Boolean and Vector Space Retrieval Models

- CS 293S, 2017
- Some of slides from R. Mooney (UTexas), J. Ghosh (UT ECE), D. Lee (USTHK).

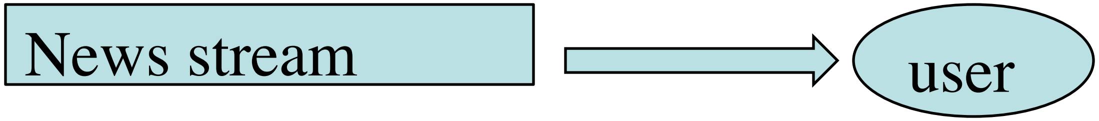
Table of Content

Which results satisfy the query constraint?

- **Boolean model**
- **Statistical vector space model**

Retrieval Tasks

- **Ad hoc retrieval: Fixed document corpus, varied queries.**
- **Filtering: Fixed query, continuous document stream.**
  - User Profile: A model of relative static preferences.
  - Binary decision of relevant/not-relevant.



- **Routing: Same as filtering but continuously supply ranked lists rather than binary filtering.**

Retrieval Models

- **A retrieval model specifies the details of:**
  - 1) Document representation
  - 2) Query representation
  - 3) Retrieval function: how to find relevant results
  - Determines a notion of relevance.
- **Classical models**
  - Boolean models (set theoretic)
    - Extended Boolean
  - Vector space models (statistical/algebraic)
    - Generalized VS
    - Latent Semantic Indexing
  - Probabilistic models

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.**
  - Rio & Brazil | Hilo & Hawaii
  - hotel & !Hilton
- **Output: Document is relevant or not. No partial matches or ranking.**
  - Can be extended to include ranking.
- **Popular retrieval model in old time:**
  - Easy to understand. Clean formalism.
  - But still too complex for web users

Query example: Shakespeare plays

- **Which plays of Shakespeare contain the words *Brutus AND Caesar* but *NOT Calpurnia*?**
- **Could grep all of Shakespeare's plays for *Brutus* and *Caesar,* then strip out lines containing *Calpurnia*?**
  - Slow (for large corpora)
  - *NOT* ***Calpurnia*** is non-trivial
  - Other operations (e.g., find the phrase ***Romans and countrymen***) not feasible

Term-document incidence

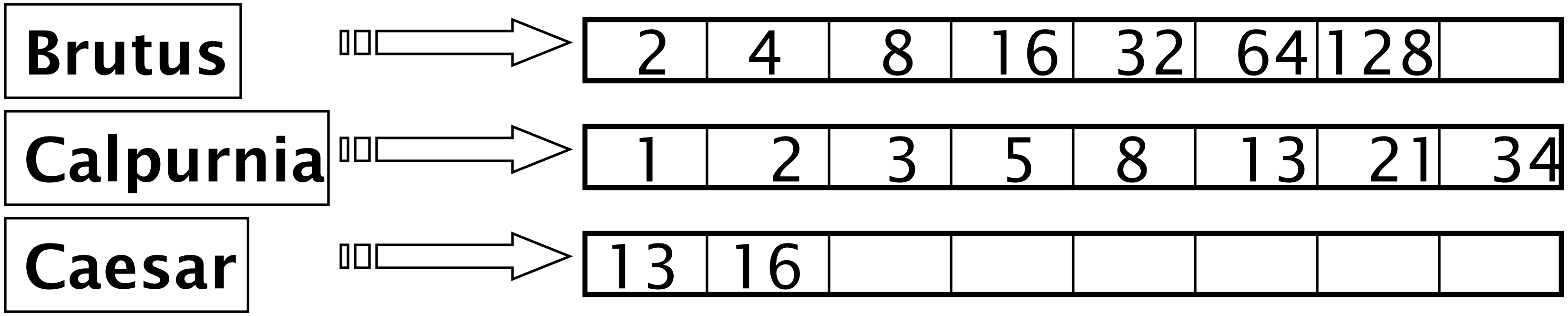
1 if play contains word, 0 otherwise

	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

- Incident vectors: 0/1 vector for each term.
- Query answer with bitwise operations (AND, negation, OR):
  - Which plays of Shakespeare contain the words *Brutus AND Caesar* but *NOT Calpurnia*?
  - 110100 *AND* 110111 *AND* 101111 = 100100.

Inverted index

- For each term *T*, must store a list of all documents that contain *T*.

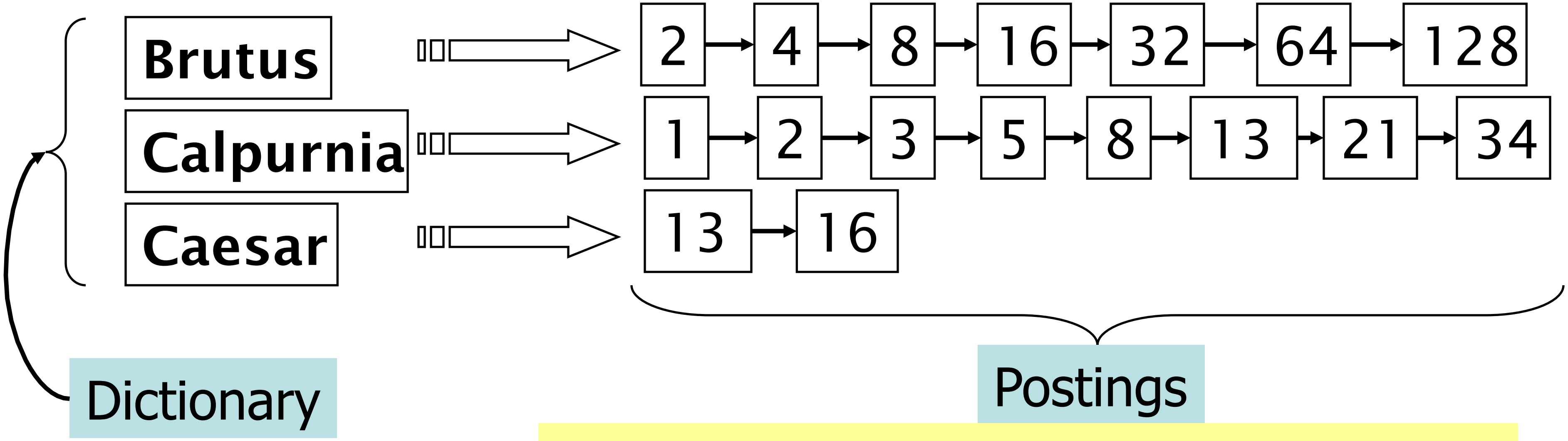


Brutus	2	4	8	16	32	64	128	
Calpurnia	1	2	3	5	8	13	21	34
Caesar	13	16						

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

Inverted index

- **Linked lists generally preferred to arrays**
  - Dynamic space allocation
  - Insertion of terms into documents easy
  - Space overhead of pointers

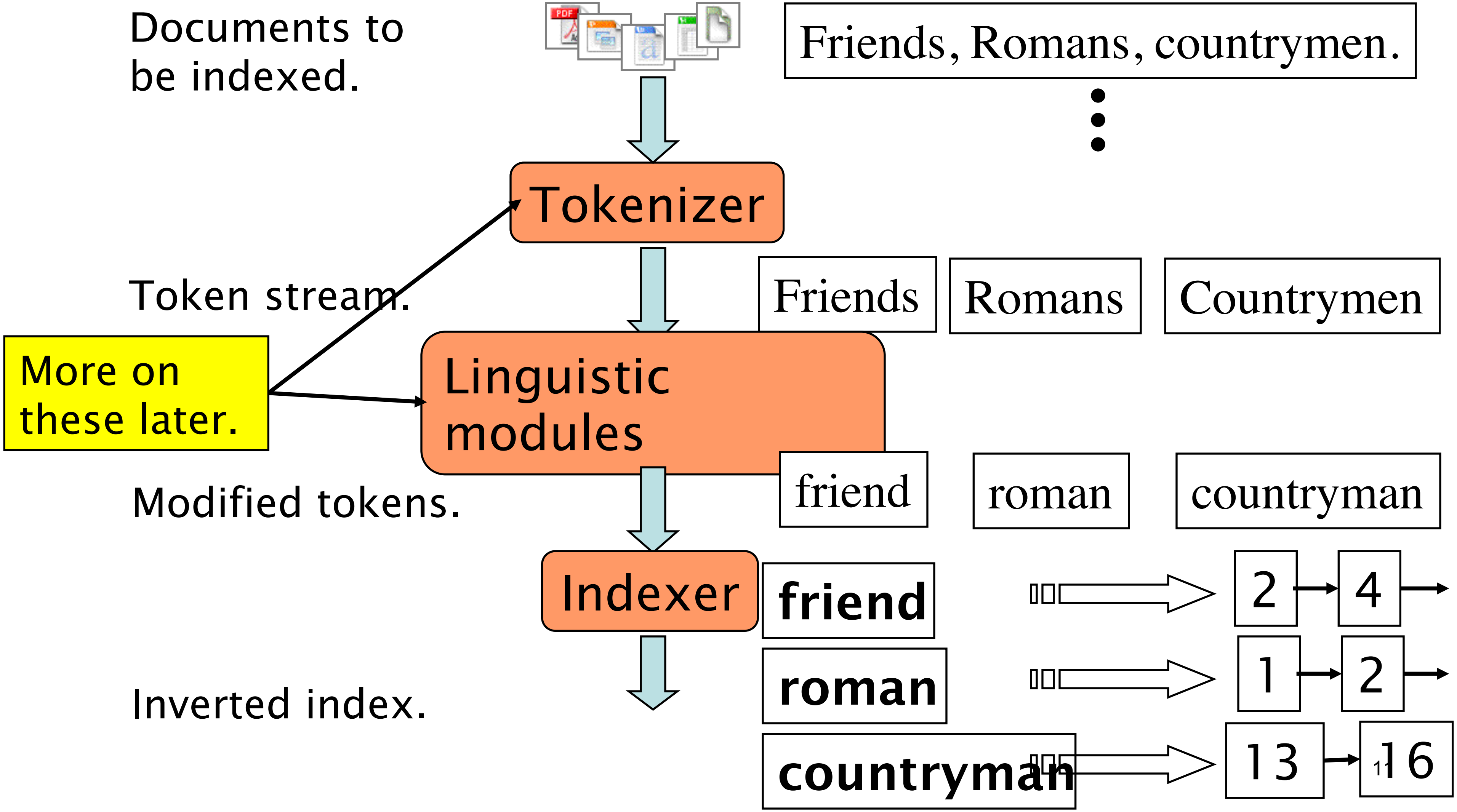


Sorted by docID (more later on why).

Possible Document Preprocessing Steps

- Strip unwanted characters/markup (e.g. HTML tags, punctuation, numbers, etc.).
- Break into tokens (keywords) on whitespace.
- Possible linguistic processing (used in some applications, but dangerous for general web search)
  - Stemming (cards ->card)
  - Remove common stopwords (e.g. a, the, it, etc.).
  - Used sometime, but dangerous
- Build inverted index
  - keyword → list of docs containing it.
  - Common phrases may be detected first using a domain specific dictionary.

Inverted index construction



Documents to be indexed.

Friends, Romans, countrymen.

Tokenizer

Token stream.

Friends Romans Countrymen

More on these later.

Linguistic modules

Modified tokens.

friend roman countryman

Indexer

Inverted index.

friend 2 4

roman 1 2

countryman 13 16

Discussions

- **Index construction**
  - Stemming?
  - Which terms in a doc do we index?
    - All words or only “important” ones?
    - Stopword list: terms that are so common
      - they MAY BE ignored for indexing.
      - *e.g.*, ***the, a, an, of, to*** …
      - language-specific.
      - May have to be included for general web search
- **How do we process a query?**
  - Stop word removal
    - Where is UCSB?
  - Stemming?

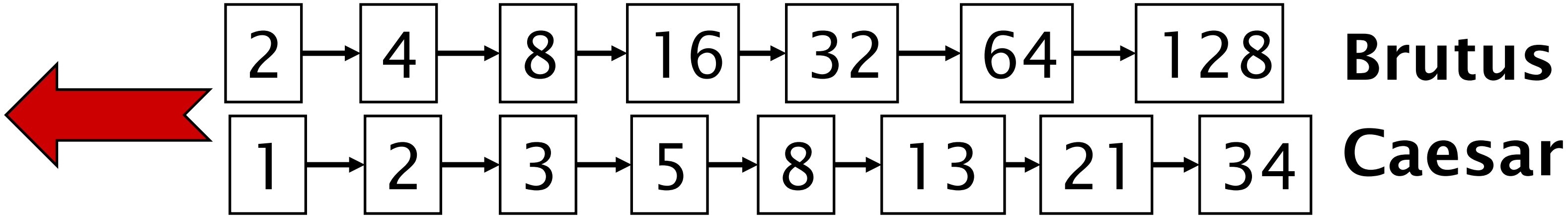
Dataset	Small	Big
Offline	Stemming	Less or no stemming
Online	Stemming Stopword removal	Less or no stemming Stopword removal

Query processing

- **Consider processing the query:**

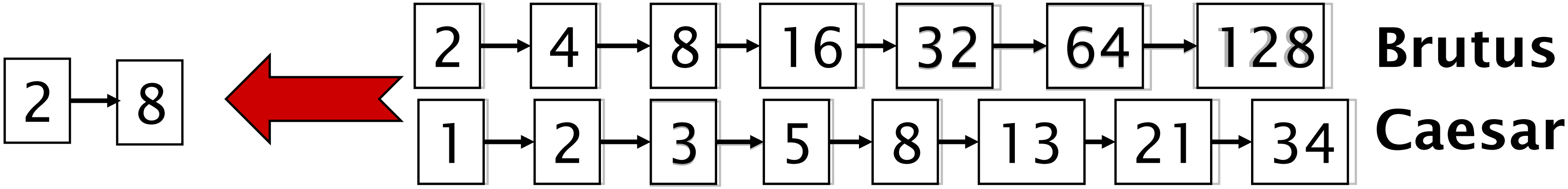
  Brutus *AND* ***Caesar***

  - Locate ***Brutus*** in the Dictionary;
    - Retrieve its postings.
  - Locate *Caesar* in the Dictionary;
    - Retrieve its postings.
  - "Merge" the two postings:



The merge

- **Walk through the two postings simultaneously, in time linear in the total number of postings entries**



If the list lengths are m and n, the merge takes O(m+n) operations.

Crucial: postings sorted by docID.

Example: WestLaw http://www.westlaw.com/

- **Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992)**
- **Majority of users *still* use boolean queries**
- **Example query:**
  - What is the statute of limitations in cases involving the federal tort claims act?
  - LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM
- **Long, precise queries; proximity operators; incrementally developed; not like web search**
  - Professional searchers (e.g., Lawyers) still like Boolean queries:
  - You know exactly what you’re getting.

More general merges

- Exercise: Adapt the merge for the queries:

  Brutus AND NOT Caesar

  Brutus OR NOT Caesar

Can we still run through the merge in time O(*m*+*n*)?

Boolean Models – Problems

- **Very rigid: AND means all; OR means any.**
- **Difficult to express complex user requests.**
    - Still too complex for general web users
- **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?

Statistical Retrieval 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.
- 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:

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

$$\left(\begin{array}{ccccc} & T_1 & T_2 & \ldots. & T_t \\ D_1 & w_{11} & w_{21} & \ldots & w_{t1} \\ D_2 & w_{12} & w_{22} & \ldots & w_{t2} \\ : & : & : & & : \\ : & : & : & & : \\ D_n & w_{1n} & w_{2n} & \ldots & w_{tn} \end{array}\right)$$

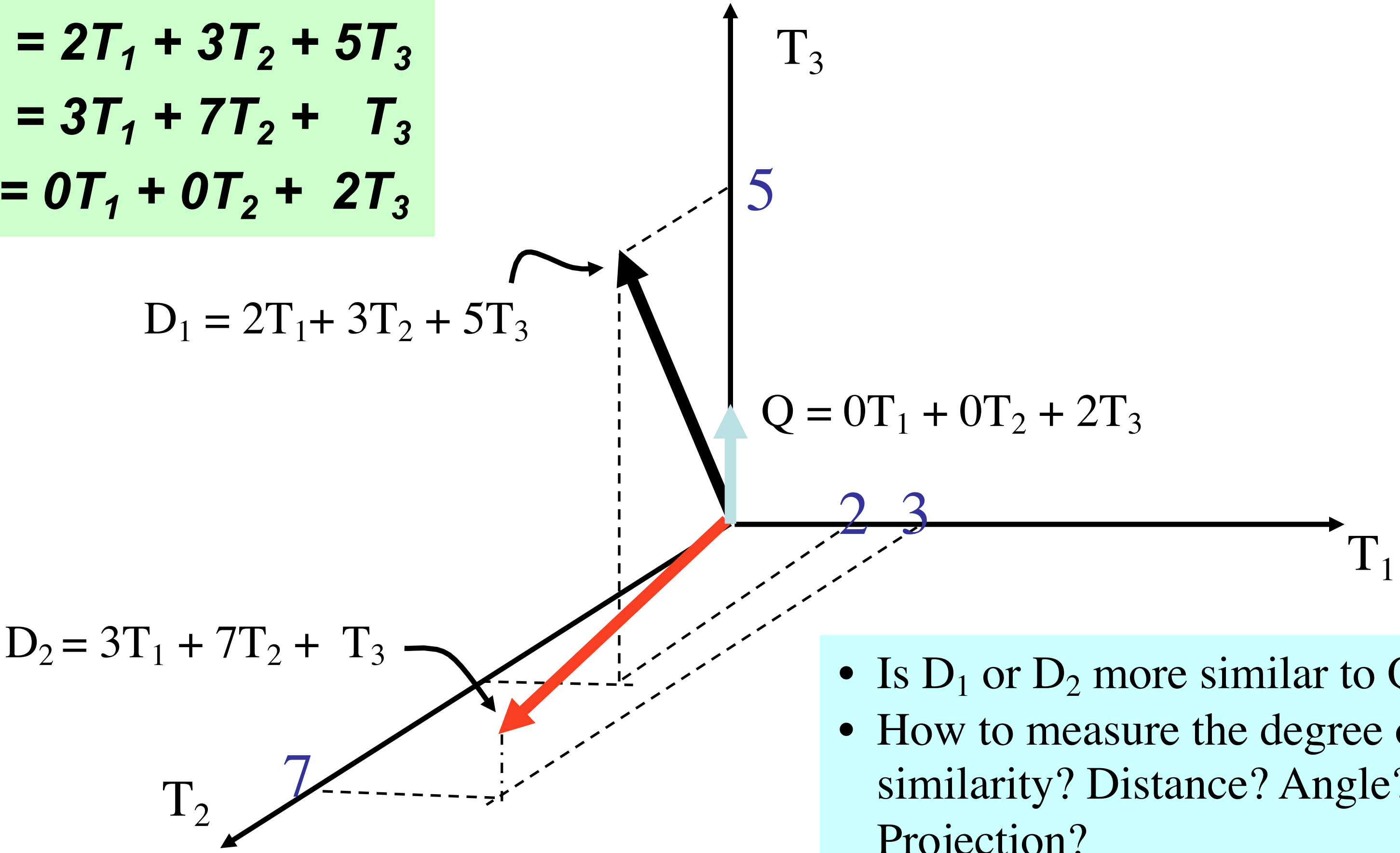
Graphic Representation

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 n-grams (and phrases, idioms,…) → 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 a 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.**

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

- **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*

  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 with term 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(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

Similarity Measure

- A **similarity measure** is a function that computes the ***degree of similarity*** between two vectors.
- Using a similarity measure between the query and each document:

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

$$\text{sim}(d_j, q) = d_j \bullet q = \qquad w_{ij} \cdot w_{iq}$$

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

Inner Product -- Examples

Binary:

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

sim(D, Q) = 3

Weighted:

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

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

$\text{sim}(D_1 , Q) = 2*0 + 3*0 + 5*2 = 10$

$\text{sim}(D_2 , Q) = 3*0 + 7*0 + 1*2 = 2$

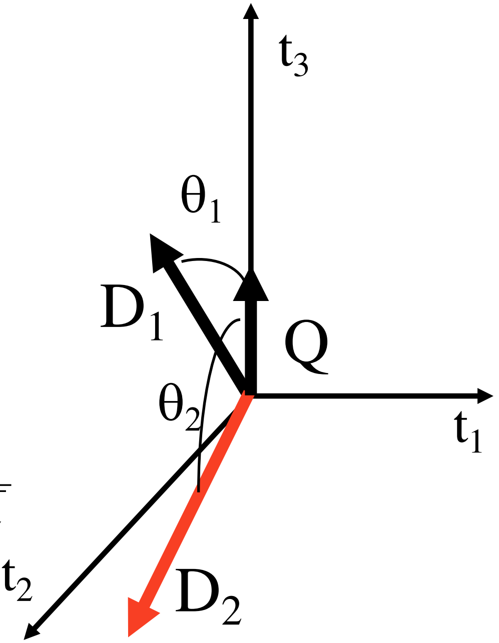
Properties of Inner Product

- 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.**

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



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

Comments on Vector Space Models

- **Simple, practical, and mathematically based approach**
- **Provides partial matching and ranked results.**
- **Problems**
  - Missing syntactic information (e.g. phrase structure, word order, proximity information).
  - Missing semantic information
    - word sense
    - Assumption of term independence. ignores synonomy.
  - 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.