Inspire…Educate…Transform.

Engineering Big Data

Text Mining: Preprocessing

Dr. Sreerama KV Murthy
CEO, Teqnium Consultancy Services

Jan 18, 2014

SOME FOUNDATIONS OF TEXT MINING

Questions

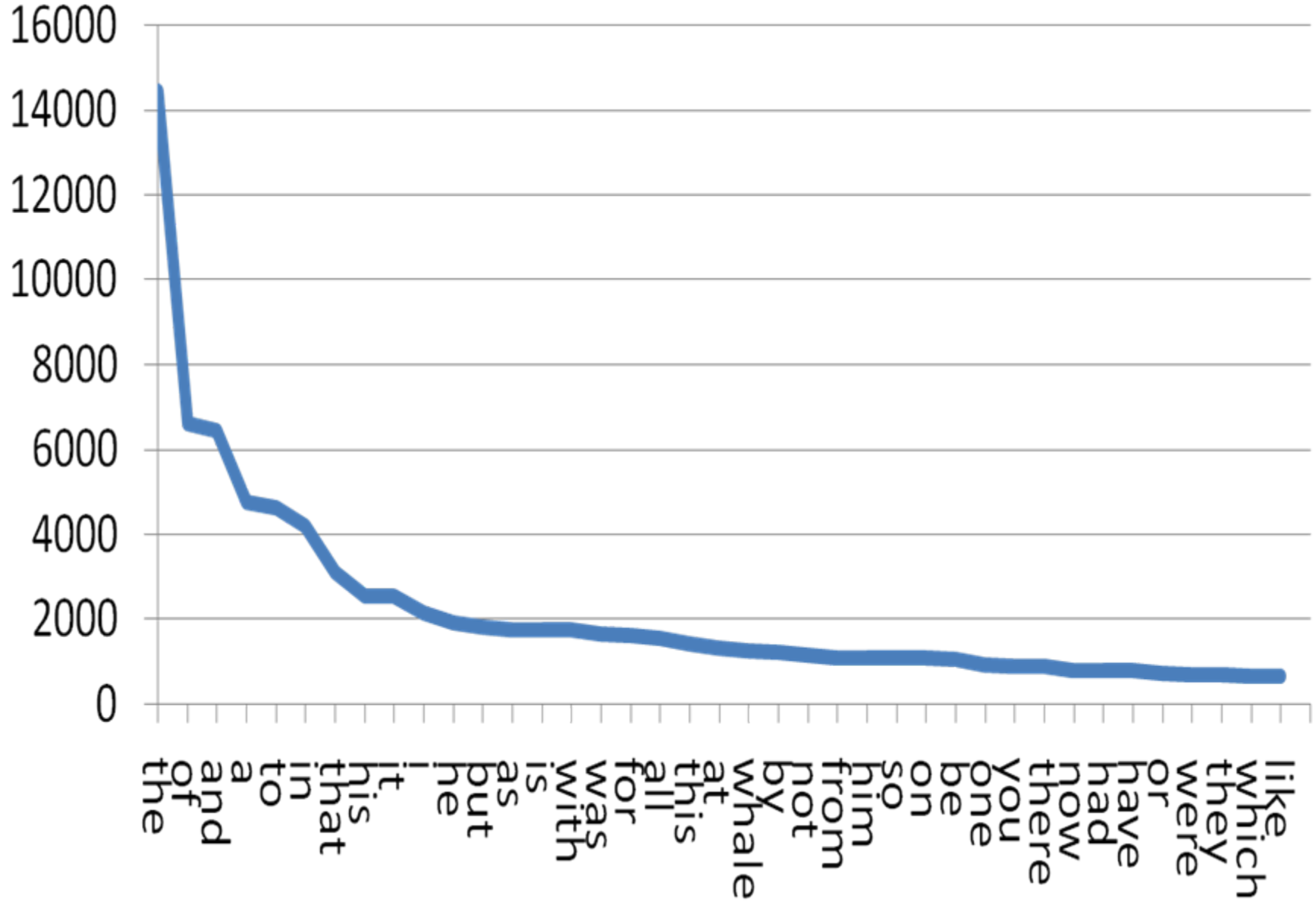
1. How is the frequency of different words distributed?

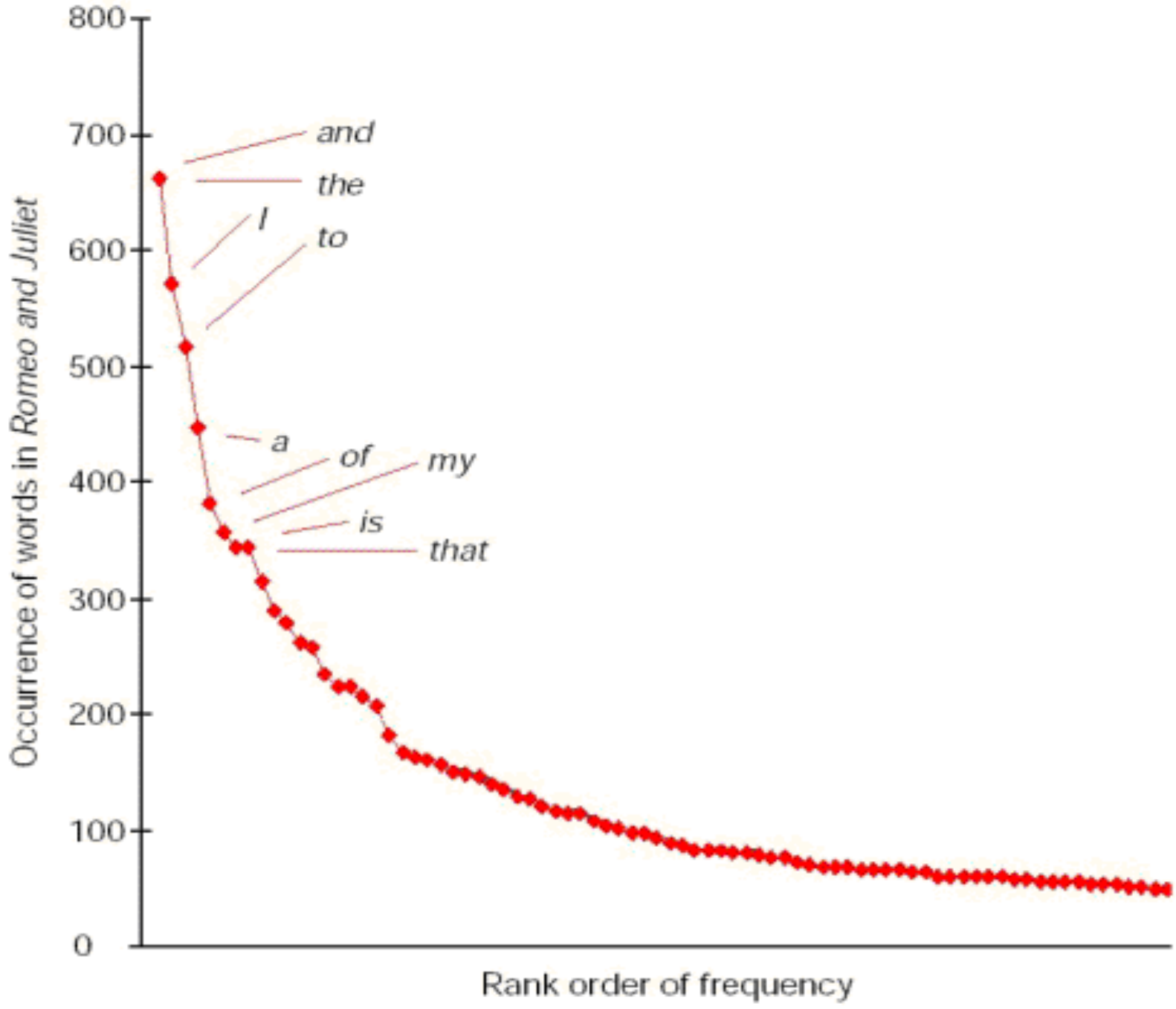
2. How fast does vocabulary size grow with the size of a corpus?

3. Are these properties corpus dependent or corpus independent?

- Factors like these affect the performance of information retrieval.

“Moby Dick” by Herman Melville, 1851




800
700
600
500
400
300
200
100
0
Occurrence of words in Romeo and Juliet
and
the
I
to
a
of
my
is
that
Rank order of frequency

Sample Word Frequency Data
(from B. Croft, UMass)

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	1.0
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 1GB TREC Volume 3 Corpus
125,720,891 total word occurrences; 508,209 unique words

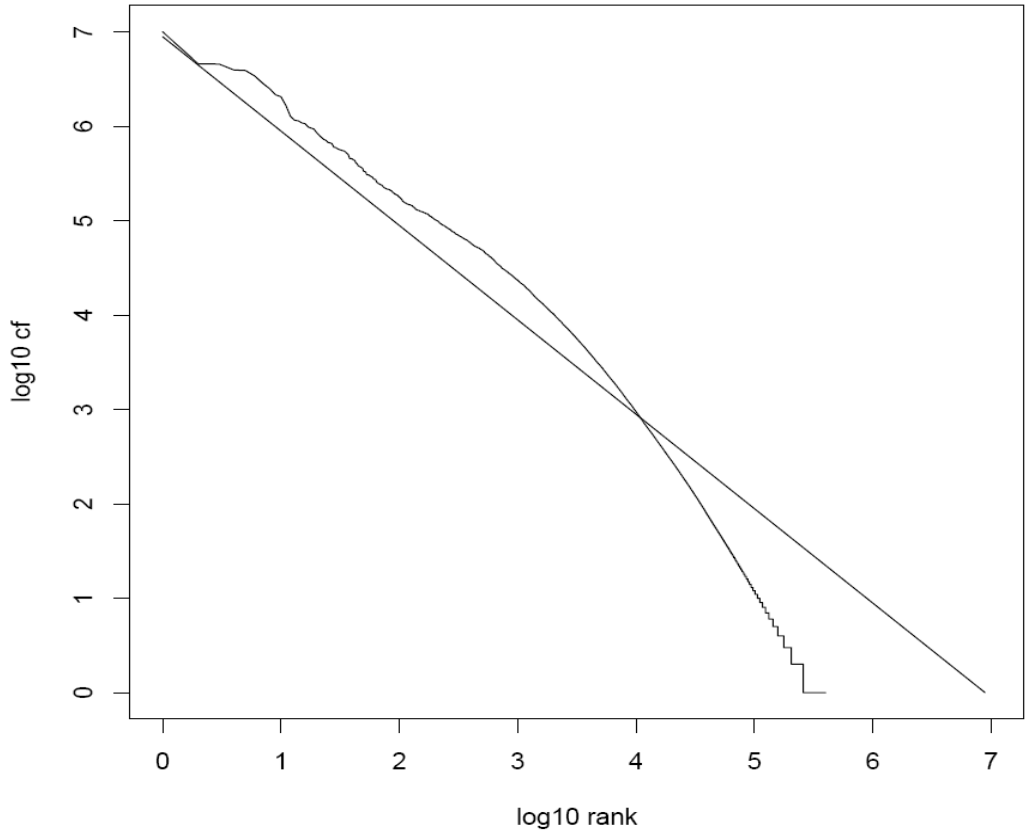
Postings Size: Zipf's Law (1949)

$$\mathrm{cf}_i = \frac{c}{i}$$

cf is the collection frequency of *i*-th common term
c is a constant

- Zipf's Law: (also) linear in log-log space
  - Specific case of Power Law distributions
- In other words:
  - A few elements occur very frequently
  - Many elements occur very infrequently

Zipf's Law for RCV1



Fit isn't that good… but good enough!

Reuters-RCV1 collection: 806,791 newswire documents (Aug 20, 1996-August 19, 1997)

Manning, Raghavan, Schütze, Introduction to Information Retrieval (2008)

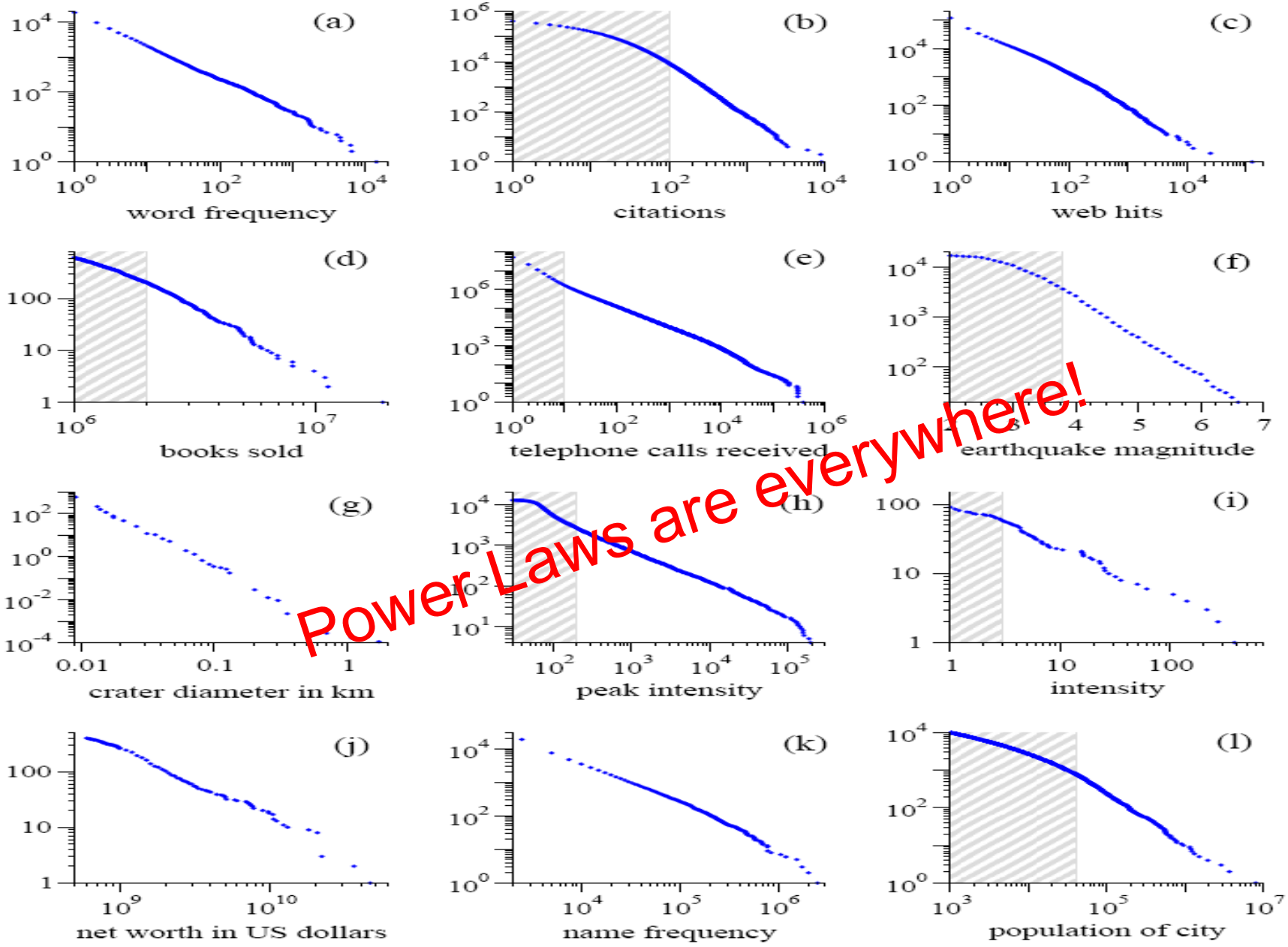


Figure from: Newman, M. E. J. (2005) “Power laws, Pareto distributions and Zipf's law.” Contemporary Physics 46:323–351.

Zipf and Term Weighting

Luhn (1958) suggested that both extremely common and extremely uncommon words were **not very useful** for indexing.

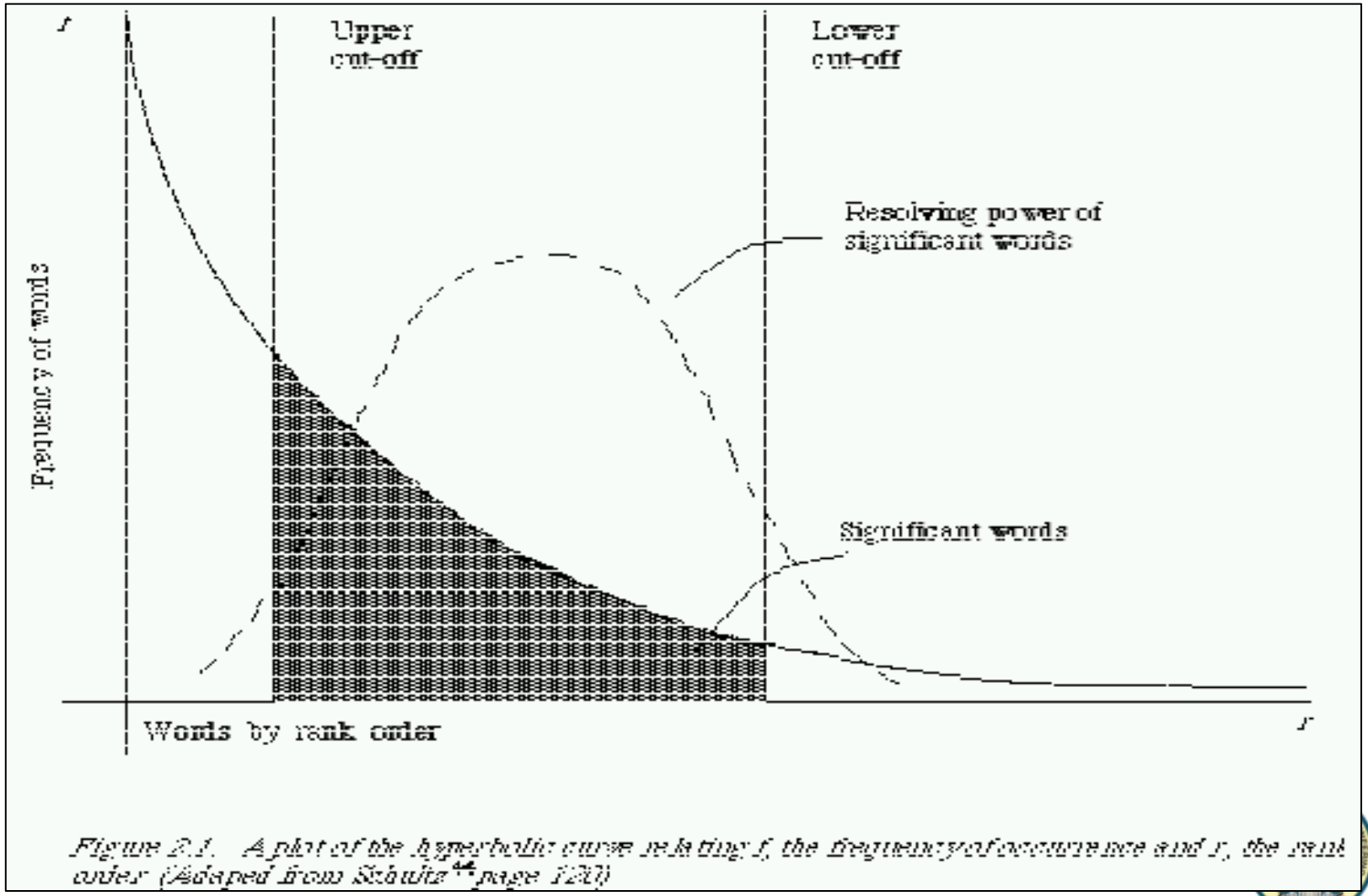


Figure 2.1. A plot of the hyperbolic curve relating f, the frequency of occurrence and r, the rank order (Adapted from Schultz[44] page 120)

Text Property 1: Word Frequency

- A few words are very common.
  - 2 most frequent words (e.g. “the”, “of”) can account for about 10% of word occurrences.
- Most words are very rare.
  - Half the words in a corpus appear only once, called *hapax legomena* (Greek for “read only once”)
- Called a “heavy tailed” distribution, since most of the probability mass is in the “tail”

Explanations for Zipf's Law

- Zipf's explanation was his "principle of least effort." Balance between speaker's desire for a small vocabulary and hearer's desire for a large one.

- Li (1992) shows that just random typing of letters including a space will generate "words" with a Zipfian distribution.
  - http://linkage.rockefeller.edu/wli/zipf/

Zipf's Law Impact on TRM

- Good News: Stopwords will account for a large fraction of text so eliminating them greatly reduces inverted-index storage costs.

- Bad News: For most words, gathering sufficient data for meaningful statistical analysis (e.g. for correlation analysis for query expansion) is difficult since they are extremely rare.

Text Property 2: Vocabulary Growth

- How does the size of the overall vocabulary (number of unique words) grow with the size of the corpus?
- This determines how the size of the inverted index will scale with the size of the corpus.
- Vocabulary not really upper-bounded due to proper names, typos, etc.

Vocabulary Size: Heaps' Law

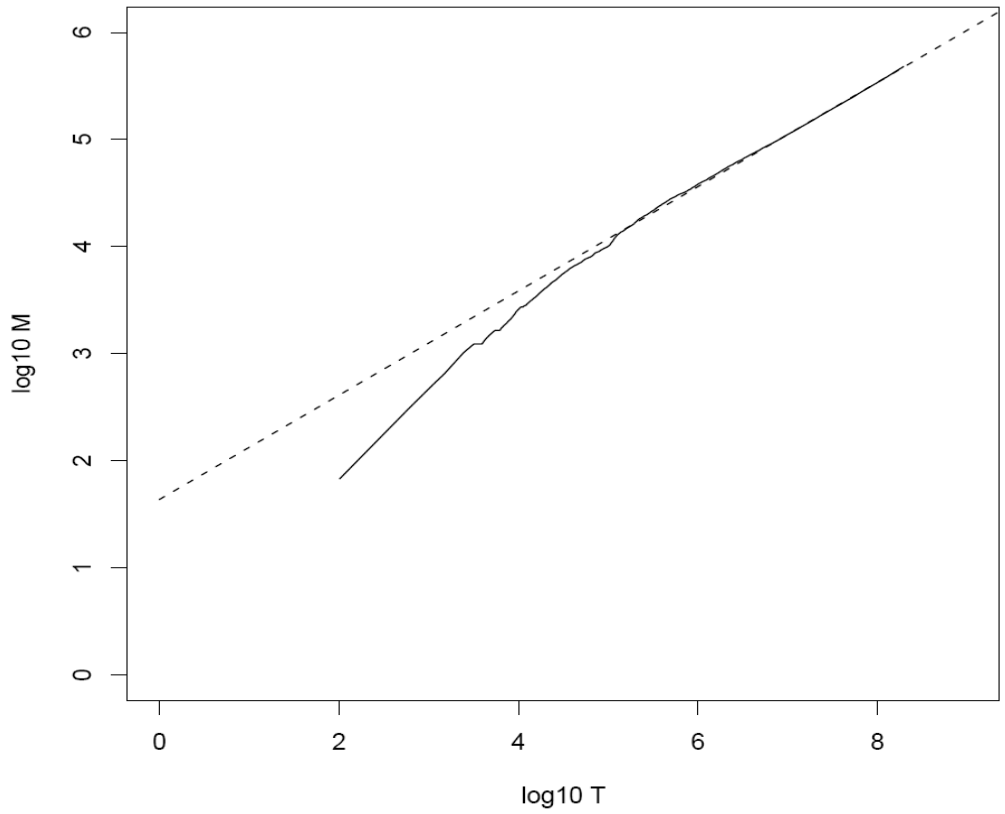
$$M = kT^b$$

M is vocabulary size
T is collection size (number of documents)
k and *b* are constants

Typically, *k* is between 30 and 100, *b* is between 0.4 and 0.6

- Heaps' Law: linear in log-log space
- Vocabulary size grows unbounded!

Heaps' Law for RCV1



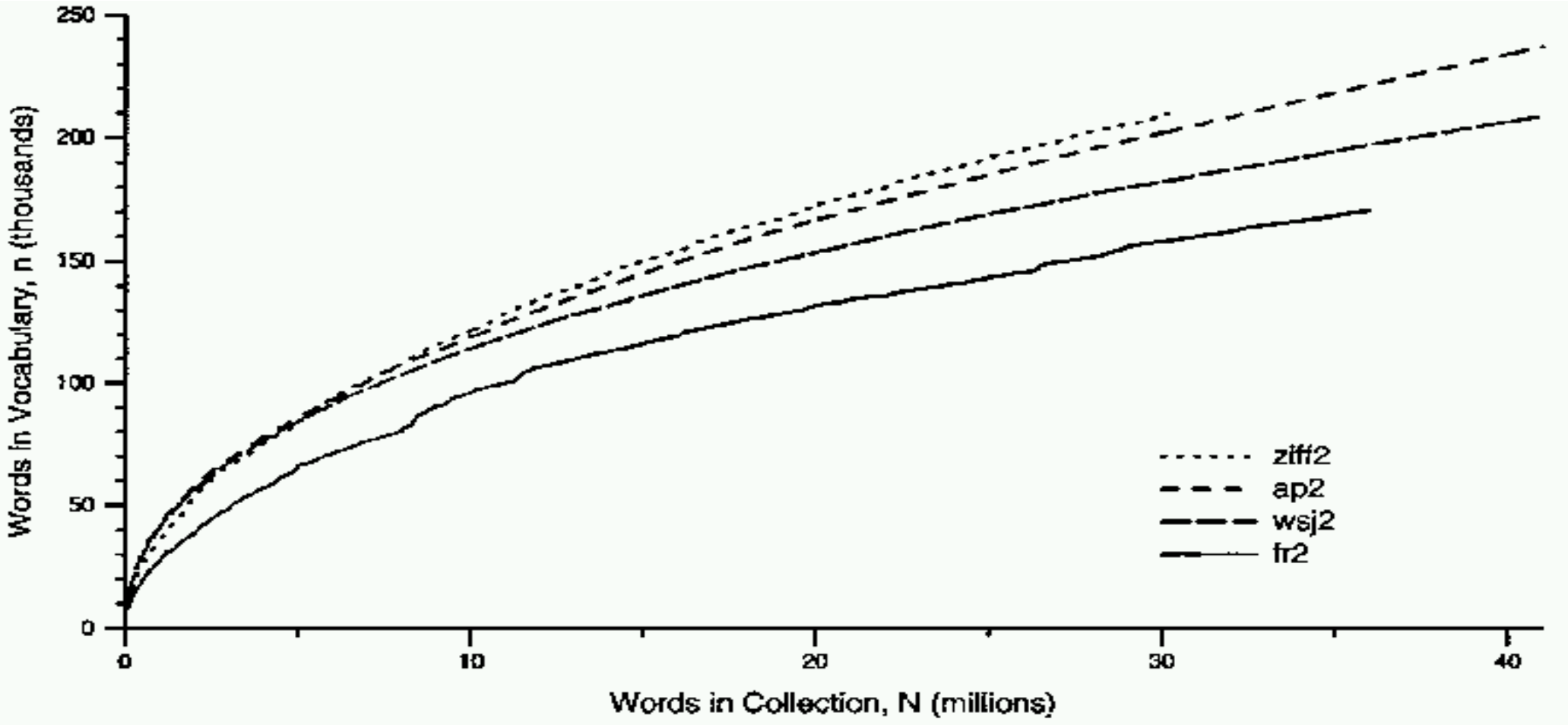
k = 44
b = 0.49

First 1,000,020 terms:
Predicted = 38,323
Actual = 38,365

Reuters-RCV1 collection: 806,791 newswire documents (Aug 20, 1996-August 19, 1997)

Manning, Raghavan, Schütze, Introduction to Information Retrieval (2008)

Heaps' Law Data



Explanation for Heaps' Law

- Can be derived from Zipf's law by assuming that documents are generated by randomly sampling words from a Zipfian distribution.

- Heap's Law holds on distribution of other data
  - Experiments on types of questions asked by users show a similar behavior

TEXT RETRIEVAL: ALGORITHMS

Step 1: Preprocessing

- Implement the preprocessing functions:
  - Tokenization
  - Stop word removal
  - Stemming
  - etc.

- Input: Documents that are read one by one from the collection

- Output: Tokens to be added to the index
  - No punctuation, no stop-words, stemmed

Step 2: Indexing

- Build an inverted index, with an entry for each word in the vocabulary
- Input: Tokens obtained from the preprocessing module
- Output: An inverted index for fast access

Step 3: Retrieval

- Use inverted index to find the limited subset of documents that contain at least one of the query words.
- Incrementally compute cosine similarity of each indexed document as query words are processed one by one.
- To accumulate a total score for each retrieved document, store retrieved documents in a hashtable, where the document id is the key, and the partial accumulated score is the value.

- Input: Query and Inverted Index
- Output: Similarity values between query and documents

Step 4: Ranking

- Sort the hashtable including the retrieved documents based on the value of cosine similarity
- Return the documents in descending order of relevance
- Input: Similarity values between query and documents
- Output: Ranked list of documented in reversed order of their relevance

STEP 1: PREPROCESSING

Tokenization

- Analyze text into a sequence of discrete tokens (words).
- Sometimes punctuation (“e-mail”, “a.out”), numbers (“1999”), and case (“Congress” vs. “congress”) can be a meaningful part of a token.
  - However, frequently they are not.
- Simplest approach: ignore all numbers and punctuation and use only case-insensitive unbroken strings of alphabetic characters as tokens.
- More careful approach:
  - Separate ? ! ; : “ ‘ [ ] ( ) < >
  - Care with .
  - Care with other punctuation marks

Tokenization (continued)

- Example: “We’re attending a tutorial now.” → we ’re attending a tutorial now

- Downloadable tool:
  - Word Splitter
    http://l2r.cs.uiuc.edu/~cogcomp/atool.php?tkey=WS

Numbers

- 3/12/91
- Mar. 12, 1991
- 55 B.C.
- B-52
- 100.2.86.144
  - Generally, don't index as text

Case Folding

- Reduce all letters to lower case
  - exception: upper case in mid-sentence
    - *e.g.,* ***General Motors***
    - ***Fed*** vs. ***fed***
    - ***SAIL*** vs***. sail***

Tokenizing HTML

- Should text in HTML commands not typically seen by the user be included as tokens?
  - Words appearing in URLs.
  - Words appearing in "meta text" of images.
- Simplest approach is to exclude all HTML tag information (between "<" and ">") from tokenization. But could lose critical information.

Stopwords

- Stop words are language & domain dependent
- For efficiency, store strings for stop words in a hash table to recognize them in constant time.

- How to determine a list of stopwords?
  - For English? – may use existing lists of stopwords
    - E.g. SMART's common word list
    - WordNet stopword list
    - http://www.ranks.nl/resources/stopwords.html
    - http://ir.dcs.gla.ac.uk/resources/linguistic_utils/stop_words
  - For Spanish? Bulgarian? Hindi?

Lemmatization

- Reduce inflectional/variant forms to base form
- Direct impact on VOCABULARY size
  - *am, are, is → be*
  - *car, cars, car's, cars' → car*

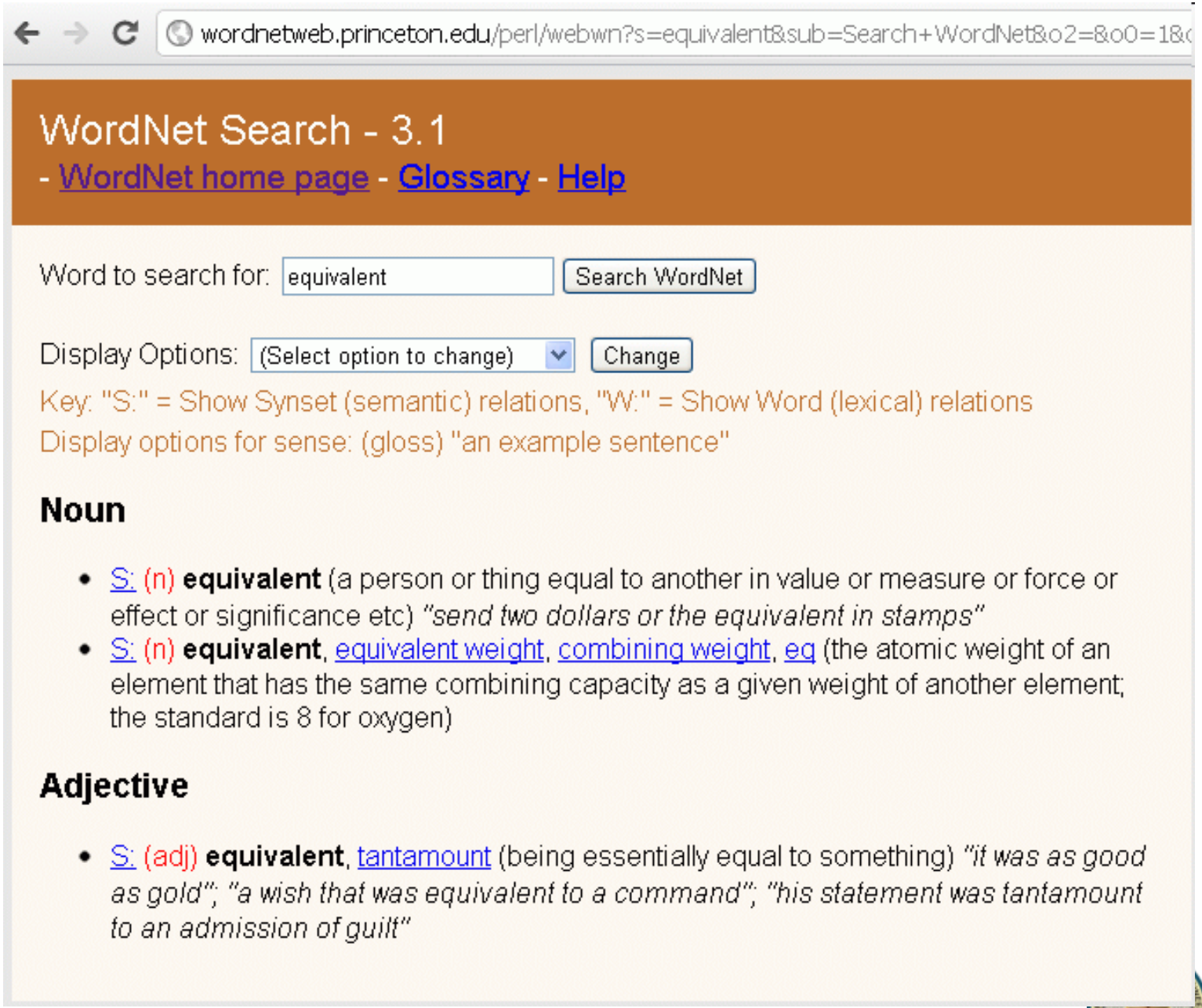
- *the boy's cars are different colors → the boy car be different color*

- How to do this?
  - Need a list of grammatical rules + a list of irregular words
  - Children → child, spoken → speak …
  - Practical implementation: use WordNet's morphstr function

WordNet

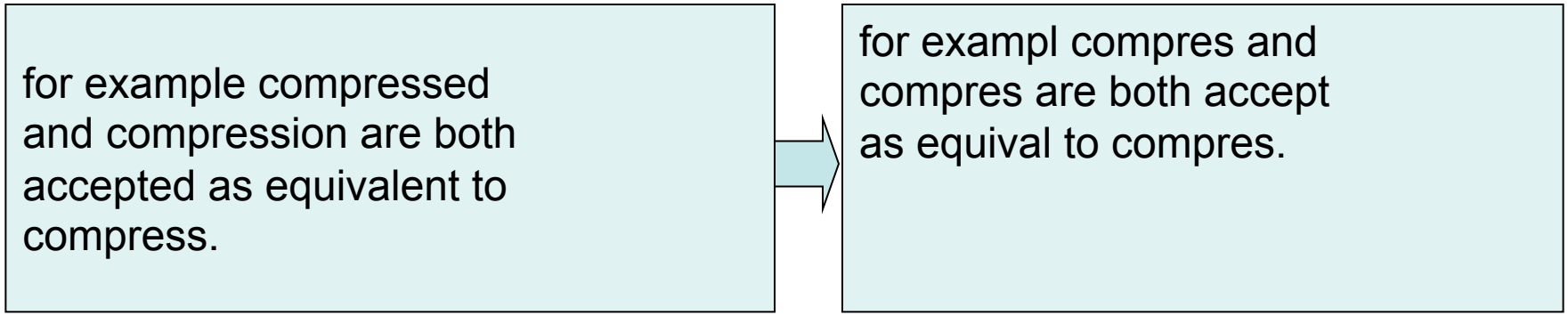
WordNet® is a large, free lexical database of English.

Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept.



Stemming

- Reduce tokens to "root" form of words to recognize morphological variation.
  - "computer", "computational", "computation" all reduced to same token "compute"
- Correct morphological analysis is language specific and can be complex.
- Stemming "blindly" strips off known affixes (prefixes and suffixes) in an iterative fashion.



Porter Stemmer – Sample Rules

http://tartarus.org/~martin/PorterStemmer/

- remove ending
  - if a word ends with a consonant other than s, followed by an s, then delete s.
  - if a word ends in es, drop the s.
  - if a word ends in ing, delete the ing unless the remaining word consists only of one letter or of th.
  - If a word ends with ed, preceded by a consonant, delete the ed unless this leaves only a single letter.
  - .....
- transform words
  - if a word ends with “ies” but not “eies” or “aies” then “ies --> y.”

Porter Stemmer

- Simple procedure for removing known affixes in English without using a dictionary.
- Can produce unusual stems that are not English words:
  - "computer", "computational", "computation" all reduced to same token "comput"
- May conflate (reduce to the same token) words that are actually distinct.
- Does not recognize all morphological derivations.

Other stemmers

- Other stemmers exist, e.g., Lovins stemmer
- Single-pass, longest suffix removal (about 250 rules)
- Full morphological analysis – only modest benefits for retrieval

Feature extraction

- Unigram features: to use each word as a feature
- Bigram features: to use two consecutive words as a feature
- Tool:
  - Write your own program/script
  - Lemur API: http://www.cs.cmu.edu/~lemur/3.0/api.html

STEP 2: INVERTED INDEX

Term-document incidence

	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

Goal: ***Brutus*** *AND* ***Caesar*** *BUT NOT* ***Calpurnia***

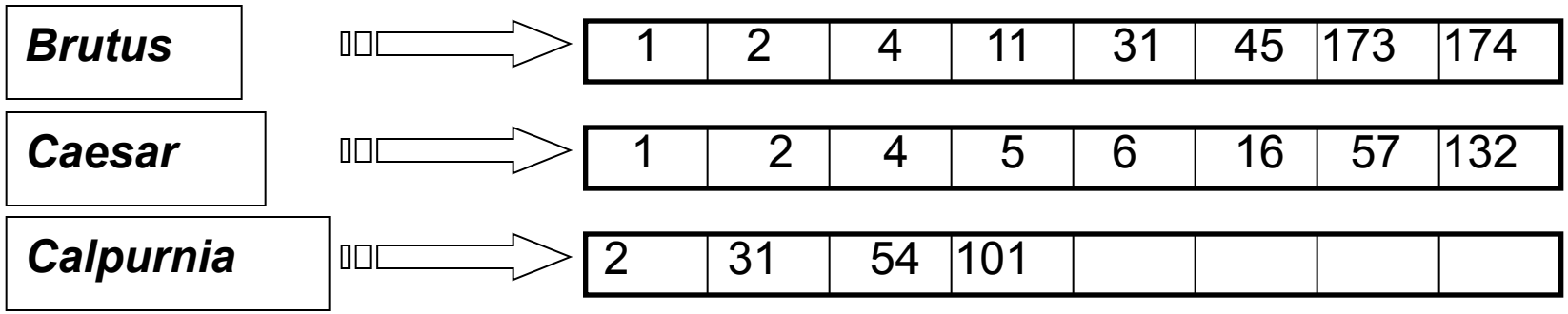
1 if play contains word, 0 otherwise

Inverted Index: Motivation

- *N* = 1 million documents, each with about 1000 words.
- *M* = 500K *distinct* terms among these.
- 500K x 1M matrix has half-a-trillion 0's and 1's.
- But it has no more than one billion 1's
  - matrix is extremely sparse.
- A better representation?
  - Only record the 1 positions.

Inverted index

- For each term *t*, store a list of all documents that contain *t*.
  - Identify each by a **docID**, a document serial number
  - Optionally store the number & position of occurrence of t in d.

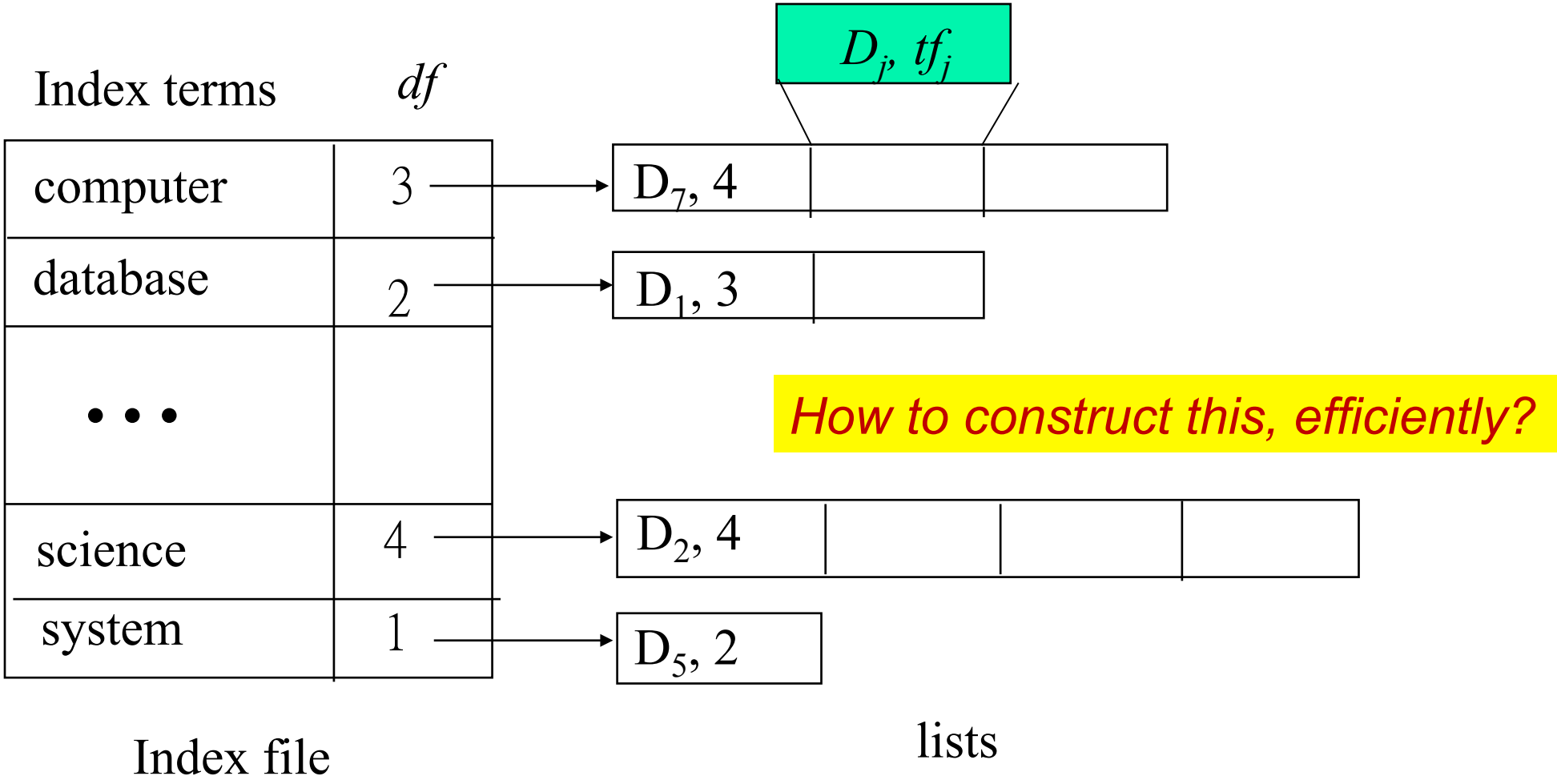


Cannot use fixed-size arrays for this.

Hence, post-facto insertion is an issue. This is best done as a batch job.

Inverted Index – in practice

- We need:
  - One entry for each word in the vocabulary
  - For each such entry:
    - Keep a list of all the documents where it appears together with the corresponding frequency → TF
  - For each such entry, keep the total number of occurrences in all documents:
    - IDF


Index terms
df
D_j, tf_j
computer
3
database
2
science
4
system
1
D_7, 4
D_1, 3
D_2, 4
D_5, 2
How to construct this, efficiently?
Index file
lists

Indexer steps: Token sequence

- Sequence of (Modified token, Document ID) pairs.

Doc 1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

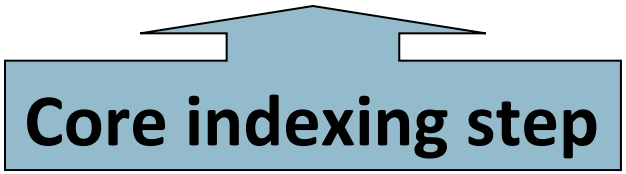
Doc 2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Indexer steps: Sort

- Sort by terms
  - And then docID



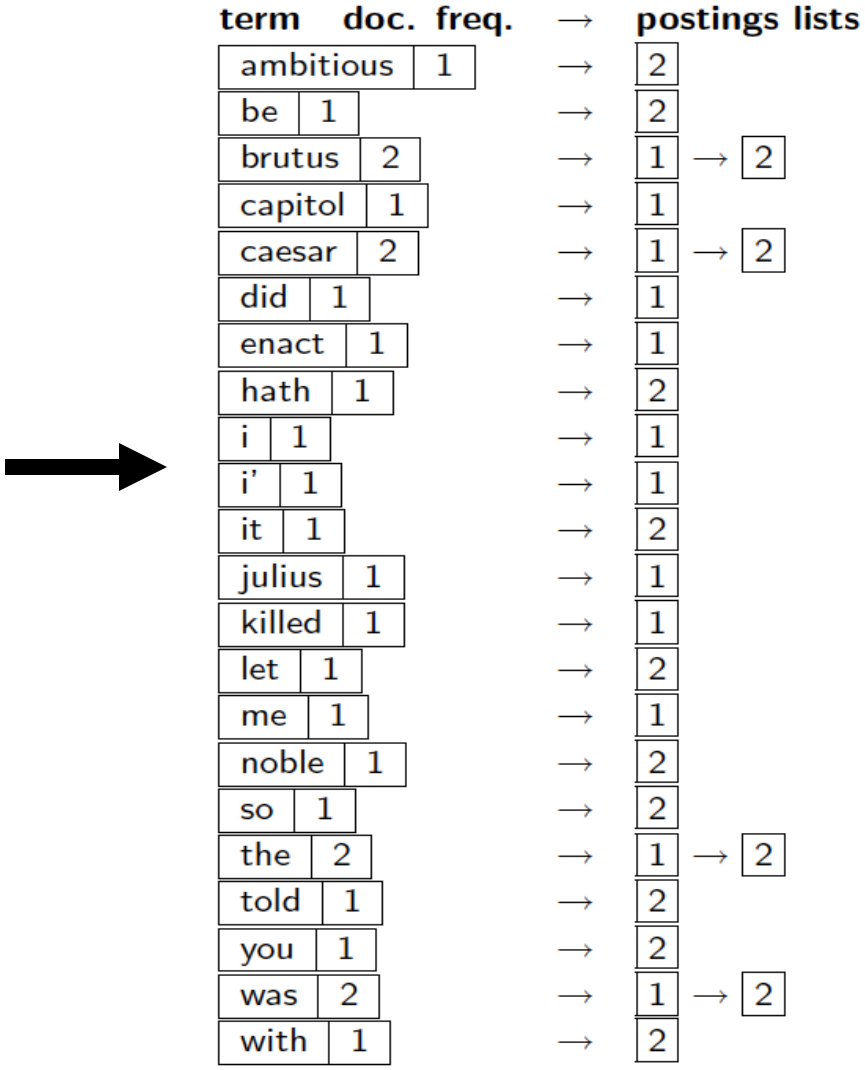
Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

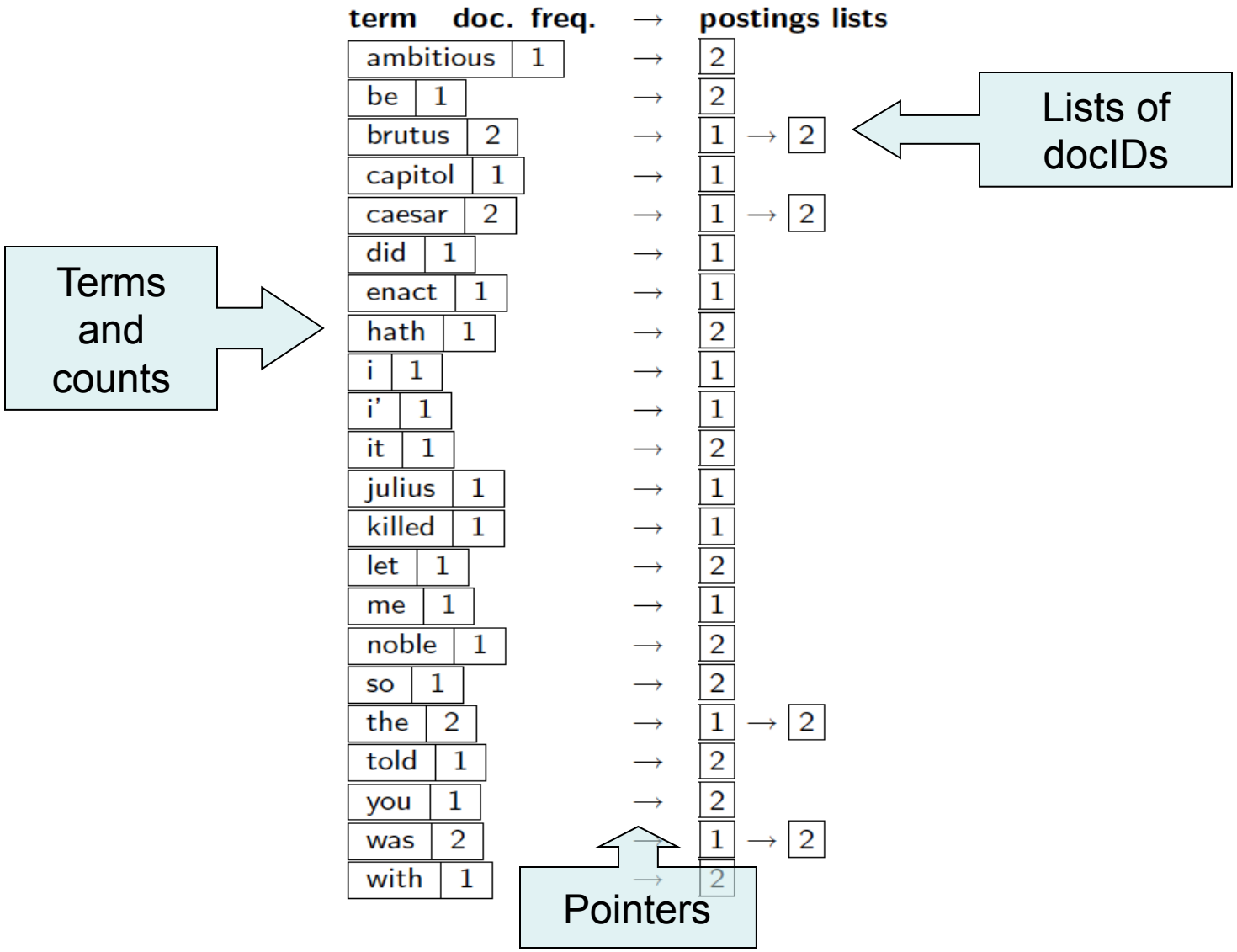
Indexer steps: Dictionary & Postings

- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Document frequency information is added.

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2



Where is the storage requirement?



Inverted Index: Another Example

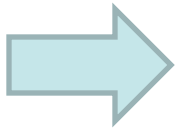
Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Doc 4
green eggs and ham

	1	2	3	4
blue		1		
cat			1	
egg				1
fish	1	1		
green				1
ham				1
hat			1	
one	1			
red		1		
two	1			



blue	→	2
cat	→	3
egg	→	4
fish	→	1 → 2
green	→	4
ham	→	4
hat	→	3
one	→	1
red	→	2
two	→	1

Inverted Index: Ranked Retrieval

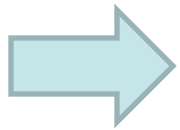
Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Doc 4
green eggs and ham

	tf 1	2	3	4	*df*
blue		1			1
cat			1		1
egg				1	1
fish	2	2			2
green				1	1
ham				1	1
hat			1		1
one	1				1
red		1			1
two	1				1



blue	1	2,1
cat	1	3,1
egg	1	4,1
fish	2	1,2 → 2,2
green	1	4,1
ham	1	4,1
hat	1	3,1
one	1	1,1
red	1	2,1
two	1	1,1

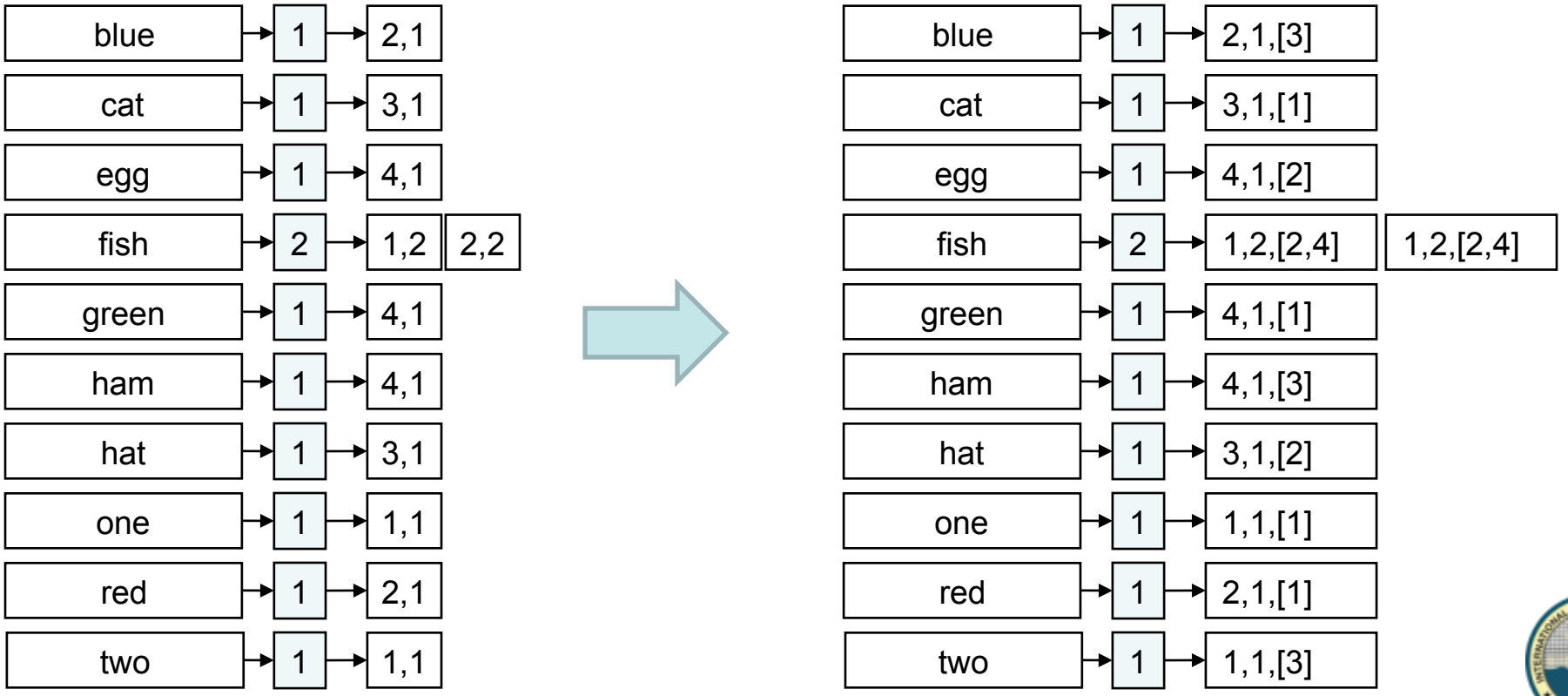
Inverted Index: Positional Information

Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Doc 4
green eggs and ham



Indexing: Performance Analysis

- Fundamentally, a large sorting problem
  - Terms usually fit in memory
  - Postings usually don't
- How is it done on a single machine?
- How can it be done with MapReduce?
- First, let's characterize the problem size:
  - Size of vocabulary
  - Size of postings

Inverted Indexing with MapReduce

Map

Doc 1
one fish, two fish

one	1	1
two	1	1
fish	1	2

Doc 2
red fish, blue fish

red	2	1
blue	2	1
fish	2	2

Doc 3
cat in the hat

cat	3	1
hat	3	1

Shuffle and Sort: aggregate values by keys

Reduce

cat	3	1		
fish	1	2	2	2
one	1	1		
red	2	1		

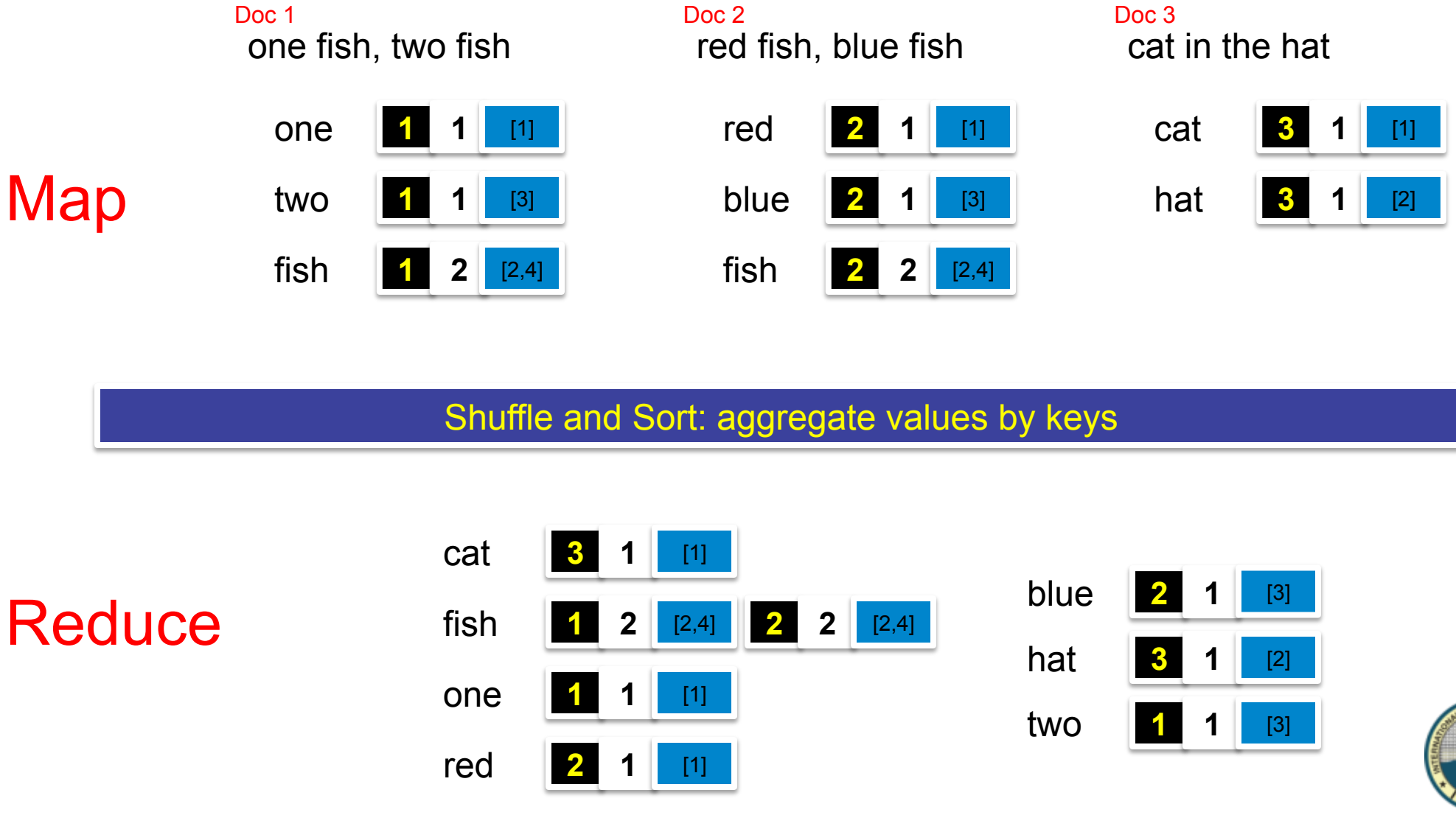
blue	2	1
hat	3	1
two	1	1

Inverted Indexing: Pseudo-Code

```
1: procedure Map(a, d)
2:     InitializeAssociativeArray(H)
3:     for all t ∈ d do
4:         H{t} ← H{t} + 1
5:     for all t ∈ H do
6:         Emit(t, ⟨a, H{t}⟩)

1: procedure Reduce(t, [⟨a1, f1⟩, ⟨a2, f2⟩ ...])
2:     InitializeList(P)
3:     for all ⟨a, f⟩ ∈ [⟨a1, f1⟩, ⟨a2, f2⟩ ...] do
4:         Append(P, ⟨a, f⟩)
5:     Sort(P)
6:     Emit(t, P)
```

Positional Indexes



Query processing: AND

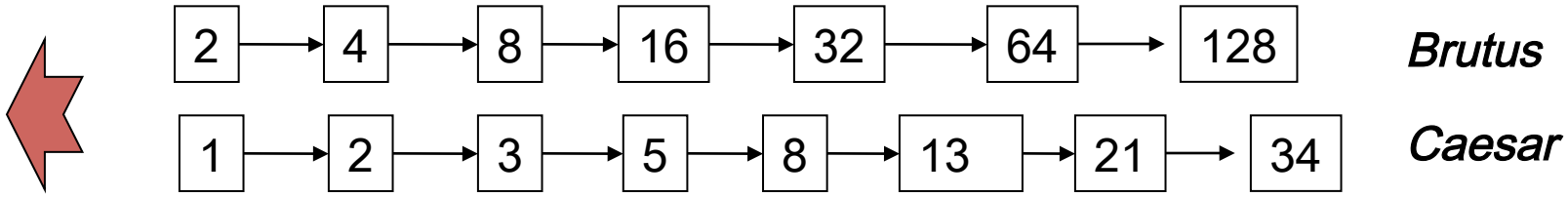
- Consider processing the query:

  Brutus *AND* ***Caesar***

  – Locate ***Brutus*** in the Dictionary;
    - Retrieve its postings.

  – Locate *Caesar* in the Dictionary;
    - Retrieve its postings.

  – "Merge" the two postings:



STEP 3 & 4: RETRIEVAL & RANKING

Vector-Space Model

- Regard query as short document
- Return the documents ranked by the closeness of their vectors to the query, also represented as a vector.
- Vectorial model was developed in the SMART system (Salton, c. 1970) and standardly used by TREC participants and web TRM systems today.

Vector-Space Model (contd.)

- t distinct terms remain after preprocessing
  - Unique terms that form the VOCABULARY

- These "orthogonal" terms form a vector space.

  Dimension = t = |vocabulary|
  - 2 terms → bi-dimensional; …; n-terms → n-dimensional

- 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:

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

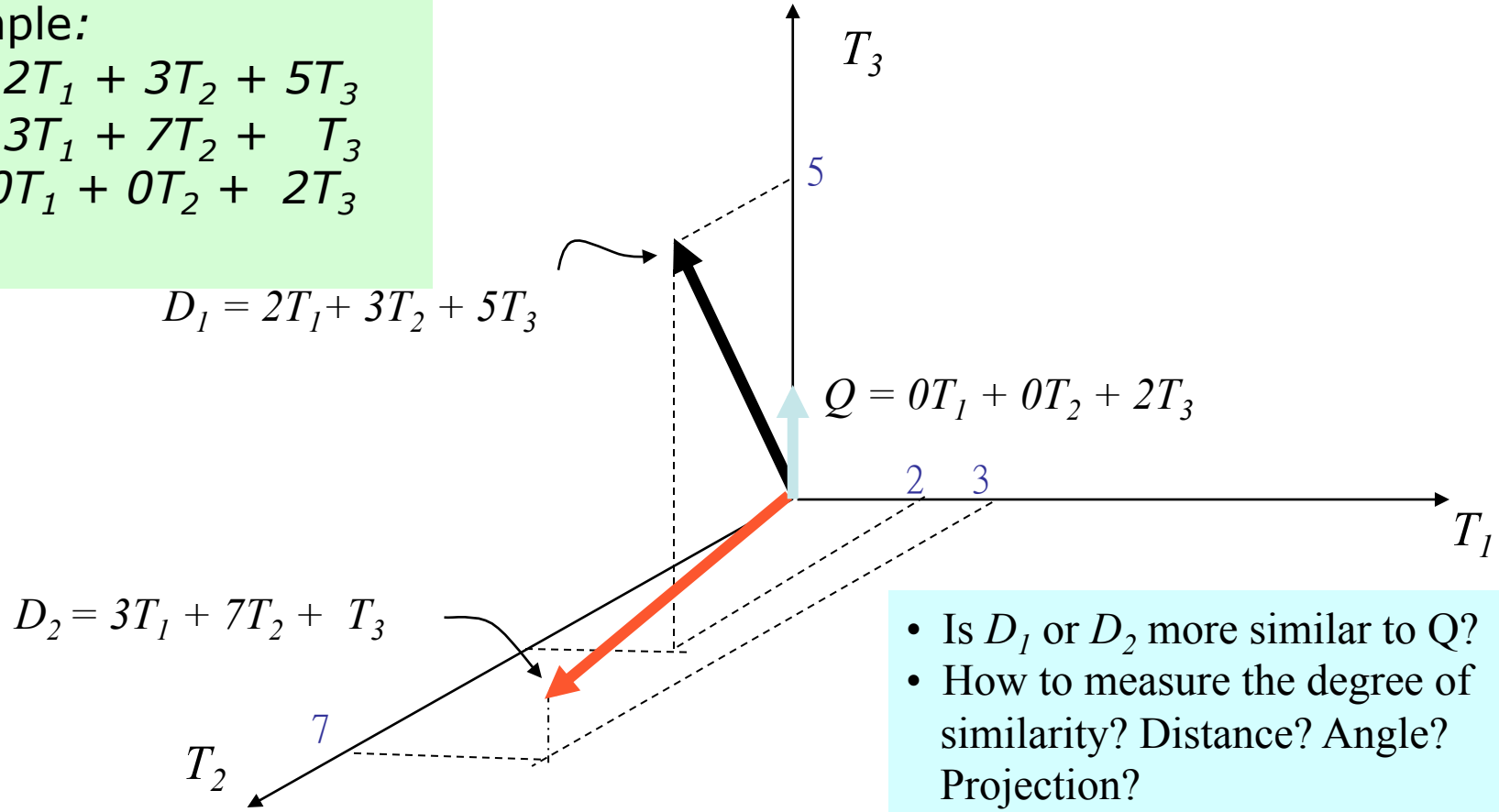
Graphic Representation

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?

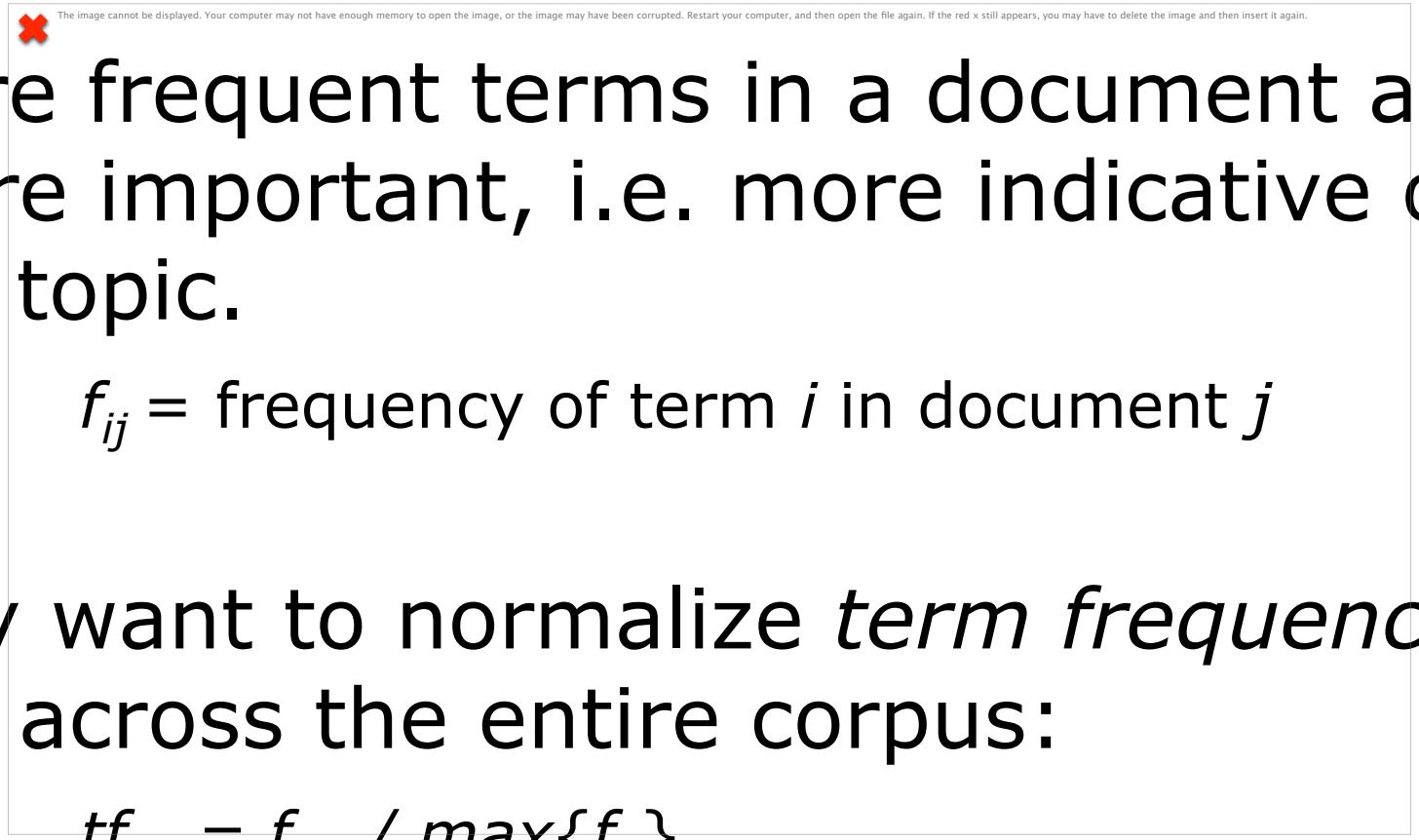
Document Collection Representation

- 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.

	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 (TF)



- 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 (IDF)

- 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 weighting 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.

- Experimentally, *tf-idf* has been found to work well. It was also theoretically proved to work well (Papineni, NAACL 2001)

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(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

Summary: Text & Representation

- d1 = {machine, learning, support vector, machine, machine, data, tree}
- d2 = {data, mining, associat, classifier, classifier, data, data, associat}
- d3 = {mining, decision, tree, decision}
- d4 = {associat, mining, data, mining,}
- d5 = {decision, tree, classifier}

Boolean Representation

	associat	classifier	data	decision	learning	machine	mining	support	tree	vector
d1	0	0	1	0	1	1	0	1	1	1
d2	1	1	1	0	0	0	1	0	0	0
d3	0	0	0	1	0	0	1	0	1	0
d4	1	0	1	0	0	0	1	0	0	0
d5	0	1	0	1	0	0	0	0	1	0

Summary (contd.)

Vector-space Representation

	associat	classifier	data	decision	learning	machine	mining	support	tree	vector
d1	0	0	1	0	1	3	0	1	1	1
d2	2	2	3	0	0	0	1	0	0	0
d3	0	0	0	1	0	0	1	0	1	0
d4	1	0	1	0	0	0	2	0	0	0
d5	0	1	0	1	0	0	0	0	1	0

TF-IDF Representation (normalized)

	associat	classifier	data	decisio n	learni ng	machine	mining	suppor t	tree	vector
d1	0	0	0.17	0	0.54	1.61	0	0.54	0.17	0.54
d2	0.61	0.61	0.51	0	0	0	0.17	0	0	0
d3	0	0	0	0.92	0	0	0.51	0	0.51	0
d4	0.46	0	0.46	0	0	0	0.51	0	0	0
d5	0	0.92	0	0.92	0	0	0	0	0.51	0

http://cran.r-project.org/web/views/NaturalLanguageProcessing.html

CRAN Task View: Natural Language Processing

Maintainer: Fridolin Wild, Knowledge Media Institute (KMi), The Open University, UK
Contact: fridolin.wild at open.ac.uk
Version: 2014-01-01

http://cran.r-project.org/web/packages/tm/vignettes/tm.pdf

Introduction to the **tm** Package
Text Mining in R

Ingo Feinerer

January 13, 2014

Questions / Suggestions?

International School of Engineering

Plot No 63/A, 1st Floor, Road No 13, Film Nagar, Jubilee Hills, Hyderabad - 500033

For Individuals: +91-9502334561/63

For Corporates: +91-9618483483

Web: http://www.insofe.edu.in

Facebook: https://www.facebook.com/Insofe

Twitter: https://twitter.com/INSOFEedu

YouTube: http://www.youtube.com/InsofeVideos

SlideShare: http://www.slideshare.net/INSOFE

This presentation may contain references to findings of various reports available in the public domain. INSOFE makes no representation as to their accuracy or that the organization subscribes to those findings.