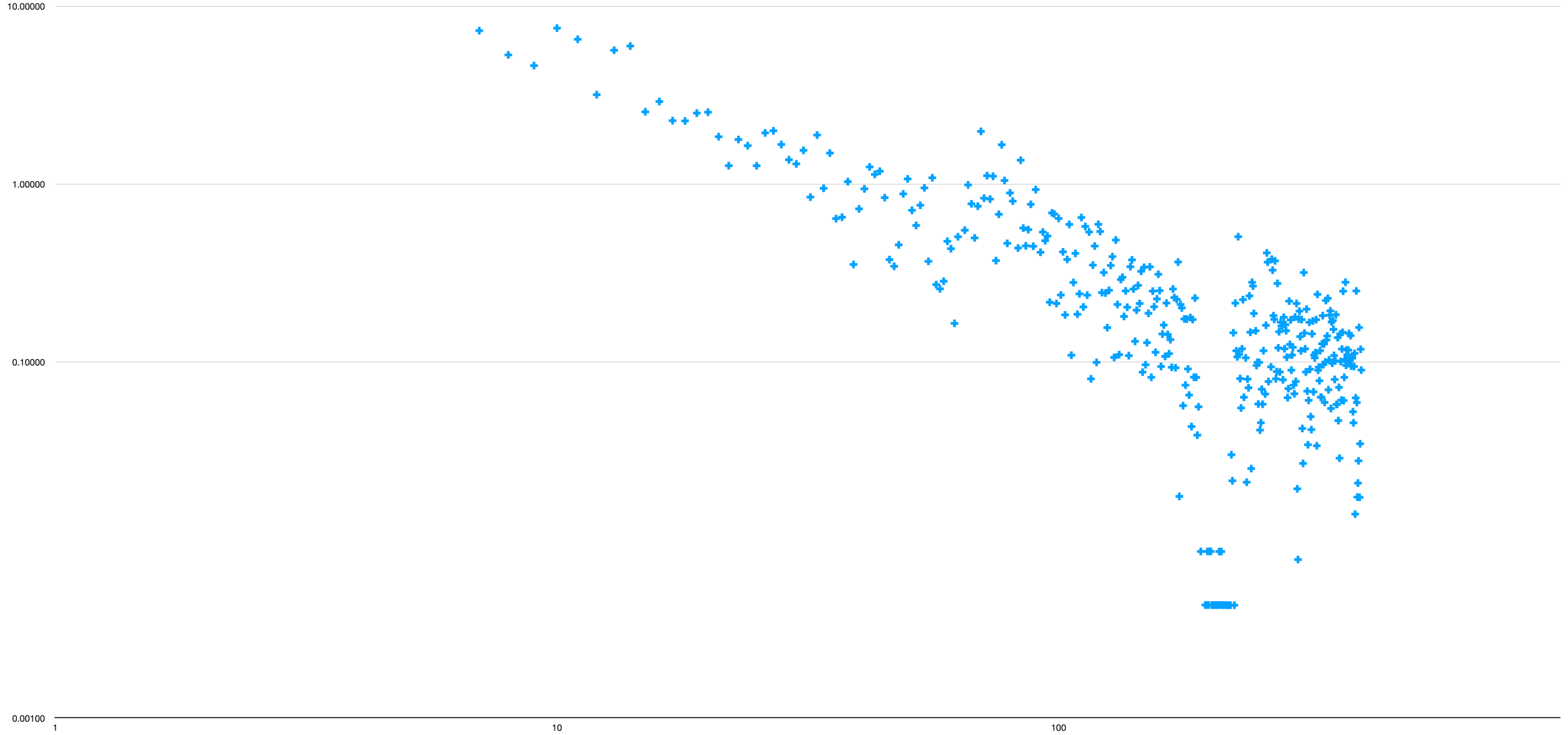
Information Retrieval 2

Databases of text

Dickens & Heaps' Law

Omitting the first few documents



Frequent Words

Frequent Word	Number of Occurrences	Percentage of Total
the	7,398,934	5.9
of	3,893,790	3.1
to	3,364,653	2.7
and	3,320,687	2.6
in	2,311,785	1.8
is	1,559,147	1.2
for	1,313,561	1
The	1,144,860	0.9
that	1,066,503	0.8
said	1,027,713	0.8

Frequencies from 336,310 documents in the 1 GB TREC Volume 3 Corpus

- 125,720,891 total word occurrences
- 508,209 unique words

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.
- Is bag-of-words a good model???
- Leacock (personal communication)Princeton students
  - better with sentences than with alphabetizd lists
    - Conclusion: there is information lost in the “bag of words”

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.

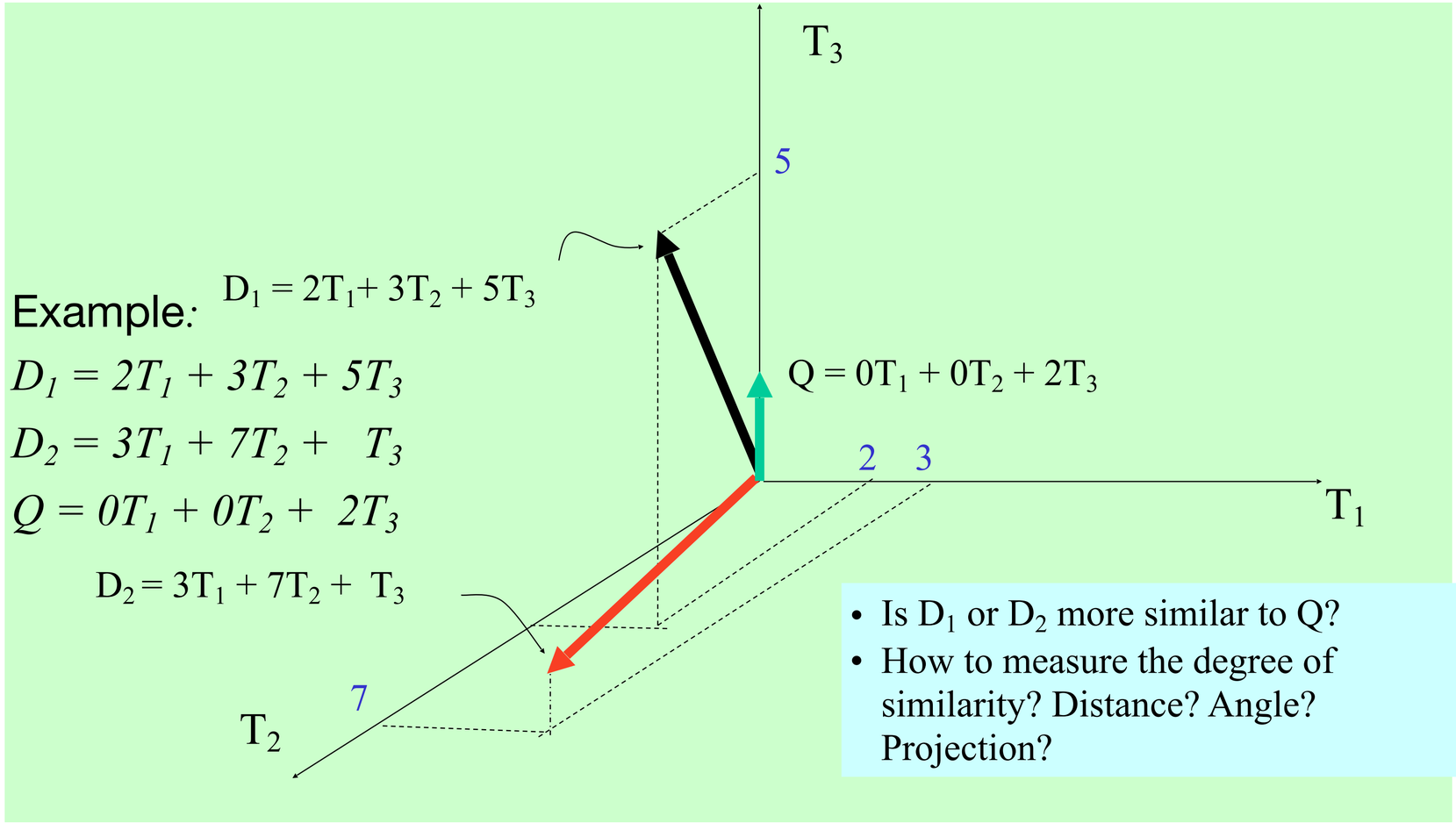
  Dimensionality = t = |vocabulary|

- 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



Issues for Vector Space Model

- How to determine important words in a document?
  - Word sense?
  - 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 the collection and what are the effects of links, formatting information, etc.?

Vector Space

- Simplest Approach
  - Represent the presence of a word in a document with just a 1 in the sport corresponding to the word
- Problems:
  - strongly favors large documents
  - Documents will all be fairly similar because the all have the, a, or, ... and these will tend to dominate

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:

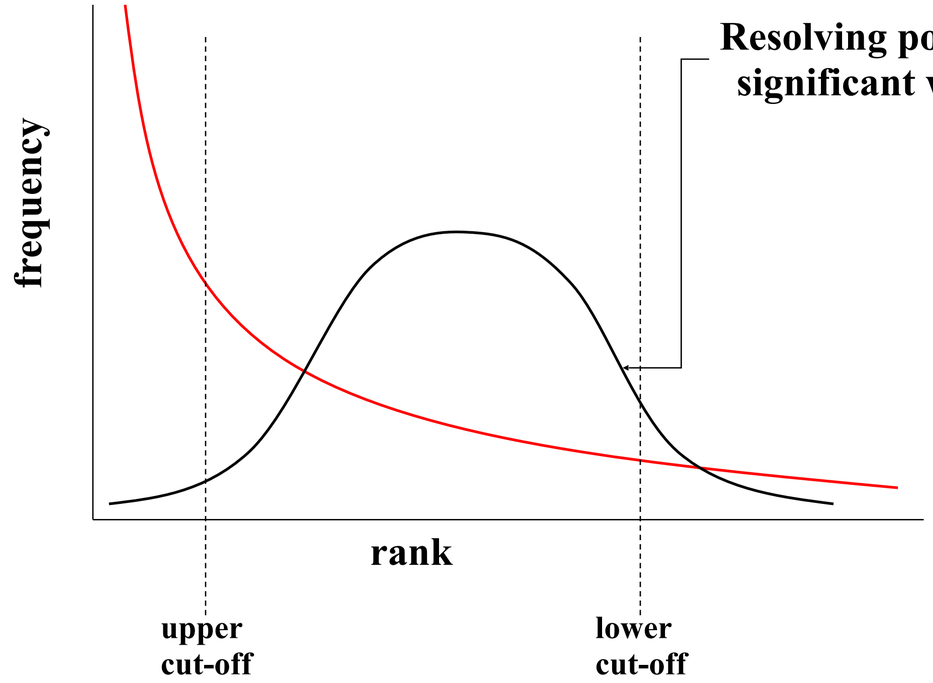
  $tf_{ij} = f_{ij} / max_i\{f_{ij}\}$

Zipf's Law and Indexing

- The most frequent words are poor index terms
  - they occur in almost every document
  - they usually have no relationship to the concepts and ideas represented in the document
- Extremely infrequent words are poor index terms
  - may be significant in representing the document
  - but, very few documents will be retrieved when indexed by terms with low frequency
- Index terms in between
  - a high and a low frequency threshold are set
  - only terms within the threshold limits are considered good candidates for index terms

Resolving Power

- Zipf (and later H.P. Luhn) postulated that the *resolving power of significant words* reached a peak at a rank order position half way between the two cut-offs
  - Resolving Power: the ability of words to discriminate content



The actual cut-off are determined by trial and error, and often depend on the specific collection.

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*:

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

where i is the index of the document and j is the intex of the term

- 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 tokens with 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.

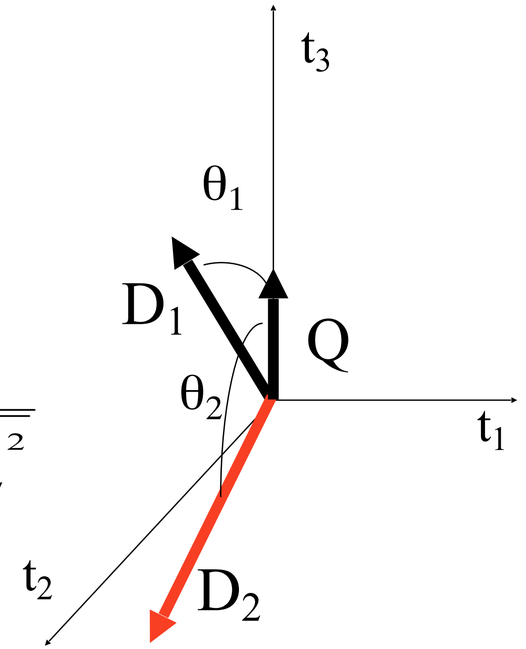
Properties of Inner Product

- 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.

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



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

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

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

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