set of <u>known</u> relevant doc vectors
- D_{nr} = set of <u>known</u> irrelevant doc vectors
  - ! Different from C_r and C_{nr} as we only get judgments from a few documents
- $\{\alpha, \beta, \gamma\}$ = weights (hand-chosen or set empirically)

Weighting

$$\vec{q}_m = \alpha \vec{q}_0 + \beta \frac{1}{|D_r|} \sum_{\vec{d}_j \in D_r} \vec{d}_j - \gamma \frac{1}{|D_{nr}|} \sum_{\vec{d}_j \in D_{nr}} \vec{d}_j$$

- Tradeoff α vs. β/γ : What if we have only a few judged documents?
- β vs. γ: Which is more valuable?
  - Many systems only allow positive feedback (γ =0). Why?
- Some weights in the query vector can go negative
  - So negative term weights are ignored (set to 0)

Evaluation of relevance feedback strategies

Use q_m and compute precision recall graph

1. Assess on all documents in the collection
   - Spectacular improvements, but … it's cheating!
   - Must evaluate with respect to documents not seen by user
2. Use documents in residual collection (set of documents minus those assessed relevant)
   - Measures usually then lower than for original query
   - But a more realistic evaluation
   - Relative performance can be validly compared

- Best: use two collections each with their own relevance assessments
  - q_o and user feedback from first collection
  - q_m run on second collection and measured

When does RF work?

Empirically, a round of RF is often very useful. Two rounds is sometimes marginally useful.

When does it work? When two assumptions hold:

1. User's initial query at least partially works.

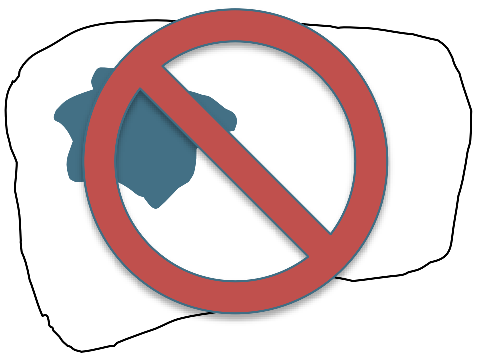
2. (Non)-relevant documents are similar.

   or term distribution in non-relevant documents are sufficiently distinct from relevant documents

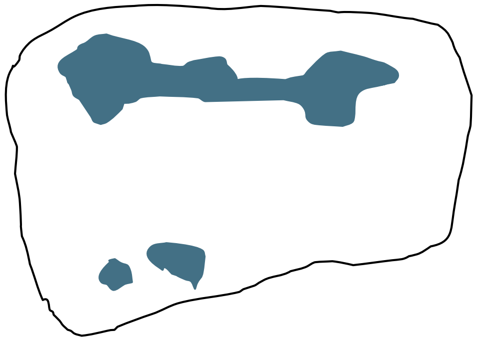
Violation of Assumption 1

- User does not have sufficient initial knowledge.
- Examples:
  - Misspellings (but not *Brittany Speers*).
  - Mismatch of searcher's vocabulary vs. collection vocabulary
    - Q: "laptop" but collection all uses "notebook"
  - Cross-language information retrieval (hígado).

Violation of Assumption 2



- There are several relevance prototypes.
- Examples:
  - Burma/Myanmar: change of name
  - Instances of a general concept
  - Pop stars that worked at Burger King



Relevance Feedback: Problems

- Long queries are inefficient for typical IR engine.
  - Long response times for user, as it deals with long queries.
  - Hack: reweight only a # of prominent terms, e.g., top 20.
- Users reluctant to provide explicit feedback
- Harder to understand why particular document was retrieved after RF

RF in Web search

- True evaluation of RF must also account for usability and time.
- Alternative: User revises and resubmits query.
- Users may prefer revision/resubmission to having to judge relevance of documents (more transparent)

- Some search engines offer a similar/related pages
  - Google (link-based), Altavista, Stanford WebBase
- Some don't use RF because it's hard to explain:
  - Alltheweb, Bing, Yahoo!
- Excite initially had true RF, but abandoned it due to lack of use.

Pseudo relevance feedback (PRF)

- Blind feedback automates the “manual” part of true RF, by assuming the top *k* is actually relevant.

- Algorithm:
  - Retrieve a ranked list of hits for the user’s query
  - Assume that the top *k* documents are relevant.
  - Do relevance feedback

- Works very well on average
- But can go horribly wrong for some queries
- Several iterations can cause query drift

QUERY EXPANSION

Relevance Feedback vs Query Expansion

- In relevance feedback, additional input (relevant/non-relevant) is given on documents, which is used to reweight terms in the documents
- In query expansion, additional input (good/bad search term) is given on words or phrases

How do we augment the user query?

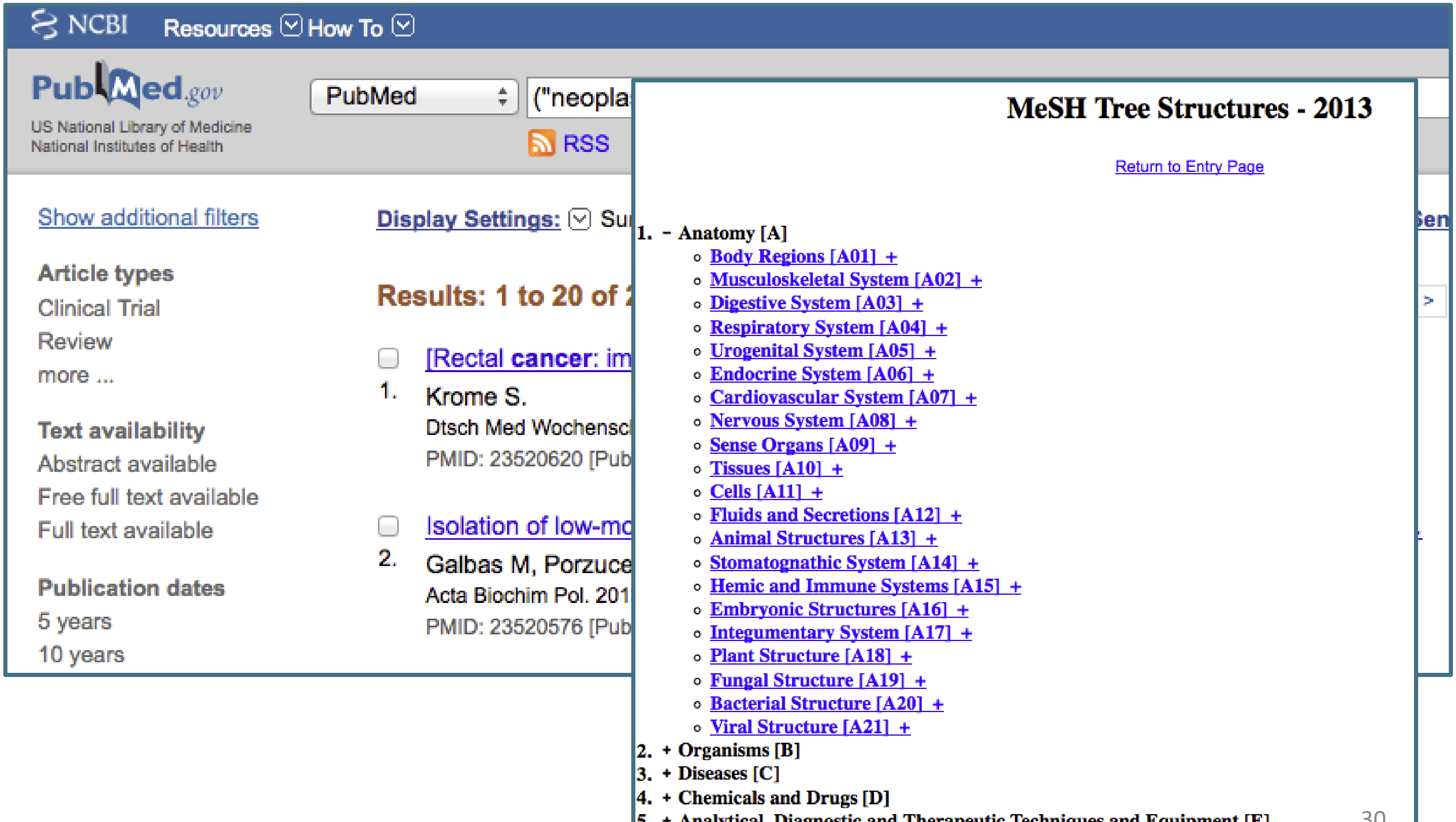
- Manual thesaurus
  - E.g. MedLine: physician, syn: doc, doctor, MD, medico
  - Can be query rather than just synonyms
- Global analysis
  - Automatic Thesaurus Generation
  - Refinements based on query log mining

Thesaurus-based query expansion

- For each term, *t*, in a query, expand the query with synonyms and related words of *t* from the thesaurus
  - feline → feline cat

- Generally increases recall, but may decrease precision when terms are ambiguous.
  E.g., “interest rate” → “interest rate fascinate evaluate”

An example of thesaurii: MeSH



Princeton's WordNet

WordNet Search - 3.1

- WordNet home page - Glossary - Help

Word to search for: washing machine [Search WordNet]

Display Options: (Select option to change) [Change]

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations

Display options for sense: (gloss) "an example sentence"

Noun

- S: (n) washer, automatic washer, **washing machine** (a home appliance for washing clothes and linens automatically)
  - *direct hypernym* / ***inherited hypernym*** / *sister term*
    - S: (n) white goods (large electrical home appliances (refrigerators or washing machines etc.) that are typically finished in white enamel)
      - S: (n) home appliance, household appliance (an appliance that does a particular job in the home)
        - S: (n) appliance (durable goods for home or office use)
          - S: (n) durables, durable goods, consumer durables (consumer goods that are not destroyed by use)
            - S: (n) consumer goods (goods (as food or clothing) intended for direct use or

```
from nltk.corpus import wordnet as wn

wn.synsets("motorcar")
wn.synsets("car.n.01").lemma_names
```

Automatic Thesaurus Generation

You shall know a word by the company it keeps
– John R. Firth

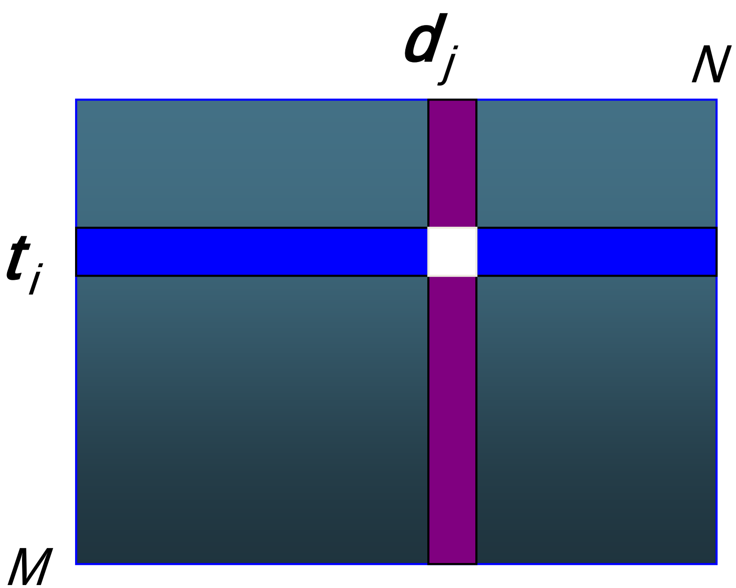
- You can "harvest", "peel", "eat" and "prepare" apples and pears, so apples and pears must be similar
- Generate a thesaurus by analyzing the documents
- Assumption: distributional similarity
- i.e., Two words are similar if they co-occur / share same grammatical relations with similar words.

Co-occurrences are more robust; grammatical relations are more accurate. Why?

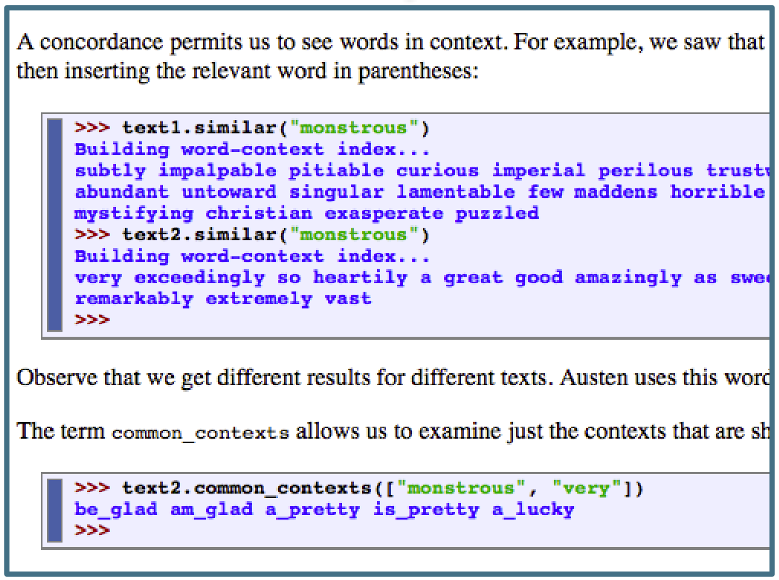
Co-occurrence Thesaurus

Simplest way to compute one is based on term-term similarities in $C = AA^T$ where A is term-document matrix.

- $w_{i,j}$ = (normalized) weight for (t_i,$\boldsymbol{d}_j$)



In NLTK! ☺
Have a look!



A concordance permits us to see words in context. For example, we saw that
then inserting the relevant word in parentheses:

```
>>> text1.similar("monstrous")
Building word-context index...
subtly impalpable pitiable curious imperial perilous trustw
abundant untoward singular lamentable few maddens horrible
mystifying christian exasperate puzzled
>>> text2.similar("monstrous")
Building word-context index...
very exceedingly so heartily a great good amazingly as swee
remarkably extremely vast
>>>
```

Observe that we get different results for different texts. Austen uses this word

The term `common_contexts` allows us to examine just the contexts that are sh

```
>>> text2.common_contexts(["monstrous", "very"])
be_glad am_glad a_pretty is_pretty a_lucky
>>>
```

- For each t_i, pick terms with high values in C

Automatic Thesaurus Generation: Problems

- Term ambiguity may introduce irrelevant statistically correlated terms.
  - "Apple computer" → "Apple red fruit computer"
- Problems:
  - False positives: Words deemed similar that are not (Especially opposites)
  - False negatives: Words deemed dissimilar that are similar
- Since terms are highly correlated anyway, expansion may not retrieve many additional documents.

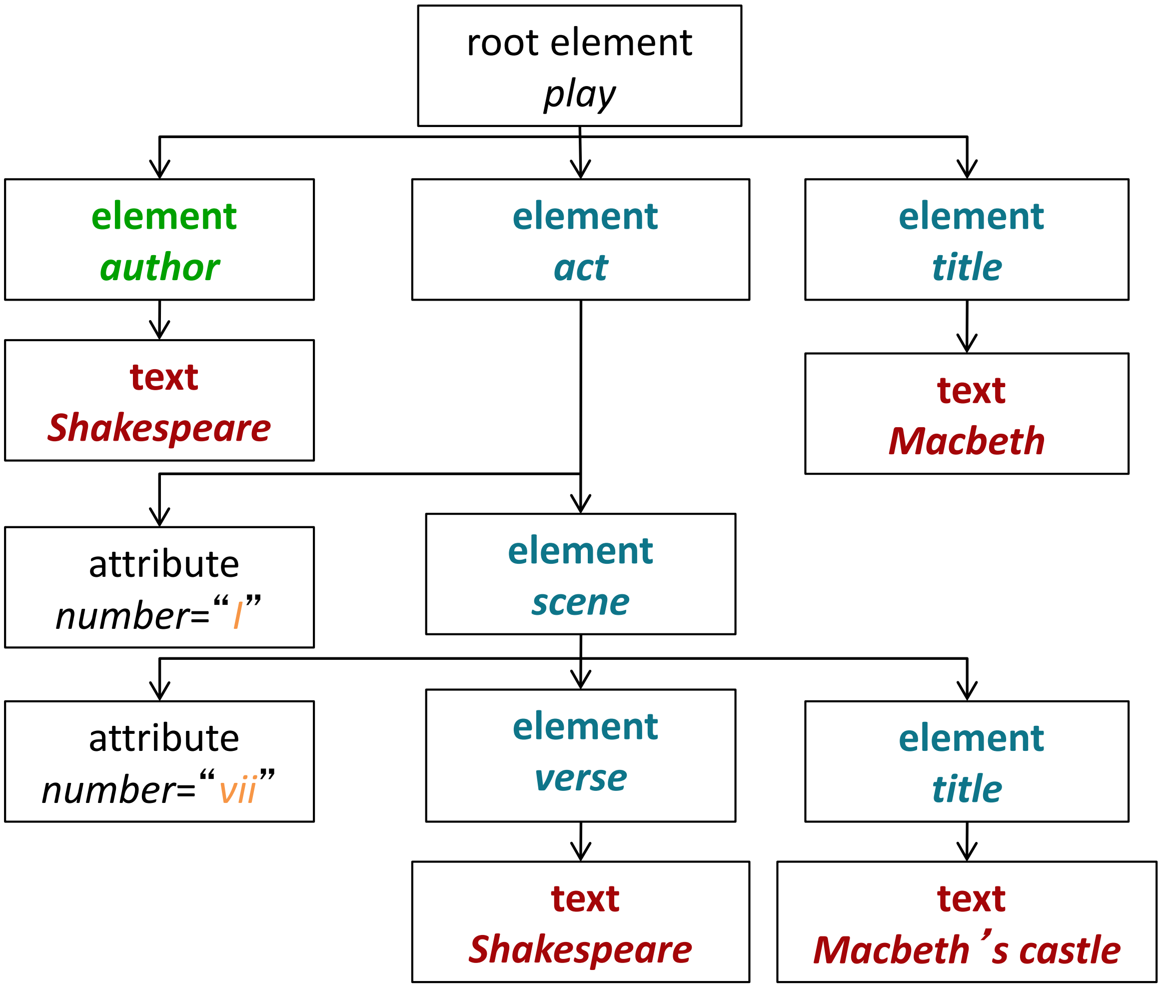
XML RETRIEVAL

XML Document

The **leaf nodes** consist of text

The internal nodes encode **document structure** or **metadata**

An element can have one or more XML **attributes**



Structured Retrieval

Premise: queries are structured or unstructured; documents are structured.

Applications of structured retrieval

Digital libraries, patent databases, blogs, tagged text with entities like persons and locations (named entity tagging)

Example

- Digital libraries: *give me a full-length article on fast fourier transforms*
- Patents: *give me patents whose claims mention RSA public key encryption and that cite US Patent 4,405,829*
- Entity-tagged text: *give me articles about sightseeing tours of the Vatican and the Coliseum*

Structured Retrieval

RDB search, Unstructured IR, Structured IR

	RDB search	unstructured retrieval	structured retrieval
objects	records	unstructured docs	trees with text at leaves
main data structure	table	inverted index	?
model	relational model	vector space & others	?
queries	SQL	free text queries	?

- Standard for encoding structured documents: Extensible Markup Language (XML)
  - structured IR → XML IR
  - also applicable to other types of markup (HTML, SGML, …)

Why RDB is not suitable in this case

Three main problems

1. An unranked system (like a DB) can return a large set leading to information overload
2. Users often don't precisely state structural constraints – may not know possible structure elements are supported
   - *tours* AND (COUNTRY: *Vatican* OR LANDMARK: *Coliseum*)?
   - *tours* AND (STATE: *Vatican* OR BUILDING: *Coliseum*)?
3. Users may be unfamiliar with structured search and the necessary advanced search interfaces or syntax

Solution: adapt ranked retrieval to structured documents

CHALLENGES IN XML RETRIEVAL

First challenge: Document parts to retrieve

- Structured or XML retrieval: users want parts of documents (i.e., XML elements), not the entire thing.

Example

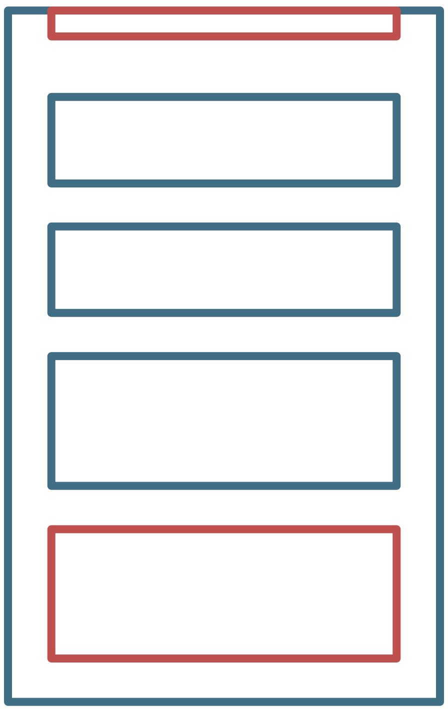
If we query Shakespeare's plays for *Macbeth's castle*, should we return the scene, the act or the entire play?

  - In this case, the user is probably looking for the scene.
  - However, an otherwise unspecified search for *Macbeth* should return the play of this name, not a subunit.
- Solution: structured document retrieval principle

Structured document retrieval principle

Structured document retrieval principle

A system should always retrieve the most specific part of a document that answers the query.



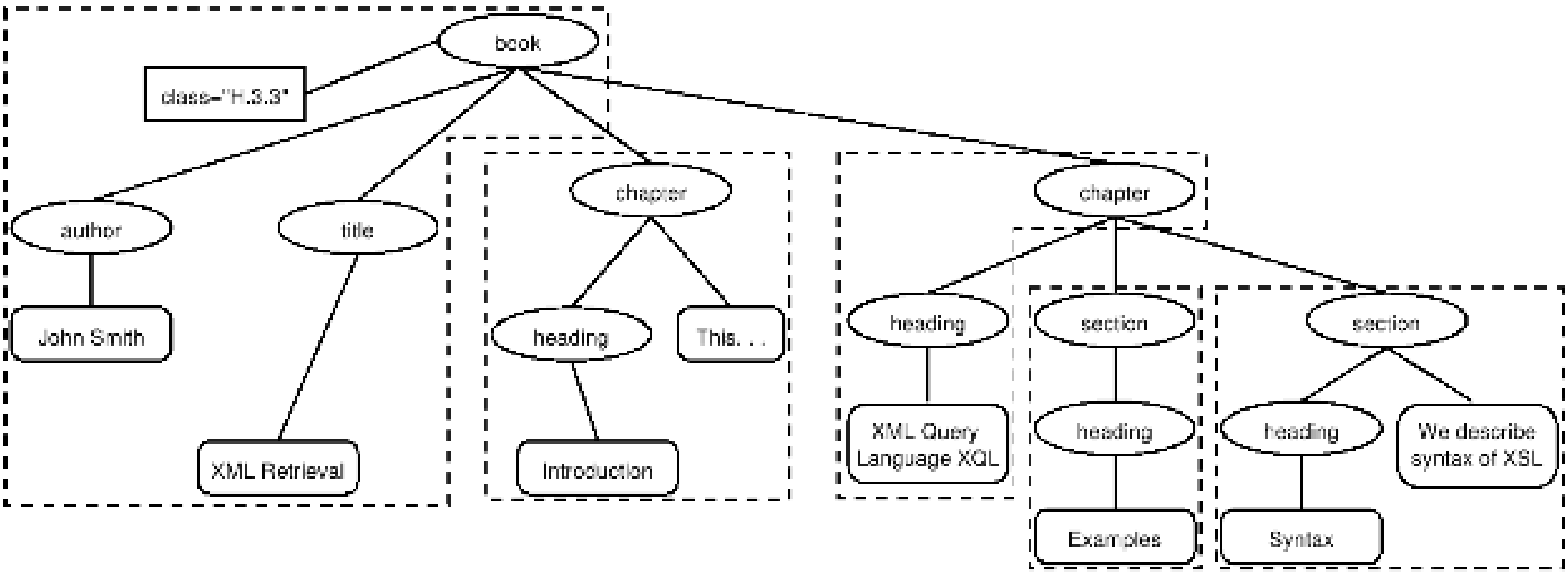
- Hard to implement this principle algorithmically. E.g. query: title:*Macbeth* can match both the title of the tragedy, *Macbeth*, and the title of Act I, Scene vii, *Macbeth's castle*.

Second challenge: Indexing Unit

- In unstructured retrieval, this is usually straightforward: files on your desktop, email messages, web pages, etc.
- In structured retrieval not so obvious what are document boundaries. 4 main methods:
  1. Non-overlapping pseudo-documents
  2. Top down
  3. Bottom up
  4. All units

1) Non-overlapping pseudodocuments

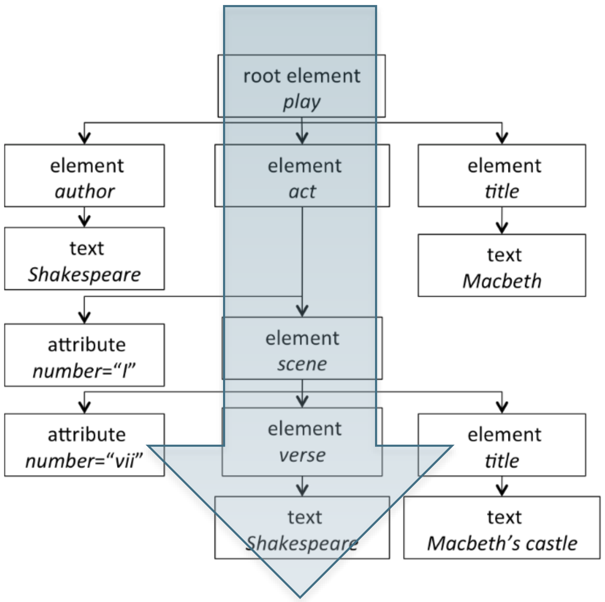
Group nodes into non-overlapping subtrees



- Indexing units: books, chapters, section, but without overlap.
- Disadvantage: pseudodocuments may not make sense to the user because they are not coherent units.

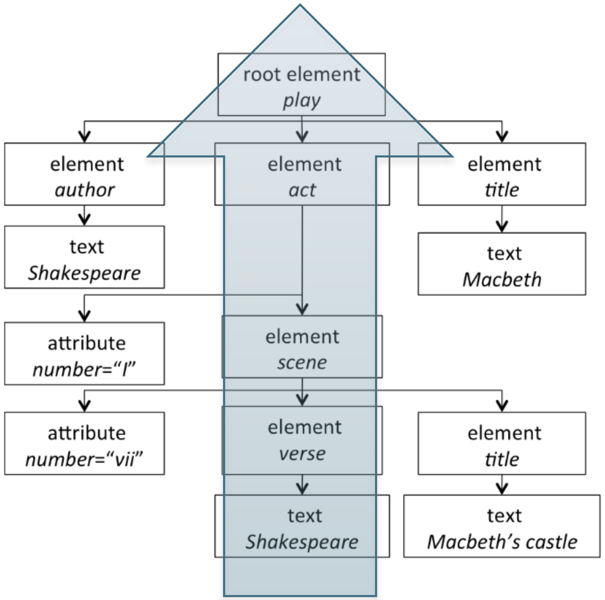
2) Top down

- A 2-stage process:
  1. Start with one of the largest elements as the indexing unit, e.g. the **<book>** element in a collection of books
  2. Then postprocess search results to find for each book the subelement that is the best hit.
- This two-stage process often fails to return the best subelement
  - The relevance of a whole book is often not a good predictor of the relevance of subelements within it.



3) Bottom Up

- We can search all leaves, select the most relevant ones and then extend them to larger units in postprocessing (bottom up).
- Similar problem as *top down*: the relevance of a leaf element is often not a good predictor of the relevance of elements it is contained in.



4) Index all elements

- The least restrictive approach, but also problematic:
- Many XML elements are not meaningful search results, e.g., an ISBN number, bolded text
- Indexing all elements means that search results will be highly redundant, due to **nested elements**.

Example

For the query *Macbeth's castle*, we would return all of the *play*, *act*, *scene* and *title* elements on the path between the root node and *Macbeth's castle*. The leaf node would then occur 4 times in the result set: 1 directly and 3 as part of other elements.

Third challenge: Nested elements

Due to the redundancy of nested elements, it is common to restrict the set of elements eligible for retrieval.

Restriction strategies include:

- discard all small elements
- discard all elements that **users** do not look at (from examining retrieval system logs)
- discard all elements that **assessors** generally do not judge to be relevant (when relevance assessments are available)
- only keep elements that a **system designer or librarian** has deemed to be useful

In most of these approaches, result sets will still contain nested elements.

Third challenge: Nested elements

Further techniques:

- remove nested elements in a postprocessing step to reduce redundancy.
- collapse several nested elements in the results list and use highlighting of query terms to draw the user's attention to the relevant passages.

Highlighting

- Gain 1: enables users to scan medium-sized elements (e.g., a section); thus, if the section and the paragraph both occur in the results list, it is sufficient to show the section.
- Gain 2: paragraphs are presented in-context (i.e., their embedding section). This context may be helpful in interpreting the paragraph.

Nested elements and term statistics

- Further challenge related to nesting: we may need to distinguish different contexts of a term when we compute term statistics for ranking, in particular inverse document frequency (*idf*).

Example

The term *Gates* under the node *author* is unrelated to an occurrence under a content node like *section* if used to refer to the plural of *gate*. It makes little sense to compute a single document frequency for *Gates* in this example.

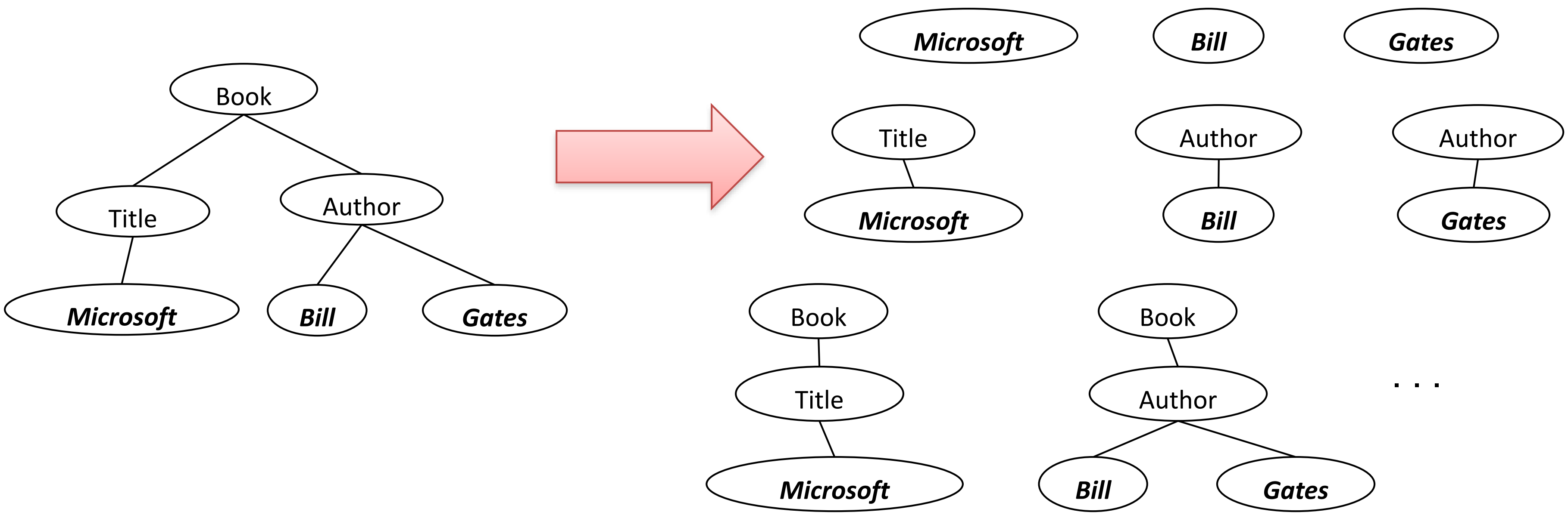
- Solution: compute *idf* for XML-context term pairs.
- Sparse data problems (many XML-context pairs occur too rarely to reliably estimate *df*)
- Compromise: consider the parent node **x** of the term and not the rest of the path from the root to **x** to distinguish contexts.

VECTOR SPACE MODEL FOR XML IR

Main idea: lexicalized subtrees

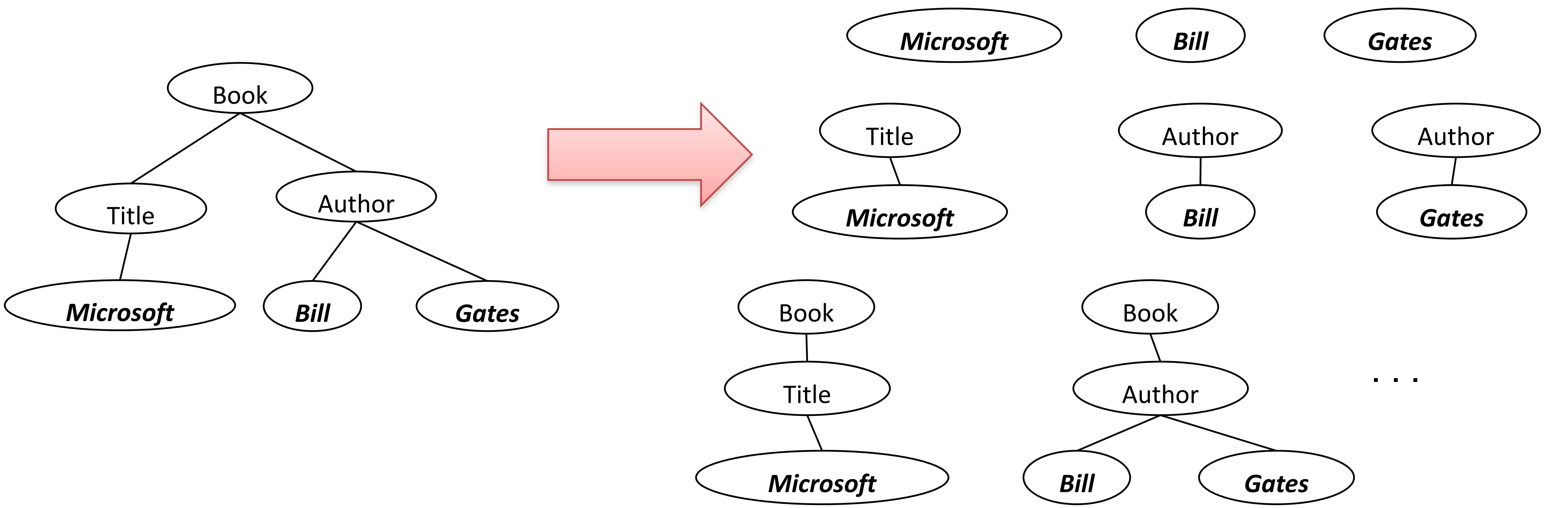
- Aim: to have each dimension of the vector space encode a word together with its position within the XML tree.
- How: Map XML documents to lexicalized subtrees.

"With words"



Creating lexicalized subtrees

- Take each text node (leaf) and break it into multiple nodes, one for each word. E.g. split **Bill Gates** into **Bill** and **Gates**
- Define the dimensions of the vector space to be lexicalized subtrees of documents – subtrees that contain at least one vocabulary term.



Lexicalized subtrees

- We can now represent queries and documents as vectors in this space of lexicalized subtrees and compute matches between them,
- e.g. using the vector space formalism.

Vector space formalism in unstructured vs. structured IR

The **main difference** is that the dimensions of vector space in unstructured retrieval are **vocabulary terms** whereas they are **lexicalized subtrees** in XML retrieval.

Structural term

- There is a tradeoff between the dimensionality of the space and the accuracy of query results.

Feast or Famine

  - If we restrict dimensions to vocabulary terms, then the VSM retrieval system will retrieve many documents that do not match the structure of the query (e.g., Gates in the title as opposed to the author element).
  - If we create a separate dimension for each lexicalized subtree in the collection, the dimensionality becomes too large.
- Compromise: index all paths that end in a **single** vocabulary term (i.e., all XML-context term pairs).
  We call such an XML-context term pair a structural term and denote it by **<c, t>**: a pair of XML-context **c** and vocabulary term **t**.

Context resemblance

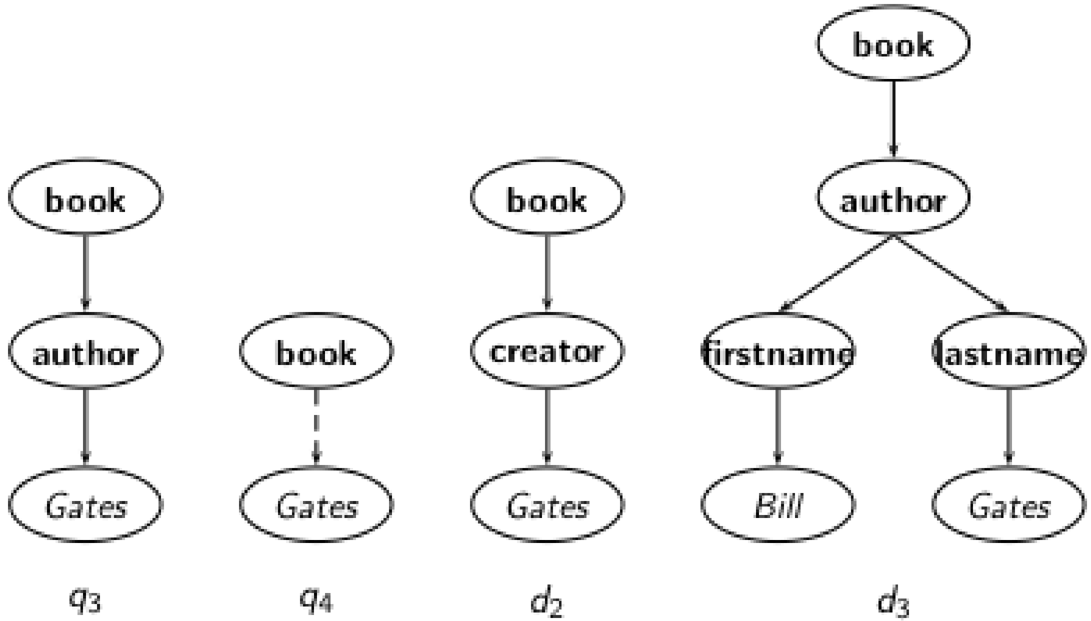
- A simple measure of the similarity of a path c_q in a query and a path c_d in a document is the following ***context resemblance*** function CR:

$$\mathrm{CR}(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{if } c_q \text{ matches } c_d \\ 0 & \text{if } c_q \text{ does not match } c_d \end{cases}$$

  $|c_q|$ and $|c_d|$ are the number of nodes in the query path and document path, respectively

- c_q matches c_d **iff** we can transform c_q into c_d by inserting additional nodes.

Context resemblance example

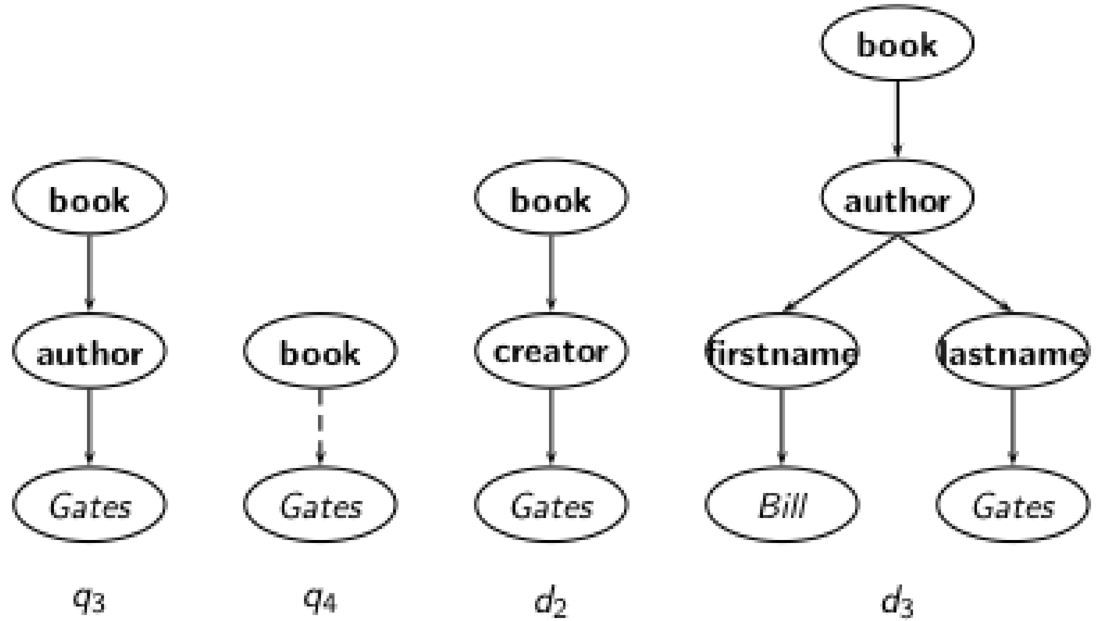


$$\mathrm{CR}(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{if } c_q \text{ matches } c_d \\ 0 & \text{if } c_q \text{ does not match } c_d \end{cases}$$

- Cr(c_{q4}, c_{d2}) = 3/4 = 0.75.
  The value of Cr (c_q, c_d) is 1.0 if *q* and *d* are identical.

Blanks on slides, you may want to fill in

Context resemblance example



$$\mathrm{CR}(c_q, c_d) = \begin{cases} \frac{1+|c_q|}{1+|c_d|} & \text{if } c_q \text{ matches } c_d \\ 0 & \text{if } c_q \text{ does not match } c_d \end{cases}$$

- Cr($c_{q?}$, $c_{d?}$) = Cr (c_q, c_d) = 3/5 = 0.6.

Document similarity measure

- The final score for a document is computed as a variant of the cosine measure, which we call SimNoMerge.
- SimNoMerge(q, d) =

$$\sum_{c_k \in B} \sum_{c_l \in B} \mathrm{CR}(c_k, c_l) \sum_{t \in V} \mathrm{weight}(q, t, c_k) \frac{\mathrm{weight}(d, t, c_l)}{\sqrt{\sum_{c \in B, t \in V} \mathrm{weight}^2(d, t, c)}}$$

- V is the vocabulary of non-structural terms
- B is the set of all XML contexts
- weight (q, t, c), weight(d, t, c) are the weights of term t in XML context c in query q and document d, resp. (standard weighting e.g. idf_t x $\mathrm{wf}_{t,d}$, where idf_t depends on which elements we use to compute df_t.)
- SimNoMerge (q, d) is not a true cosine measure since its value can be larger than 1.0.

SIMNOMERGE example

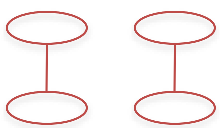
example **slightly different** from book

query

$\langle c_1, t \rangle$

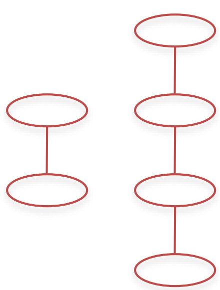
Inverted index

dictionary	postings		
$\langle c_1,t \rangle$	$\langle d_1,0.5 \rangle$	$\langle d_4,0.1 \rangle$	$\langle d_9,0.2 \rangle$
$\langle c_2,t \rangle$	$\langle d_2,0.25 \rangle$	$\langle d_3,0.1 \rangle$	$\langle d_{12},0.9 \rangle$
$\langle c_3,t \rangle$	$\langle d_3,0.7 \rangle$	$\langle d_6,0.8 \rangle$	$\langle d_9,0.5 \rangle$



$CR(c_1,c_1) = 1.0$

$CR(c_1,c_2) = 0.0$



$CR(c_1,c_3) = 0.60$

if $w_q = 1.0$, then $sim(q,d_9) =$
$(1.0 \times 1.0x0.2) + (0.6 \times 1.0x0.5) = .5$

"No Merge" because each context is separately calculated

SimNoMerge algorithm

ScoreDocumentsWithSimNoMerge (q, *B*, *V*, *N*, *normalizer*)

for $n \leftarrow 1$ **to** N
do $score[n] \leftarrow 0$
for each $\langle c_q, t \rangle \in q$
do $w_q \leftarrow \textsc{Weight}(q, t, c_q)$
  for each $c \in B$
  do if $\textsc{Cr}(c_q, c) > 0$
    then $postings \leftarrow \textsc{GetPostings}(\langle c, t \rangle)$
      for each $posting \in postings$
      do $x \leftarrow \textsc{Cr}(c_q, c) * w_q * weight(posting)$
        $score[docID(posting)] += x$
for $n \leftarrow 1$ **to** N
do $score[n] \leftarrow score[n] / normalizer[n]$
return $score$

XML IR EVALUATION

Initiative for the Evaluation of XML retrieval (INEX)

INEX: standard benchmark evaluation (yearly) that has produced test collections (documents, sets of queries, and relevance judgments). Based on IEEE journal collection (since 2006 INEX uses the much larger English Wikipedia test collection).
The relevance of documents is judged by human assessors.

INEX 2002 collection statistics	
12,107	number of documents
494 MB	size
1995—2002	time of publication of articles
1,532	average number of XML nodes per document
6.9	average depth of a node
30	number of CAS topics
30	number of CO topics

INEX Topics

- Two types:

1. content-only or **CO topics**: regular keyword queries as in unstructured information retrieval
2. content-and-structure or **CAS topics**: have structural constraints in addition to keywords

- Since CAS queries have both structural and content criteria, relevance assessments are more complicated than in unstructured retrieval

INEX relevance assessments

- INEX 2002 defined component coverage and topical relevance as orthogonal dimensions of relevance.

Component coverage

Evaluates whether the element retrieved is "structurally" correct, i.e., neither too low nor too high in the tree.

- We distinguish four cases:

1. Exact coverage (E): The information sought is the main topic of the component and the component is a meaningful unit of information.
2. Too small (S): The information sought is the main topic of the component, but the component is not a meaningful (self-contained) unit of information.
3. Too large (L): The information sought is present in the component, but is not the main topic.
4. No coverage (N): The information sought is not a topic of the component.

INEX relevance assessments

- The **topical relevance** dimension also has four levels: highly relevant (3), fairly relevant (2), marginally relevant (1) and nonrelevant (0).

Combining the relevance dimensions

Components are judged on both dimensions and the judgments are then combined into a digit-letter code, e.g. **2S** is a fairly relevant component that is too small. In theory, there are 16 combinations of coverage and relevance, but many cannot occur. For example, a nonrelevant component cannot have exact coverage, so the combination **3N** is not possible.

INEX relevance assessments

- The relevance-coverage combinations are quantized as follows:

$$\mathbf{Q}(rel, cov) = \begin{cases} 1.00 & \text{if} \quad (rel, cov) = 3\text{E} \\ 0.75 & \text{if} \quad (rel, cov) \in \{2\text{E}, 3\text{L}, 3\text{S}\} \\ 0.50 & \text{if} \quad (rel, cov) \in \{1\text{E}, 2\text{L}, 2\text{S}\} \\ 0.25 & \text{if} \quad (rel, cov) \in \{1\text{S}, 1\text{L}\} \\ 0.00 & \text{if} \quad (rel, cov) = 0\text{N} \end{cases}$$

- This evaluation scheme takes account of the fact that binary relevance judgments are not appropriate for XML retrieval. The quantization function **Q** instead allows us to grade each component as partially relevant. The number of relevant components in a retrieved set *A* of components can then be computed as:

$$\#(\text{relevant items retrieved}) = \sum_{c \in A} \mathbf{Q}(rel(c), cov(c))$$

Summary

1. Relevance Feedback – “Documents”
2. Query Expansion – “Terms”

3. XML IR and Evaluation
   - Structured or XML IR: effort to port unstructured IR know-how to structured
     (DB-like) data
   - Specialized applications such as patents and digital libraries

- Resources
  - IIR Ch 9/10
  - MG Ch. 4.7 and MIR Ch. 5.2 – 5.4
  - http://inex.is.informatik.uni-duisburg.de/