Information Retrieval

Databases of text

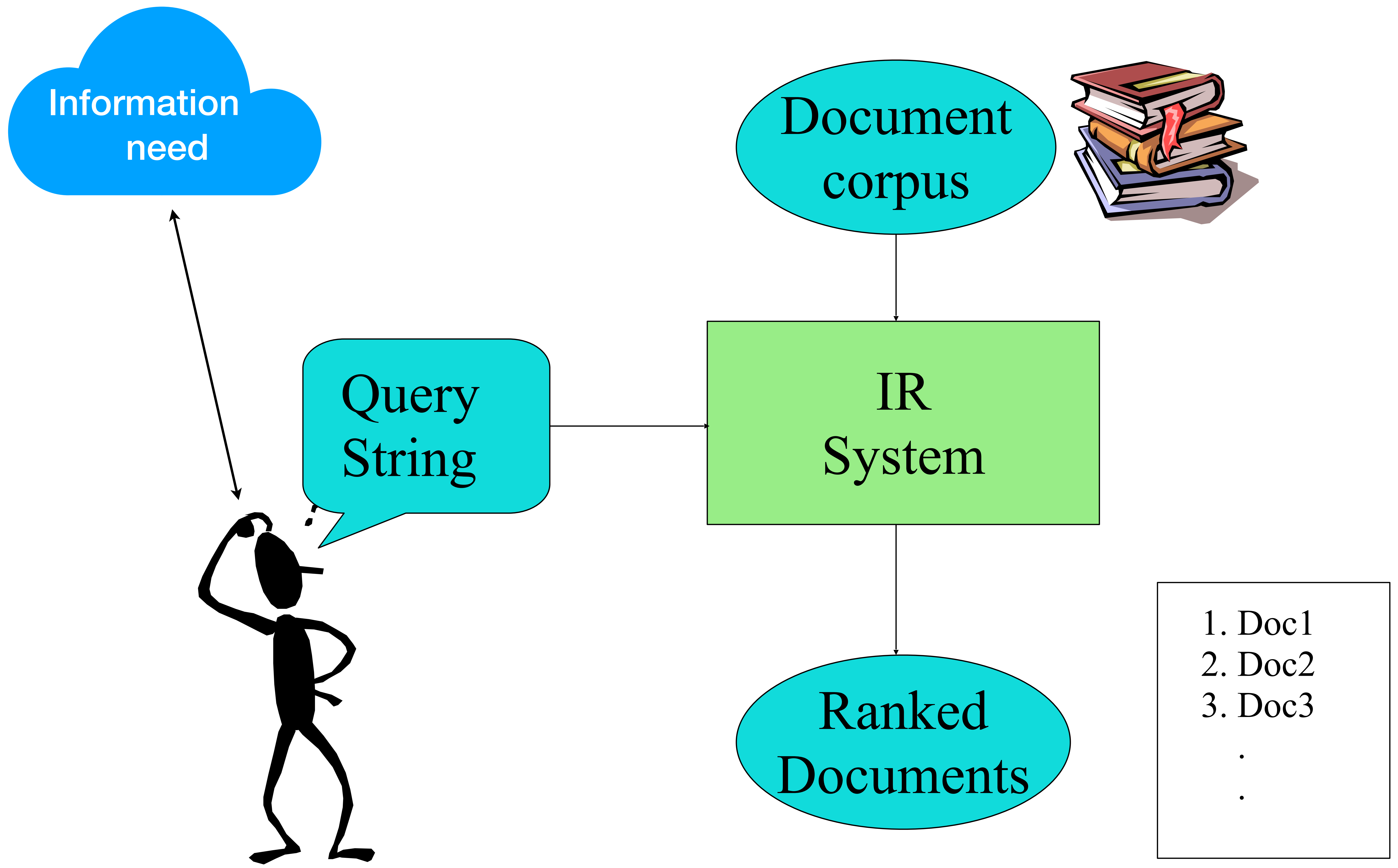
Information Retrieval

- 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).

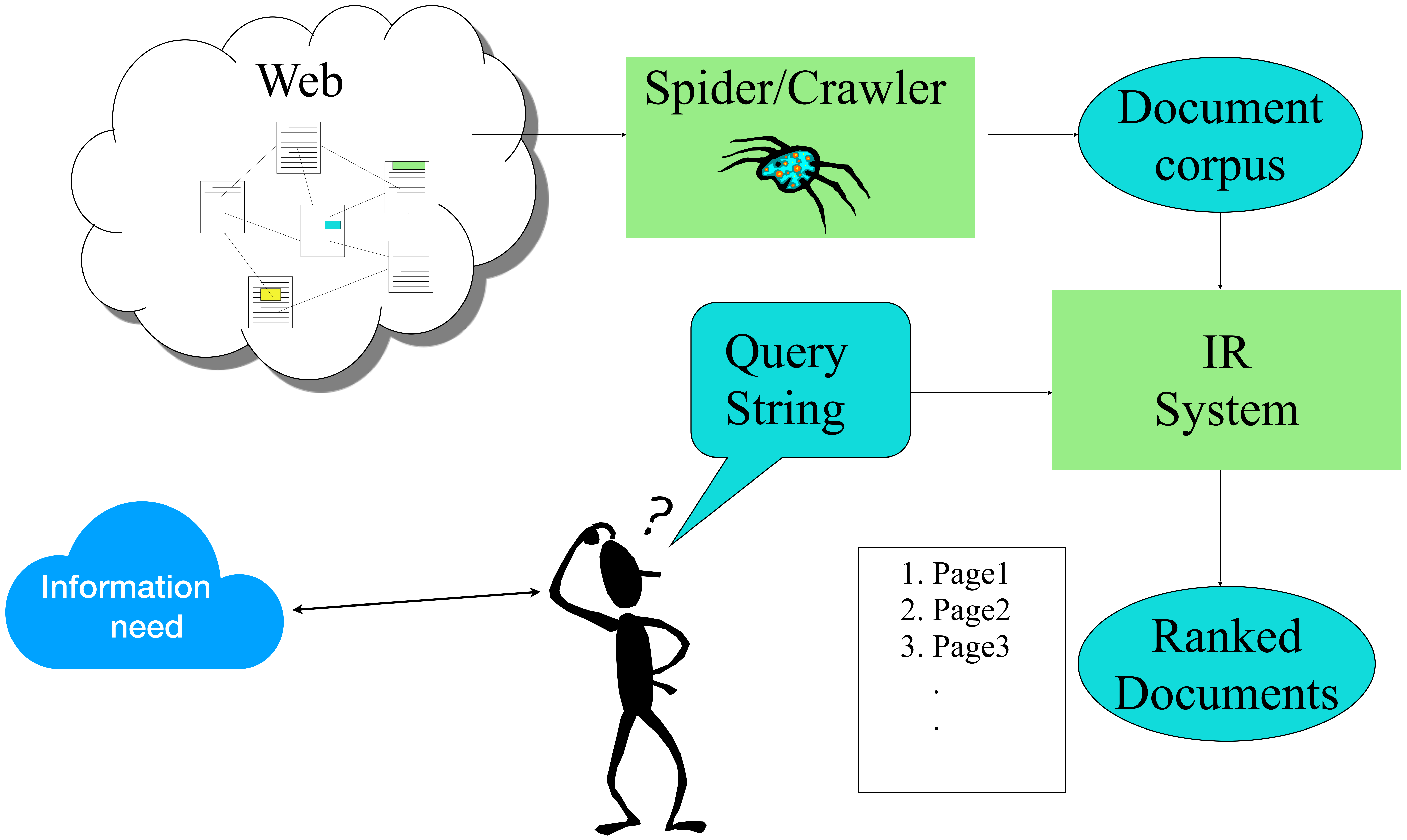
Information Retrieval (IR)

- The indexing and retrieval of textual documents.
- Concerned firstly with retrieving *relevant* documents to a query.
- Concerned secondly with retrieving from *large* sets of documents *efficiently*.

IR System


Information need
Document corpus
Query String
IR System
Ranked Documents
1. Doc1
2. Doc2
3. Doc3
.
.

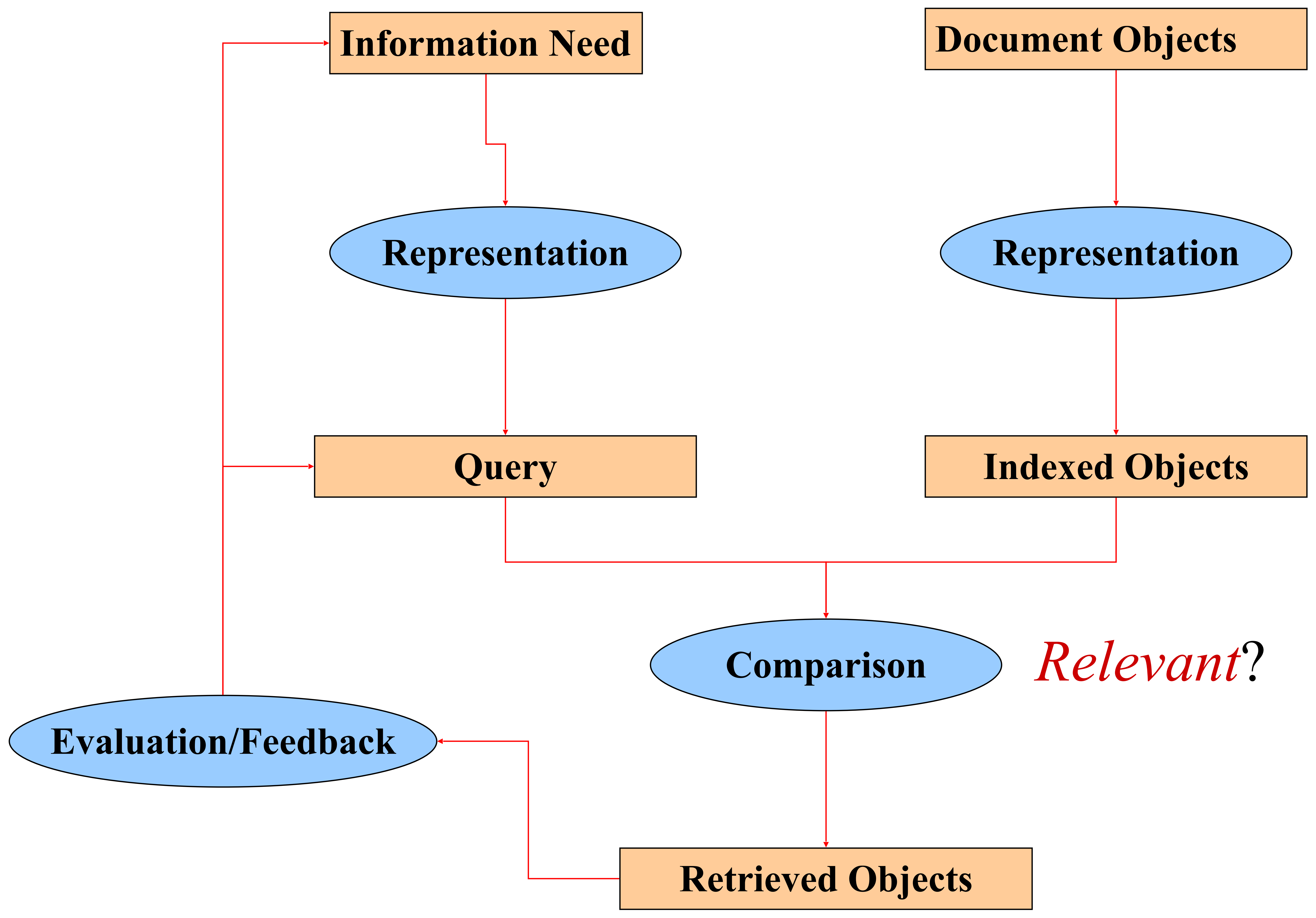
Web Search System



IR v. Database Systems

- Emphasis on effective, efficient retrieval of unstructured (or semi-structured) data
- IR systems typically have very simple schemas
- Query languages emphasize free text and Boolean combinations of keywords
- Matching is more complex than with structured data (semantics is less obvious)
  - easy to retrieve the wrong objects
  - need to measure the accuracy of retrieval
- Less focus on concurrency control and recovery (although update is very important).

Information Retrieval as a Process



Keyword Search

- Simplest notion of relevance is that the query string appears verbatim in the document.
- Slightly less strict notion is that the words in the query appear frequently in the document, in any order (*bag of words*).

Problems with Keywords

- May not retrieve relevant documents that include synonymous terms.
  - "restaurant" vs. "café"
  - "PRC" vs. "China"
- May retrieve irrelevant documents that include ambiguous terms.
  - "bat" (baseball vs. mammal)
  - "Apple" (company vs. fruit)
  - "bit" (unit of data vs. act of eating)

Indexing

- Indexing is the process of transforming items (documents) into a searchable data structure
  - creation of document surrogates to represent each document
  - requires analysis of original documents
    - simple: identify meta-information (e.g., author, title, etc.)
    - complex: linguistic analysis of content

- The search process involves correlating user queries with the documents represented in the index

On what should you index?

- Entire document? Documents have many parts!
- Undifferentiated?
    - handle meta-information separately?
      what is meta-information for books
    - if not separate, how to merge?
- What about the web?
    - link text? Usually not even written by the author!

Index Terms or "Features"

- Most common feature types:
  - Words in text
  - N-grams — (consecutive substrings) of a document
  - Manually assigned terms (controlled vocabulary)
  - Document structure (sentences & paragraphs)
  - Inter- or intra-document links (e.g., citations)
- Composed features
  - Feature sequences (phrases, names, dates, monetary amounts)
  - Feature sets (e.g., synonym classes, concept indexing)
  - Bi-Grams and Tri-Grams
    - there are lots
      - Google TriGram corpus — min freq 40
        - 1-Grams about 13 million — 185 M
        - 2-Grams about 31,000,000 items
        - 3-Grams about 97,000,000 items

Basic Automatic Indexing

1. Parse documents to recognize structure
   – e.g. title, date, other fields
2. Scan for word tokens (Tokenization)
   – lexical analysis using finite state automata(?)
   – numbers, special characters, hyphenation, capitalization, etc.
   – languages like Chinese need *segmentation* since there is not explicit word separation
   – record positional information for *proximity* operators
3. *Stopword* removal
   – based on short list of common words such as “the”, “and”, “or”
   – saves storage overhead of very long postings lists
   – can be dangerous (e.g. “Mr. The”, “and-or gates”)

Basic Automatic Indexing

4. Stem words
   - morphological processing to group word variants such as plurals
   - better than string matching (e.g. comput*)
   - can make mistakes but generally preferred
5. Weight words
   - using frequency in documents and database
   - frequency data is independent of retrieval model
6. Optional
   - phrase indexing / positional indexing
   - thesaurus classes / concept indexing

Tokenization

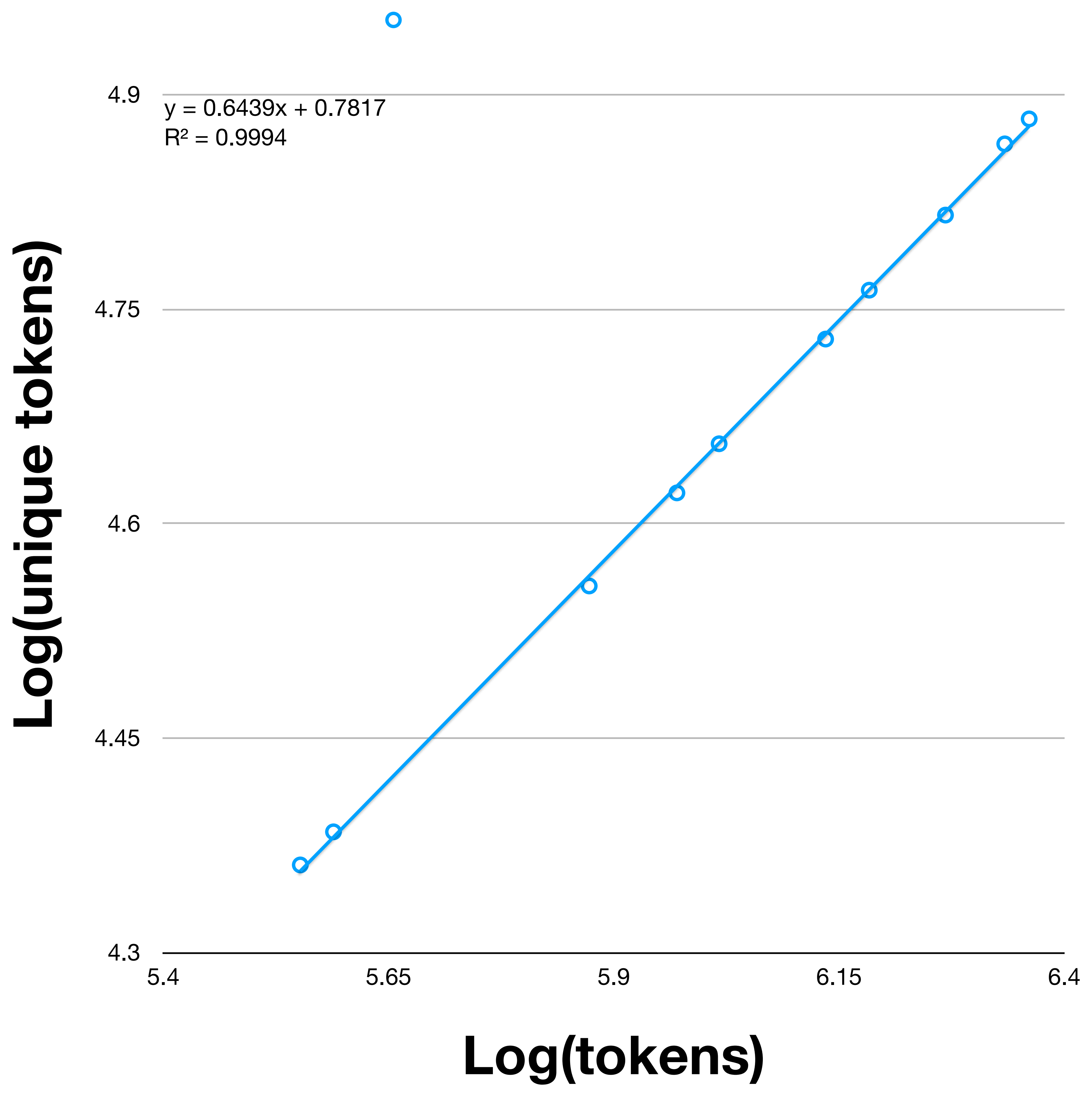
- Turn text strings into searchable items
    - remove suffixes (Stemming)
        - Should you stem??
            - As opposed to stemming, just use N-grams
    - Numbers?
    - Punctuation
        - U.S.A.
        - State-of-the-art
- Fix Misspellings?
    - Tempting to say “yes always”.
        - Sometimes misspelling are highly diagnostic.
            - Misspellings of names — how do you even know?

Stop words

- Idea: exclude from the dictionary the list of words with little semantic content: a, and, or , how, where, to, ….
  - They tend to be the most common words: 30 most common words account for 30% of all words
- But the trend is away from doing this:
  - Good compression techniques means the space for including stop words 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 titles, etc.: “Let it be”, “To be or not to be”
    - “Relational” queries: “flights to London”
  - Google ignores common words and characters such as where, the, how, and other digits and letters which slow down your search without improving the results.” (Though you can explicitly ask for them to remain)

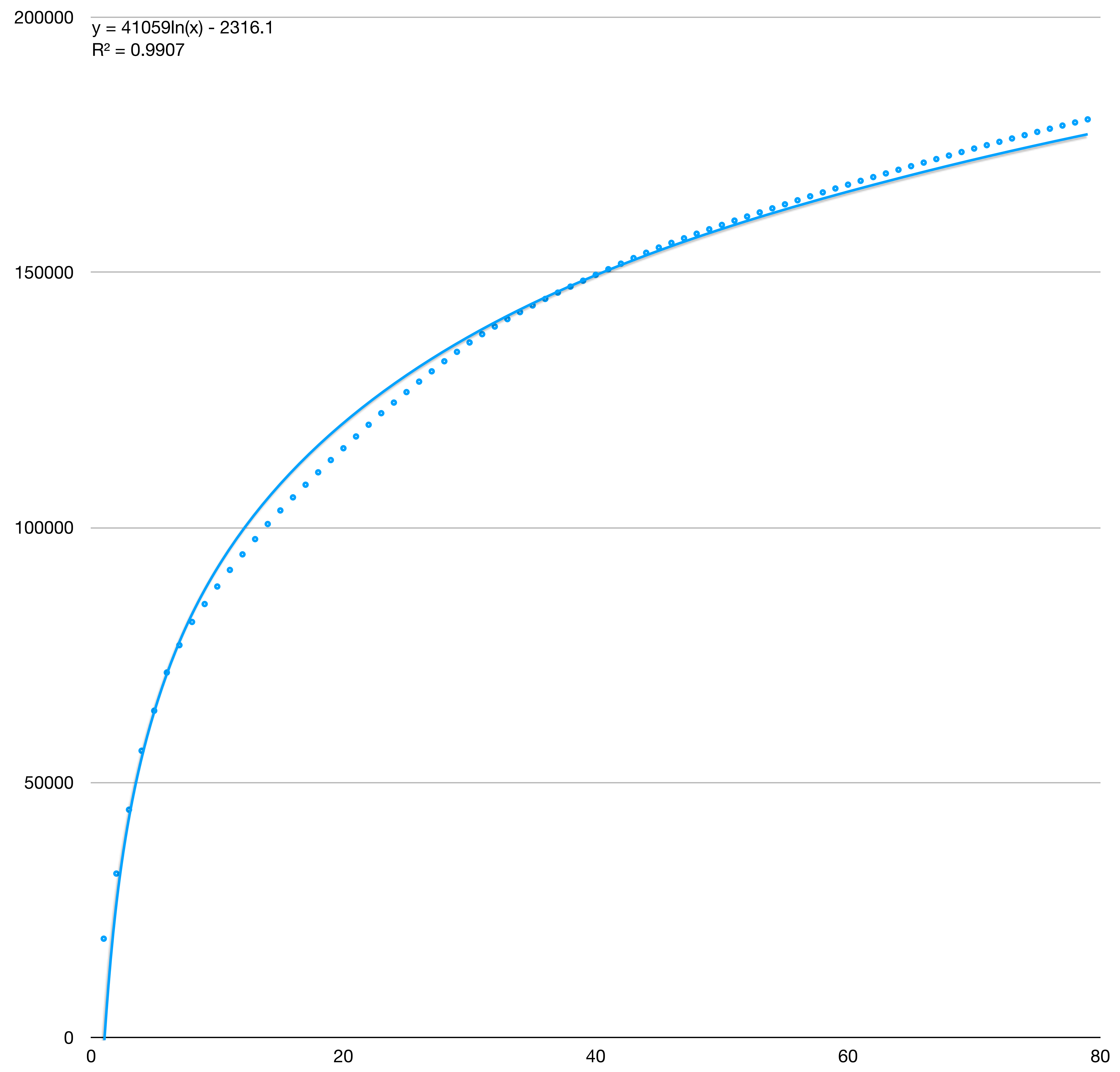
Heaps' Law & Dickens

- **Heaps Law (1960):**
  - **Vocabulary size is a function of collection size.**
- M = kT^b
  - M = vocabulary
  - k = constant
    - typically 30-100
  - T = collection size (just wc)
  - b rate of growth.
    - Typically 0.5
- Dickens
  - k = 6.02 (10^0.78)
  - b = 0.64



Zipf's law

- the frequency of any word is inversely proportional to its rank in the frequency table
- George Zipf 1935
  - but it predates him
- Zipf's law is asymptotically equivalent to Heaps law



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*).

Statistical Models

- A document is typically represented by a *bag of words* (unordered words with frequencies).
- Bag = set that allows multiple occurrences of the same element.
- User specifies a set of desired terms with optional weights:
  - Weighted query terms:
    Q = < database 0.5; text 0.8; information 0.2 >
  - Unweighted query terms:
    Q = < database; text; information >
  - No Boolean conditions specified in the query.

Statistical Retrieval

- Retrieval based on *similarity* between query and documents.
- Output documents are ranked according to similarity to query.
- Similarity based on occurrence *frequencies* of keywords in query and document.
- Automatic relevance feedback can be supported:
  - Relevant documents “added” to query.
  - Irrelevant documents “subtracted” from query.

The Vector-Space Model

- Assume t distinct terms remain after preprocessing; call them index terms or the vocabulary.

- These "orthogonal" terms form a vector space.

  Dimensionality = t = |vocabulary|

- Each term, i, in a document or query, j, is given a real-valued weight, $w_{ij.}$

- Both documents and queries are expressed as t-dimensional vectors:

  $d_j = (w_{1j}, w_{2j}, \ldots, w_{tj})$

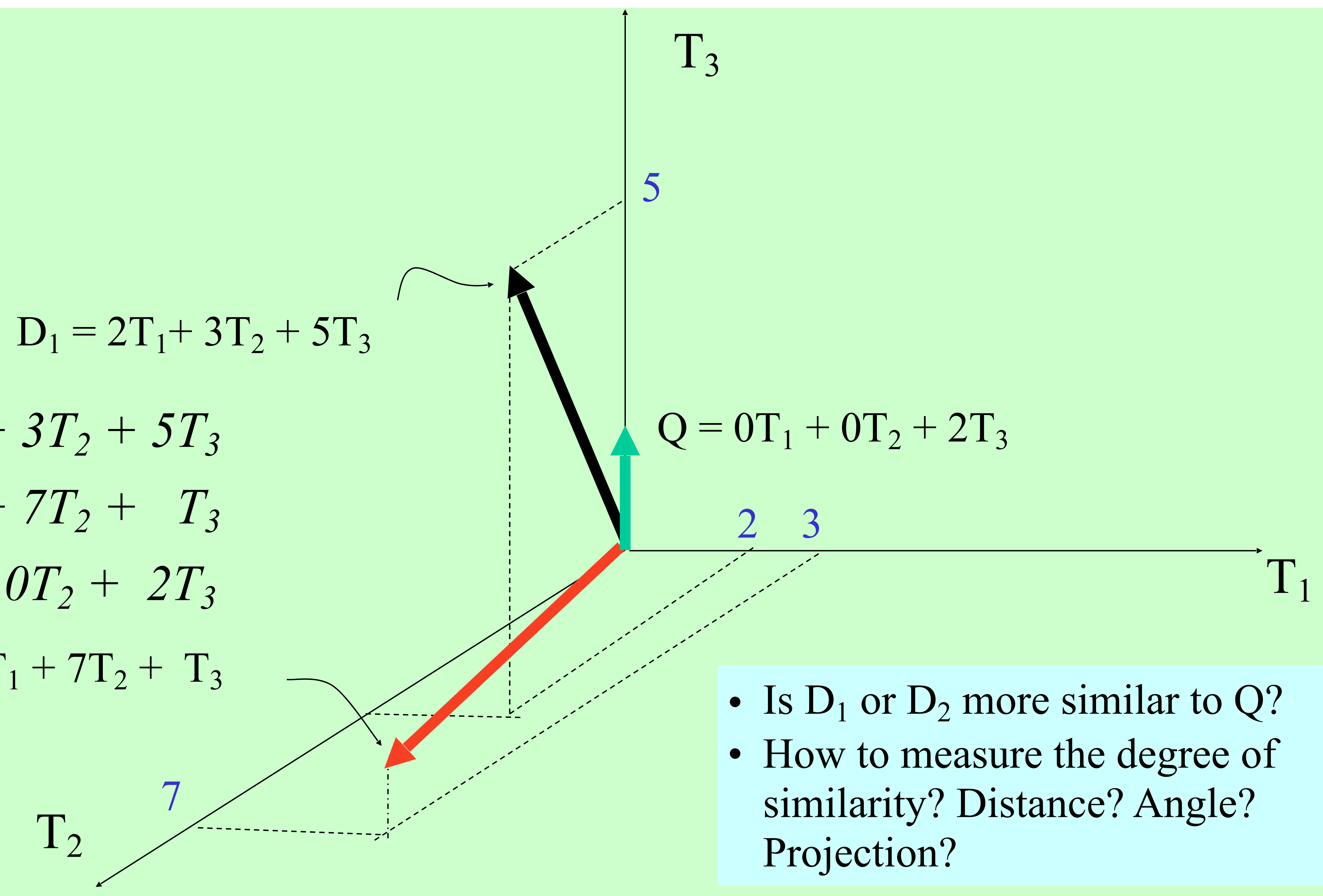
Graphic Representation

Example:

$D_1 = 2T_1 + 3T_2 + 5T_3$

$D_2 = 3T_1 + 7T_2 + T_3$

$Q = 0T_1 + 0T_2 + 2T_3$



- Is D_1 or D_2 more similar to Q?
- How to measure the degree of similarity? Distance? Angle? Projection?

Issues for Vector Space Model

- How to determine important words in a document?
  - Word sense?
  - Word n-grams (and phrases, idioms,…) $\rightarrow$ terms
- How to determine the degree of importance of a term within a document and within the entire collection?
- How to determine the degree of similarity between a document and the query?
- In the case of the web, what is the collection and what are the effects of links, formatting information, etc.?

Term Weights: Term Frequency

- More frequent terms in a document are more important, i.e. more indicative of the topic.

  f_{ij} = frequency of term i in document j

- May want to normalize *term frequency* (*tf*) by dividing by the frequency of the most common term in the document:

  $tf_{ij} = f_{ij} / max_i\{f_{ij}\}$

Term Weights:

Inverse Document Frequency

- Terms that appear in many *different* documents are *less* indicative of overall topic.

  df_i = document frequency of term i

  = number of documents containing term i

  idf_i = inverse document frequency of term i,

  $= \log_2 (N/ df_i)$

  (N: total number of documents)

- An indication of a term's *discrimination* power.

- Log used to dampen the effect relative to *tf*.

TF-IDF Weighting

- A typical combined term importance indicator is *tf-idf weighting*:

$$w_{ij} = tf_{ij}\, idf_i = tf_{ij} \log_2 (N/\, df_i)$$

- A term occurring frequently in the document but rarely in the rest of the collection is given high weight.
- Many other ways of determining term weights have been proposed.
- Experimentally, *tf-idf* has been found to work well.

Computing TF-IDF -- An Example

Given a document containing terms with given frequencies:

A(3), B(2), C(1)

Assume collection contains 10,000 documents and document frequencies of these terms are:

A(50), B(1300), C(250)

Then:

A: tf = 3/3; idf = $\log_2(10000/50) = 7.6$; tf-idf = 7.6

B: tf = 2/3; idf = $\log_2(10000/1300) = 2.9$; tf-idf = 2.0

C: tf = 1/3; idf = $\log_2(10000/250) = 5.3$; tf-idf = 1.8

Similarity Measure
Inner Product

- Similarity between vectors for the document $\boldsymbol{d}_i$ and query $\boldsymbol{q}$ can be computed as the vector inner product (a.k.a. dot product):

$$\text{sim}(\boldsymbol{d}_j, \boldsymbol{q}) = \boldsymbol{d}_j \bullet \boldsymbol{q} = \sum_{i=1}^{t} w_{ij} w_{iq}$$

  where w_{ij} is the weight of term i in document j and w_{iq} is the weight of term i in the query

- For binary vectors, the inner product is the number of matched query terms in the document (size of intersection).
- For weighted term vectors, it is the sum of the products of the weights of the matched terms.

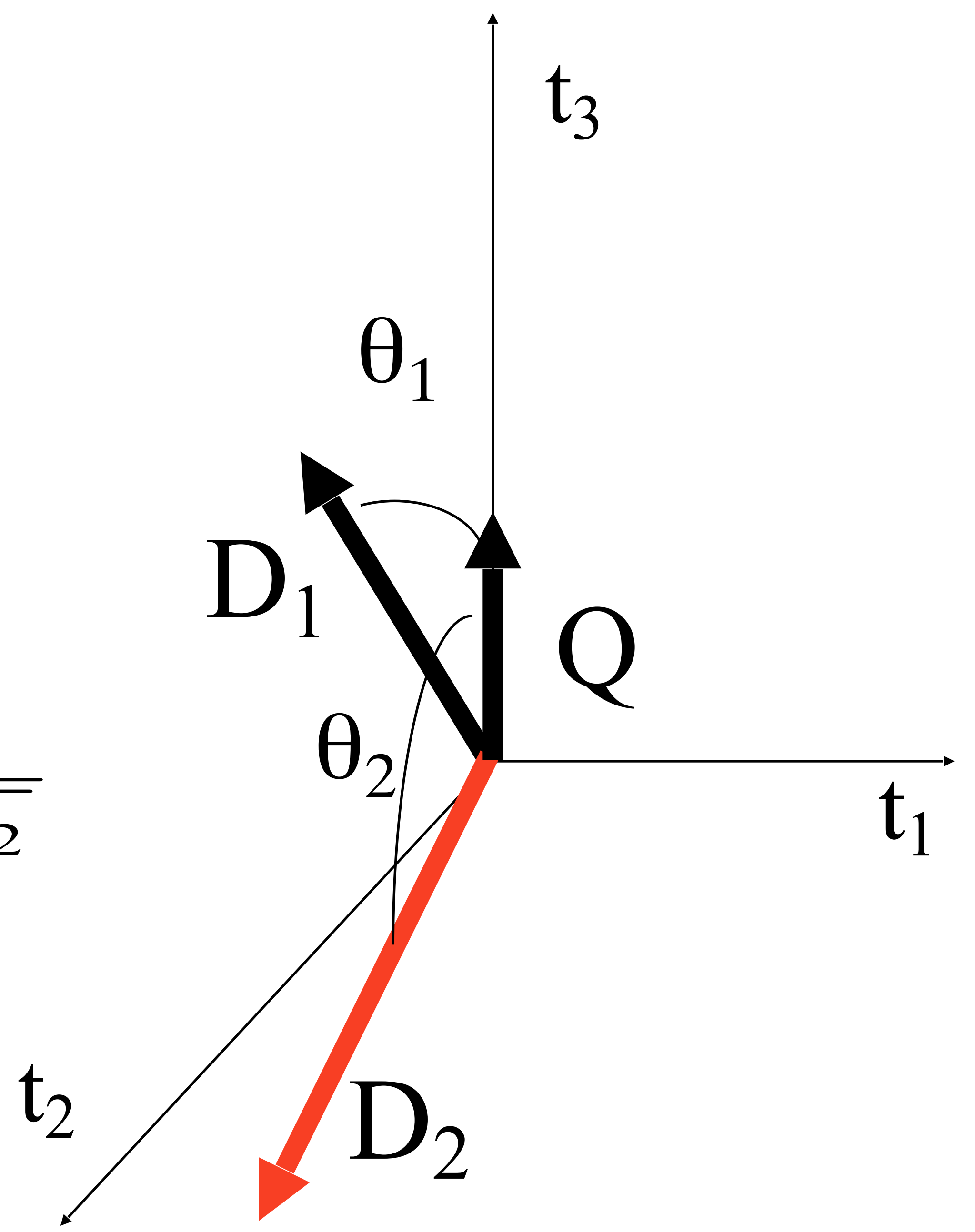
Properties of Inner Product

- The inner product is unbounded.
- Favors long documents with a large number of unique terms.
- Measures how many terms matched but not how many terms are *not* matched.

Cosine Similarity Measure

- Cosine similarity measures the cosine of the angle between two vectors.
- Inner product normalized by the vector lengths.

$$\text{CosSim}(\mathbf{d}_j, \mathbf{q}) = \frac{\sum_{i=1}^{t} (w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$



$D_1 = 2T_1 + 3T_2 + 5T_3$ CosSim(D_1 , Q) = $10 / \sqrt{(4+9+25)(0+0+4)} = 0.81$
$D_2 = 3T_1 + 7T_2 + 1T_3$ CosSim(D_2 , Q) = $2 / \sqrt{(9+49+1)(0+0+4)} = 0.13$
$Q = 0T_1 + 0T_2 + 2T_3$

D_1 is 6 times better than D_2 using cosine similarity but only 5 times better using inner product.

Simple Implementation

Convert all documents in collection D to *tf-idf* weighted vectors, $\boldsymbol{d_j}$, for keyword vocabulary V.

Convert query to a *tf-idf*-weighted vector $\boldsymbol{q}$.

```
For each $\boldsymbol{d_j}$ in D do
    Compute score $s_j$ = cosSim($\boldsymbol{d_j}, \boldsymbol{q}$)
```

Sort documents by decreasing score.

Present top ranked documents to the user.

Time complexity: O(|V|·|D|) Bad for large V & D !

|V| = 10,000; |D| = 100,000; |V|·|D| = 1,000,000,000