Introduction to Information Retrieval

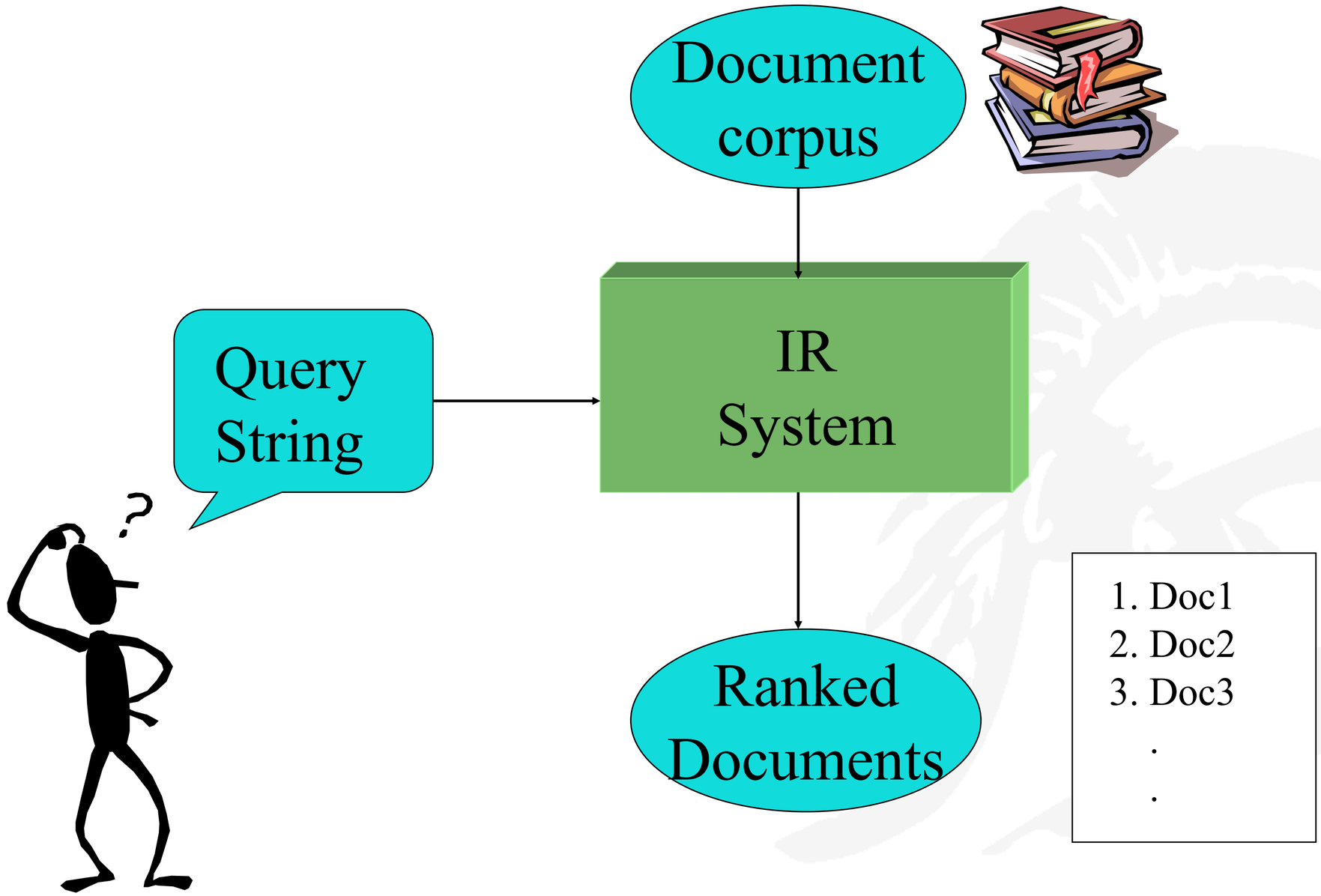
Information Retrieval

- ***Information retrieval (IR)* has been a computer science subject for many decades**
  - Traditionally it deals with the *indexing* of a given set of textual documents and the *retrieval* of relevant documents given a query
- Relatively recently searching for pages on the World Wide Web has become the "killer app."

- **There has been a great deal of research on**
  - How to index a set of documents (a corpus)
  - How to efficiently retrieve relevant documents
- Jurafsky and Manning have an excellent video introducing the subject of Information Retrieval;
- http://www-scf.usc.edu/~csci572/movies/01_IntroIR.mp4 (9 minutes)

then jump to slide 19

The Traditional IR System



History of IR

- **1960-70's:**
  - Initial exploration of text retrieval systems for "small" corpora of scientific abstracts, and law and business documents.
  - Development of the basic Boolean and vector-space models of retrieval.
  - Prof. Salton and his students at Cornell University were the leading researchers in the area.

IR History Continued

- **1980's:**
  - **Creation of large document database systems, many run by companies:**
    - Lexis-Nexis, http://www.lexisnexis.com/
      - information to legal, corporate, government and academic markets, and publishes legal, tax and regulatory information
    - Dialog, http://www.dialog.com/
      - data from more than 1.4 billion unique records of key information.
    - MEDLINE, http://www.medlineplus.gov/
      - National Library of Medicine health information

IR History Continued

- **1990's:**
  - **Searching FTP'able documents on the Internet**
    - Archie
    - WAIS
  - **After the World Wide Web is invented, search engines appear**
    - Lycos
    - Yahoo
    - Altavista

IR History Continued

- **1990's continued:**
  - **Organized Competitions**
    - NIST TREC (Text REtrieval Conferences, http://trec.nist.gov/)
    - Sponsored by National Institute of Standards and Technology, NIST
  - **Several New Types of IR Systems are Developed**

    1. Recommender Systems: computer programs which attempt to predict items (movies, music, books, news, web pages) that a user may be interested in, given some information about the user's profile.
       - **Often implemented as a collaborative filtering algorithm, examples include:**
         - » **Ringo, music recommendation system**
         - » **Amazon's recommendation system, see http://answers.google.com/answers/threadview?id=29373**

    2. Automated Text Categorization & Clustering
       - **Useful for grouping news articles**

Recent IR History Moves to the Web

- **2000's**
  - **Link analysis for Web Search**
    - Google started this
  - **Extension to retrieval of multimedia: images, music, video**
    - It is much harder to index multimedia artifacts
  - **Question Answering**
    - Question answering systems return an actual answer rather than a ranked list of documents
    - Since 1999 TREC has had a Question/Answer track, see http://trec.nist.gov/data/qa.html

Areas Related To, But Different Than, Information Retrieval

- **Database Management**
- **Library and Information Science**
- **Artificial Intelligence**
- **Natural Language Processing**
- **Machine Learning**

Database Management is Different from IR

- **Focused on *structured* data stored in relational tables rather than free-form text**
- **Focused on efficient processing of well-defined queries in a formal language (SQL)**
- **Clearer semantics for both data and queries**
- **Web pages are mostly unstructured, though the Document Object Model (DOM) can provide some clues**

Library and Information Science

- **Focused on the human user aspects of information retrieval (human-computer interaction, user interface, visualization).**
- **Concerned with effective categorization of human knowledge.**
- **Concerned with citation analysis and *bibliometrics* (structure of information).**
- **Recent work on *digital libraries* brings it closer to Computer Science & IR.**

Artificial Intelligence

- **Focused on the representation of knowledge, reasoning, and intelligent action.**
- **Formalisms for representing knowledge and queries:**
  - ***First-order Predicate Logic*** **–** a formal system that uses quantified variables over a specified domain of discourse
  - ***Bayesian Networks*** **–** a directed acyclic graph model that represents a set of random variables and their dependencies
    - **E.g.** A Bayesian Network that represents the probabilistic relationships between diseases and symptoms
- **Recent work on web ontologies and intelligent information agents brings it closer to IR**
  - Web Ontology Language OWL is a family of knowledge representation languages for authoring ontologies
  - See https://www.w3.org/OWL/

Natural Language Processing

- **Focused on the syntactic, semantic, and pragmatic analysis of natural language text and discourse.**
- **Ability to analyze syntax (phrase structure) and semantics could allow retrieval based on *meaning* rather than keywords**
- **But search engines discourage long, natural language queries as there is too much chance for ambiguity of meaning**

Machine Learning

- **A branch of Artificial Intelligence concerned with algorithms that allow computers to evolve their behavior based on empirical data**
- **Focused on the development of computational systems that improve their performance with experience**
- **Two major subtypes of machine learning are**:
  - Automated classification of examples based on learning concepts from labeled training examples (*supervised learning*).
  - Automated methods for clustering unlabeled examples into meaningful groups (*unsupervised learning*)
- **Machine learning is distinct from *data mining*, which focuses on the discovery of previously unknown properties of the given data**
  - Data mining is akin to query analysis and ranking

Basic Information Retrieval Begins with Keyword Matching

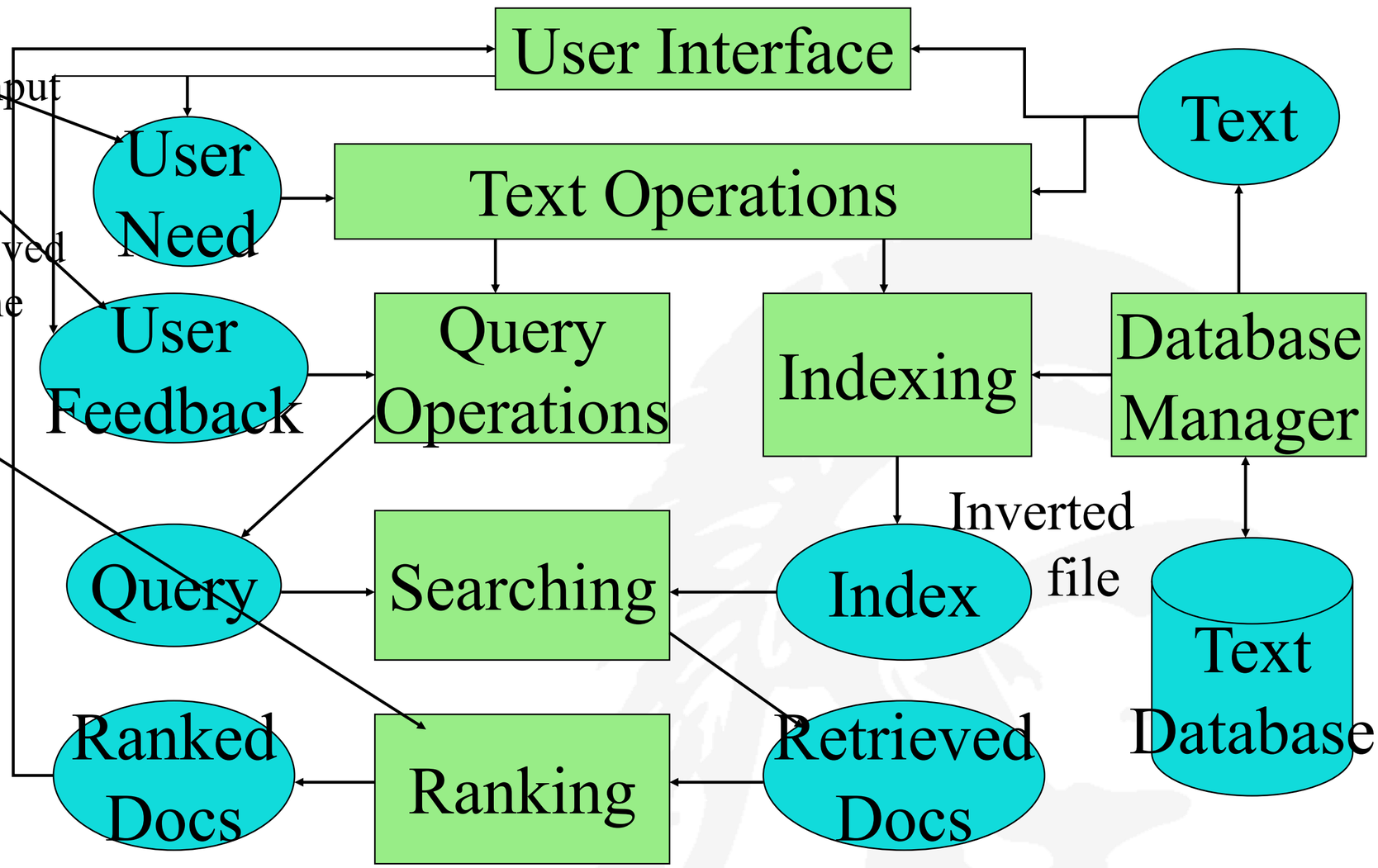
- **Simplest notion of relevance is that the query string appears verbatim in the document.**
  - Slightly less strict notion is that the words in the query appear frequently in the document, in any order (this is like viewing the document as a *bag of words*).
- **But that may not retrieve relevant documents that include synonymous terms.**
  - "restaurant" vs. "café"
  - "PRC" vs. "China"
- **And it may retrieve irrelevant documents that include ambiguous terms.**
  - "bat" (baseball vs. mammal)
  - "Apple" (company vs. fruit)
  - "bit" (unit of data vs. act of eating)

- **Goes beyond using just keyword matching, instead it**
  - Takes into account the *meaning* of the words used
  - Takes into account the *order* of words in the query
  - Adapts to the user based on direct or indirect feedback
  - Taking into account the *authority* of the source

A More Detailed IR Architecture

Logical View

- User needs are part of the input
- User feedback is provided
- Queries initiate a search of the index and docs are retrieved
- A ranking function orders the results



Start with a text database; it is indexed; a user interface permits query operations which cause a search on the Index; matched documents are retrieved and ranked

Defining Terms

- **Parsing forms index words (tokens) and includes:**
  - *Stopword* removal
    - See http://www.ranks.nl/tools/stopwords.html for google stopwords
  - *Stemming*: reducing a word to its root
    - More about this later
- ***Indexing* constructs an *inverted index* of word to document pointers.**
- ***Searching* retrieves documents that contain a given query token from the inverted index.**
- ***Ranking* scores all retrieved documents according to a relevance metric.**

Boolean and Vector Space Retrieval Models

***Primary goal*: to formalize the processes that underlie a person making the decision that a piece of text is relevant to his information need**

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*)**
- **Three major Information Retrieval Models are:**

1. Boolean models (set theoretic) (*Chapter 1 in Manning et al*)
2. Vector space models (statistical/algebraic) (*Chapter 2 in Manning et al*)
3. Probabilistic models (*Chapter 11 in Manning et al*)

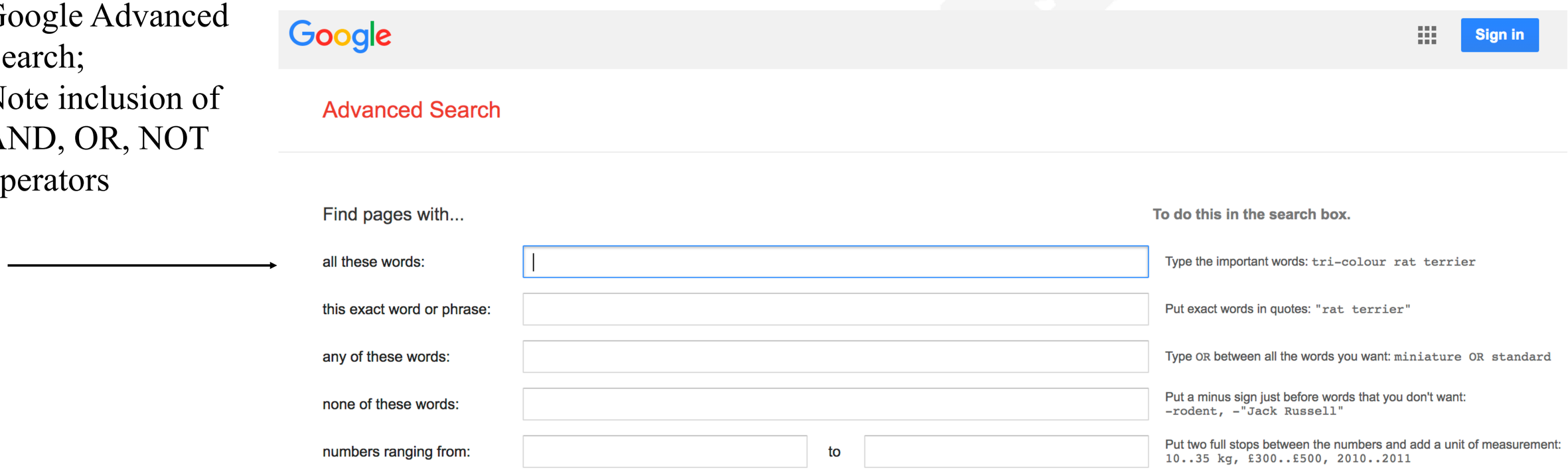
Common Pre-Processing Steps

1. **Strip unwanted characters/markup (e.g. HTML tags, punctuation, numbers, etc.).**
2. **Break into tokens (keywords) or whitespace.**
3. **Stem tokens to "root" words**
   - **computational → comput** **(more about this later)**
4. **Remove common stopwords (e.g. a, the, it, etc.).**
5. **Detect common phrases (possibly using a domain specific dictionary).**
6. **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**
- **Here is a sample Boolean query with explicit AND, OR, NOT operators**
  - [[Rio & Brazil] | [Hilo & Hawaii]] & hotel & !Hilton]

Google Advanced Search;
Note inclusion of AND, OR, NOT operators



Boolean Retrieval Model

- **Popular retrieval model because:**
  - Easy to understand for simple queries.
  - Clean formalism.
- **Boolean models can be extended to include ranking**
- **Reasonably efficient implementations possible for normal queries.**

Boolean Models – Problems

- **Very rigid: AND means all; OR means any**
- **Difficult to express complex user requests**
- **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?

Problem Example For the Boolean Model

- **The simple query "Lincoln"**
  - Too many matches including Lincoln cars and places named Lincoln as well as Abraham Lincoln
- **More detailed query "President AND Lincoln"**
  - Returns documents that discuss the President of Ford Motor company that makes the Lincoln car
- **Even more detailed query "president AND Lincoln AND NOT (automobile OR car)"**
  - Better, but the use of NOT will remove a document about President Lincoln that says "Lincoln's body departs Washington in a nine car funeral train"
- **Perhaps try**
  - President AND lincoln AND biography AND life AND birthplace AND gettysburg AND NOT (automobile OR car), but too many ANDs can lead to nothing, so
  - President AND lincoln AND (biography OR life OR birthplace OR gettysburg) AND NOT (automobile OR car)

USC Viterbi School of Engineering

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**

  size of the vocabulary = Dimension = t = |vocabulary|
- **A document D_i is represented by a vector of index terms**

  $$D_i = (d_{i1}, d_{i2}, \ldots, d_{it})$$
- **Where d_{ij} represents the weight of the *j*-th term in the i^{th} doc**
  - *but how is the weight computed?*
- **Both documents and queries are expressed as *t*-dimensional vectors**

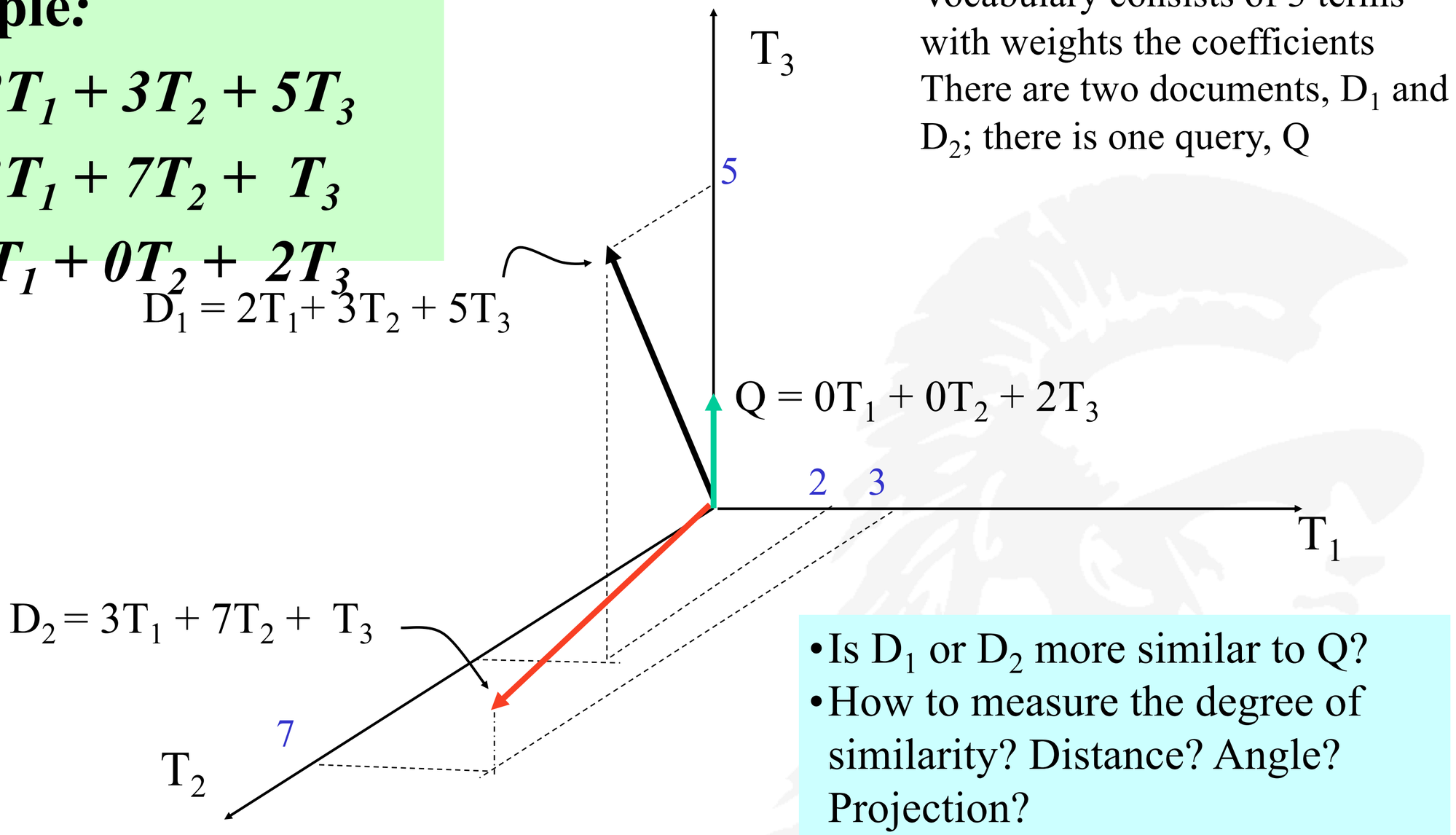
Graphic Representation Example

Example:

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

$$D_2 = 3T_1 + 7T_2 + T_3$$

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



Vocabulary consists of 3 terms with weights the coefficients
There are two documents, D_1 and D_2; there is one query, Q

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

Representing a Collection of Documents

- A collection of *n* documents can be represented in the vector space model by a term-document matrix.
- An entry in the matrix corresponds to the "weight" of a term in the document; zero means the term has no significance in the document or it simply doesn't exist in the document; but we still need a way to compute the weight

	T_1	T_2	….	T_t
D_1	w_{11}	w_{21}	…	w_{t1}
D_2	w_{12}	w_{22}	…	w_{t2}
:	:	:		:
:	:	:		:
D_n	w_{1n}	w_{2n}	…	w_{tn}

Term Weights: Term Frequency

- **One way to compute the weight is to use the term's frequency in the document**
- **Assumption: the 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*) across the entire corpus:**

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

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*

  of course df_i is always <= N (total number of documents)

  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 is used to dampen the effect relative to *tf***

An Example of Inverse Document Frequency

term	df_i	idf_i
Calpurnia	1	log(1,000,000/1)=6
animal	100	4
Sunday	1,000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

- $idf_i = log_{10}(N/df_i)$, $N = 1{,}000{,}000$
- there is one *idf* value for each term *t* in a collection

TF.IDF Weighting

- **A typical combined term importance indicator is *tf-idf weighting* (*note: it is often written with a hyphen, but the hyphen is NOT a minus sign; some people replace the hyphen with a dot*):**

$$w_{ij} = tf_{ij}.idf_i = (1 + log\ tf_{ij}) * \log_2 (N/ df_i)$$

- **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**

- **Given a query *q*, then we score the query against a document *d* using the formula**

- $Score\ (q, d) = \sum(tf.idf_{t,d})$ **where *t* is in $q \cap d$**

Computing TF.IDF -- An Example

Given a document containing 3 terms with given frequencies:

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

Assume collection contains 10,000 documents and document frequencies of these 3 terms are:

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

Then:

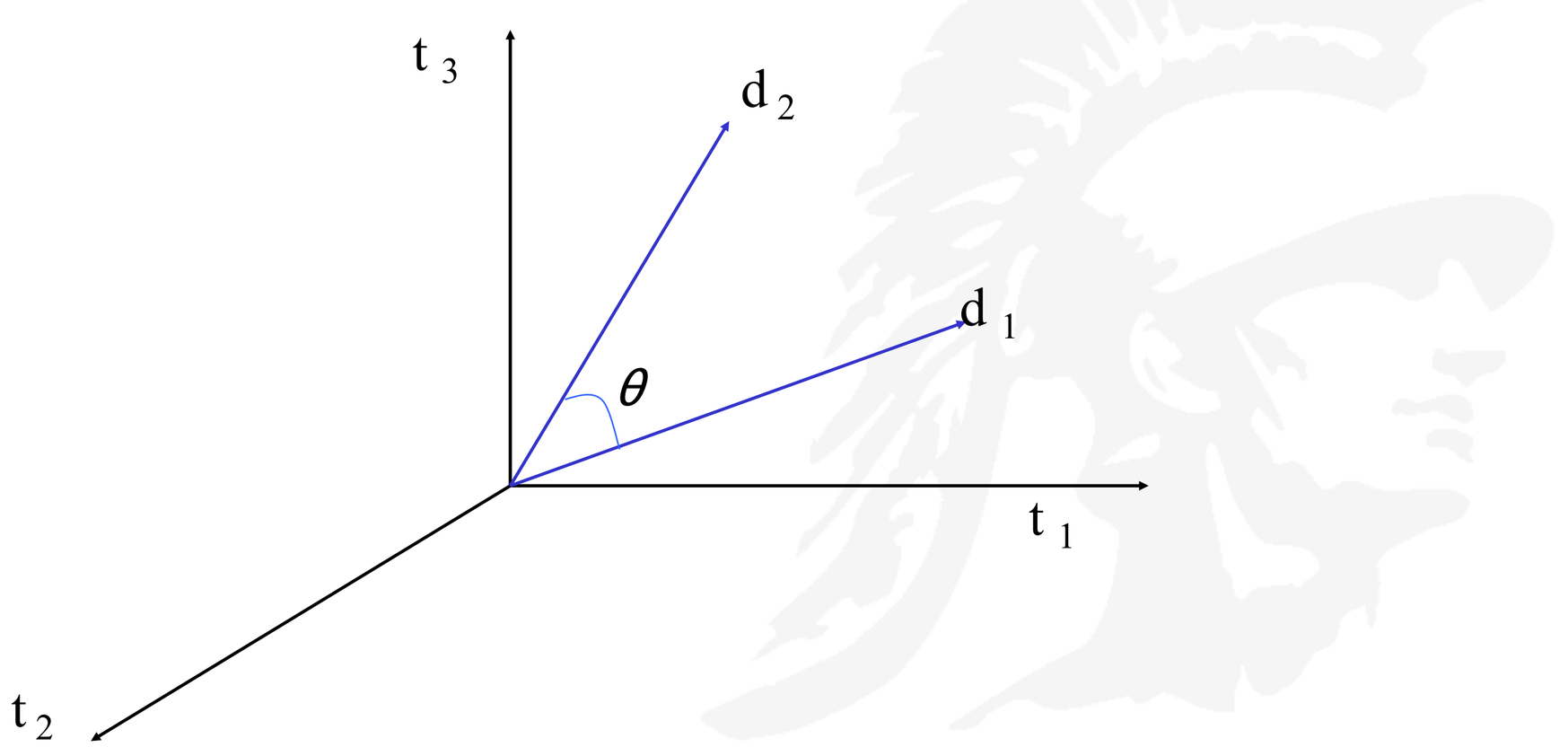
A: tf = 3/3; idf = log(10000/50) = 5.3; tf.idf = 5.3

B: tf = 2/3; idf = log(10000/1300) = 2.0; tf.idf = 1.3

C: tf = 1/3; idf = log(10000/250) = 3.7; tf.idf = 1.2

Cosine Similarity

- **Distance between vectors d_1 and d_2 is *captured* by the cosine of the angle x between them.**
- **Note – this is a *similarity* measure, not a distance measure**



Similarity Measure

- **A similarity measure is a function that computes the *degree of similarity* between two vectors**
  - **Look back at the previous lecture slides for the definition of similarity**
- **Using a similarity measure between the query and each document has positive aspects:**
  - It is possible to rank the retrieved documents in the order of presumed relevance.
  - It is possible to enforce a certain threshold so that the size of the retrieved set can be controlled.

Normalized Vectors

- A vector can be normalized (given a length of 1) by dividing each of its components by the vector's length
- This maps vectors onto the unit circle:
- Then, $\left|\vec{d}_j\right| = \sqrt{\sum_{i=1}^{n} w_{i,j}} = 1$
- Longer documents don't get more weight
- For normalized vectors, the cosine is simply the dot product:

$$\cos(\vec{d}_j, \vec{d}_k) = \vec{d}_j \cdot \vec{d}_k$$

Similarity Measure for Document and Query

- **Similarity between vectors for the document d_j and query q can be computed as the vector inner product:**

$$sim(d_j,q) = d_j \bullet q = \sum_{i=1}^{t} w_{ij} \cdot w_{iq}$$

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.**

Limitations of the Inner Product

- **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 Using Inner Product -- Examples

Binary:

	retrieval	database	architecture	computer	text	management	information
– D =	1,	1,	1,	0,	1,	1,	0
– Q =	1,	0 ,	1,	0,	0,	1,	1

Size of vector = size of vocabulary = 7
0 means corresponding term not found in document or query

similarity(D, Q) = 3 **(the inner product)**

Weighted:

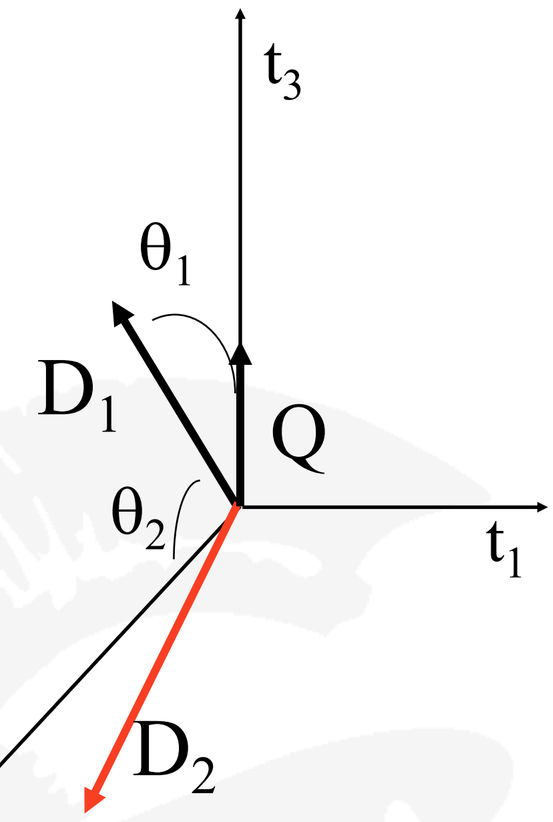
$D_1 = 2T_1 + 3T_2 + 5T_3 \qquad D_2 = 3T_1 + 7T_2 + 1T_3$
$Q = 0T_1 + 0T_2 + 2T_3$

$sim(D_1 , Q) = 2*0 + 3*0 + 5*2 = 10$
$sim(D_2 , Q) = 3*0 + 7*0 + 1*2 = 2$

Cosine Similarity Measure Normalized

- **Cosine similarity measures the cosine of the angle between two vectors**
- **We compute the inner product normalized by the vector lengths**

$$CosSim(d_j, q) = \frac{\vec{d_j} \cdot \vec{q}}{|\vec{d_j}| \cdot |\vec{q}|} = \frac{\sum_{i=1}^{t} (w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$



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

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

Naïve Implementation

1. Convert all documents in collection D to *tf.idf* weighted vectors, the j^{th} document denoted by d_j, for keywords in vocabulary V
2. Convert each query to a *tf.idf* weighted vector *q*
3. For each d_j in D do
   Compute score $s_j = cosSim(d_j, q)$
4. Sort documents by decreasing score
5. 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

Efficient Cosine Ranking

- **Ranking consists of computing the *k* docs in the corpus "nearest" to the query ⇒ *k* largest query-doc cosines.**
- **To do efficient ranking one must:**
  - **Compute a single cosine efficiently.**
  - **Choose the *k* largest cosine values efficiently.**

Computing Cosine Scores

cosineScore(q)

1. **float** ***Scores[N] = 0;*** ***//Scores array for all documents***
2. **float** ***Length[N]*** ***//lengths of all documents***
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 do***
9. ***Scores[d] = Scores[d]/Length[d]***
10. **return Top** ***K*** **components of** ***Scores[]***

in practice we only work with a subset of the documents

weight of query term is *1*;
then,
for each document in the postings list, the term *t* occurs *tf* times;
then we take the dot product of weight of term *t* in document times weight of term *t* in query;

divide scores by length of each document

ranking

Summary of Algorithm for Vector Space Ranking

- Represent the query as a weighted *tf.idf* vector
- represent each document as a weighted *tf.idf* vector
- compute the cosine similarity score for the query vector and each document vector that contains the query term
- Rank documents with respect to the query by score
- Return the top *k* (e.g. *k=10*) to the user

USC Viterbi School of Engineering Use Heap for Selecting Top *k*

- **Binary tree in which each node's value > values of children**
- **Takes *2n* operations to construct, then each of *k* log *n* "winners" read off in 2log *n* steps.**
- **For *n=1M*, *k=100*, this is about 10% of the cost of sorting.**

Bottleneck

- Still need to first compute cosines from query to each of *n* docs → several seconds for *n* = 1M
- Can select from only non-zero cosines
  - Need union of postings lists accumulators (<<1M)
- Can further limit to documents with non-zero cosines on rare (high idf) words
- Enforce conjunctive search (a la Google): non-zero cosines on *all* words in query
  - Need min of postings lists sizes accumulators
- But still potentially expensive

A Pre-Processing Strategy

- **Preprocess: Pre-compute, for each term, its *k* nearest docs.**
  - **(Treat each term as a 1-term query)**
  - **lots of preprocessing.**
  - **Result: "preferred list" for each term.**
- **Search:**
  - **For a *t*-term query, take the union of their *t* preferred lists – call this set *S*.**
  - **Compute cosines from the query to only the docs in *S*, and choose top *k*.**

Comments on Vector Space Model

- **Simple, mathematically based approach.**
- **Considers both local (*tf*) and global (*idf*) word occurrence frequencies.**
- **Provides partial matching and ranked results.**
- **Tends to work quite well in practice despite obvious weaknesses.**
- **Allows efficient implementation for large document collections.**

Problems with Vector Space Model

- **Missing semantic information (e.g. word sense).**
- **Missing syntactic information (e.g. phrase structure, word order, proximity information).**
- **Assumption of term independence**
- **Lacks the control of a Boolean model (e.g., *requiring* a term to appear in a document).**
  - Given a two-term query "A B", may prefer a document containing A frequently but not B, over a document that contains both A and B, but both less frequently.