

Text Mining – an introduction

Michalis Vazirgiannis

LIX @ Ecole Polytechnique

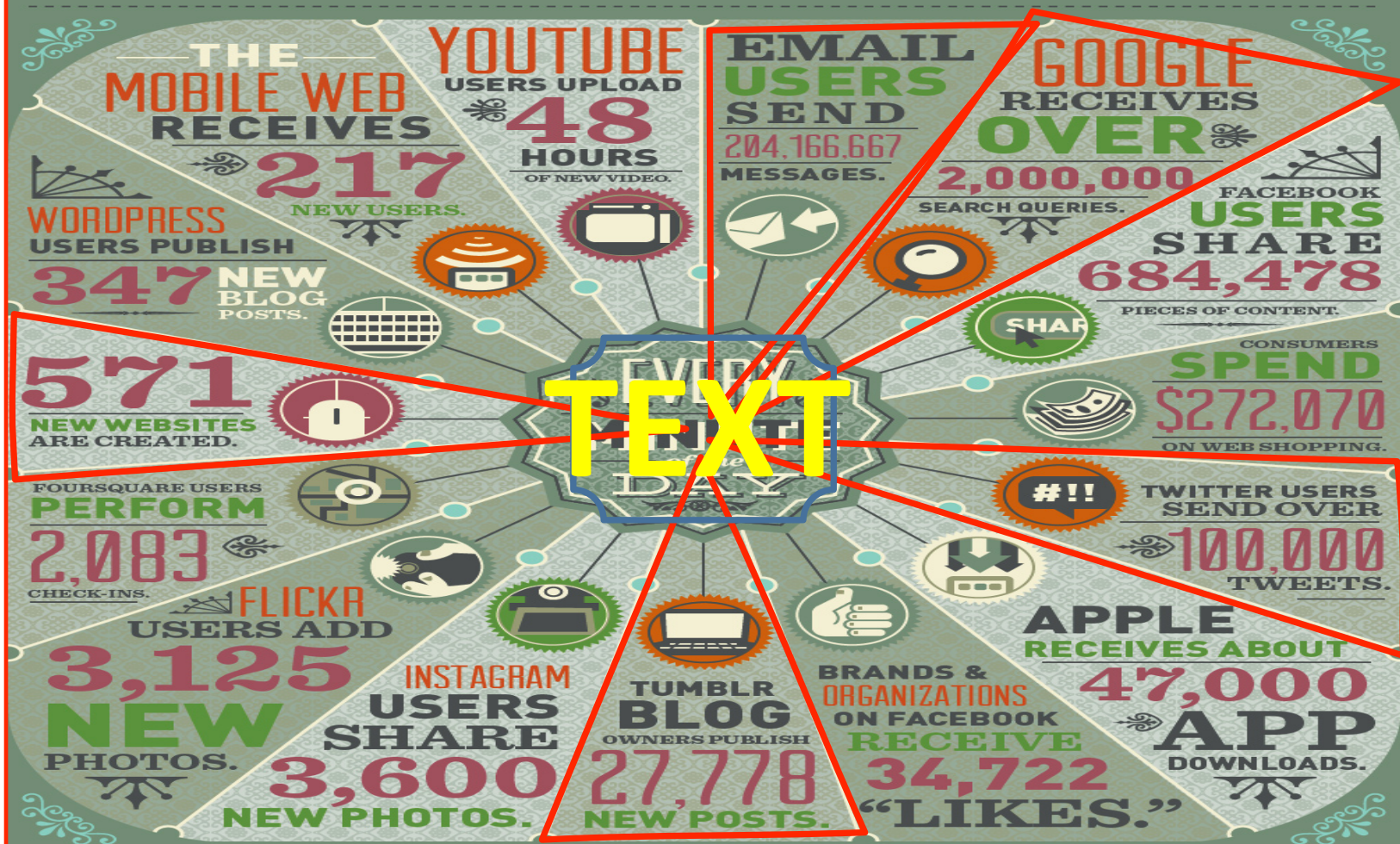
June 2015

DOMO

DATA NEVER SLEEPS

How Much Data Is Generated Every Minute?

Big data is not just some abstract concept used to inspire and mystify the IT crowd; it is the result of an avalanche of digital activity pulsating through cables and airwaves across the world. This data is being created every minute of the day through the most innocuous of online activity that many of us barely even notice. But with every website browsed, status shared, or photo uploaded, we leave digital trails that continually grow the hulking mass of big data. Below, we explore how much data is generated in one minute on the Internet.



WITH NO SIGNS OF SLOWING, THE DATA KEEPS GROWING

These are just some of the more common ways that Internet users add to the big data pool. In truth, depending on the niche of business you're in, there are virtually countless other sources of relevant data to pay attention to. Consider the following:

The global Internet population grew 6.59 percent from 2010 to 2011 and now represents

2.1 BILLION PEOPLE.

These users are real, and they are out there leaving data trails everywhere they go. The team at Domo can help you make sense of this seemingly insurmountable heap of data, with solutions that help executives and managers bring all of their critical information together in one intuitive interface, and then use that insight to transform the way they run their business. To learn more, visit www.domo.com.

DOMO

SOURCES: [HTTP://NEWS.INVESTORS.COM/](http://news.investors.com/), [ROYAL.PINGDOM.COM](http://royal.pingdom.com), [BLOG.GROOV.COM](http://blog.groov.com), [BLOG.HUBSPOT.COM](http://blog.hubspot.com), [SIMPLYZESTY.COM](http://simplyzesty.com), [PEWRESEARCH.COM](http://pewresearch.com), [WWW.WASHINGTONPOST.COM](http://www.washingtonpost.com)

<http://visually/data-never-sleeps>

Outline

- Document collection preprocessing
- Feature Selection
- Indexing
- Query processing & Ranking
- Retrieval evaluation
- Latent Semantic Indexing

Text representation for Information Retrieval

- We seek a transformation of the textual content of documents into a vector space representation.
 - Assume documents as the data
 - Dimensions are the distinct terms used
- Example : “This is the database lab of the IS master course”

This	is	the	database	lab	of	IS	master	course

Boolean Vector Model

- Boolean model
 - Text 1: “This is the database lab of the IS master course”
 - Text 2: “This is a database course”

Text 1:

Text 2:

This	is	a	the	database	lab	of	IS	master	course
1	1	0	1	1	1	1	1	1	1
1	1	1	0	1	0	0	0	0	1

Vector Space Model

- Vector Space Model:
 - To VSM represents the significance of each term for each document
 - The cell values are computed based on the terms' frequency
 - Most common approach is TF/IDF

Feature Selection

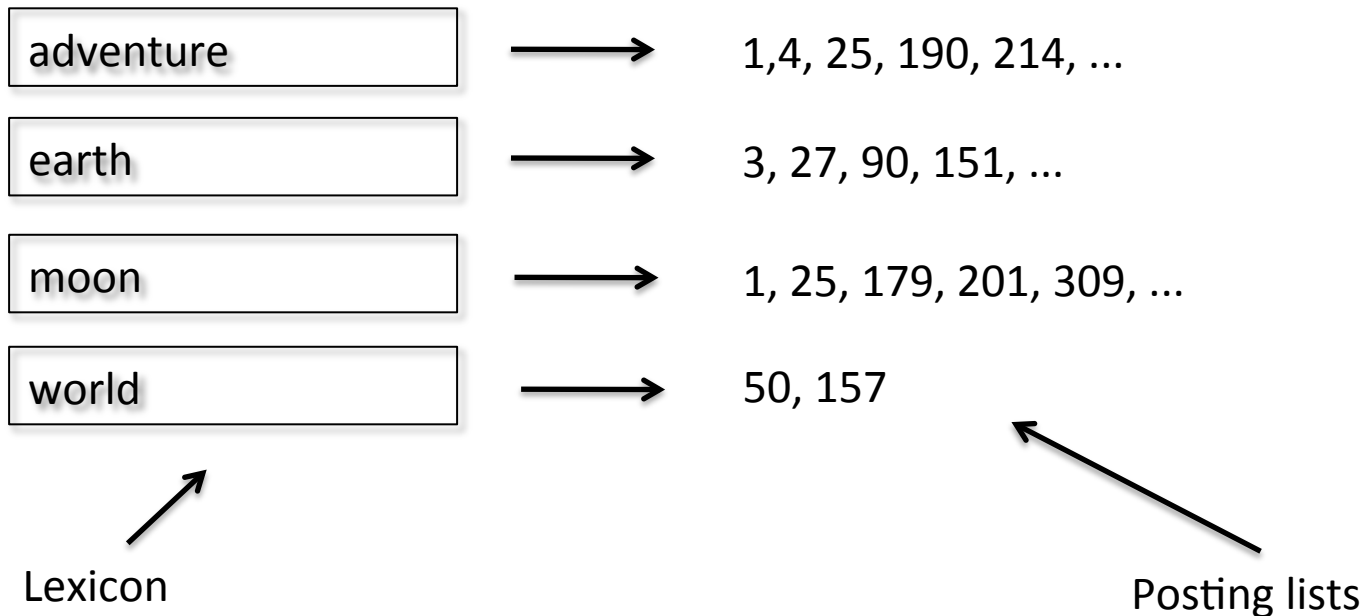
Text 1:

Text 2:

This	is	a	the	database	lab	of	IS	master	course
0.1	0.1	0	0.2	0.1	0.1	0.1	0.1	0.1	0.1
0.2	0.2	0.2	0	0.2	0	0	0	0	0.2

Inverted Index

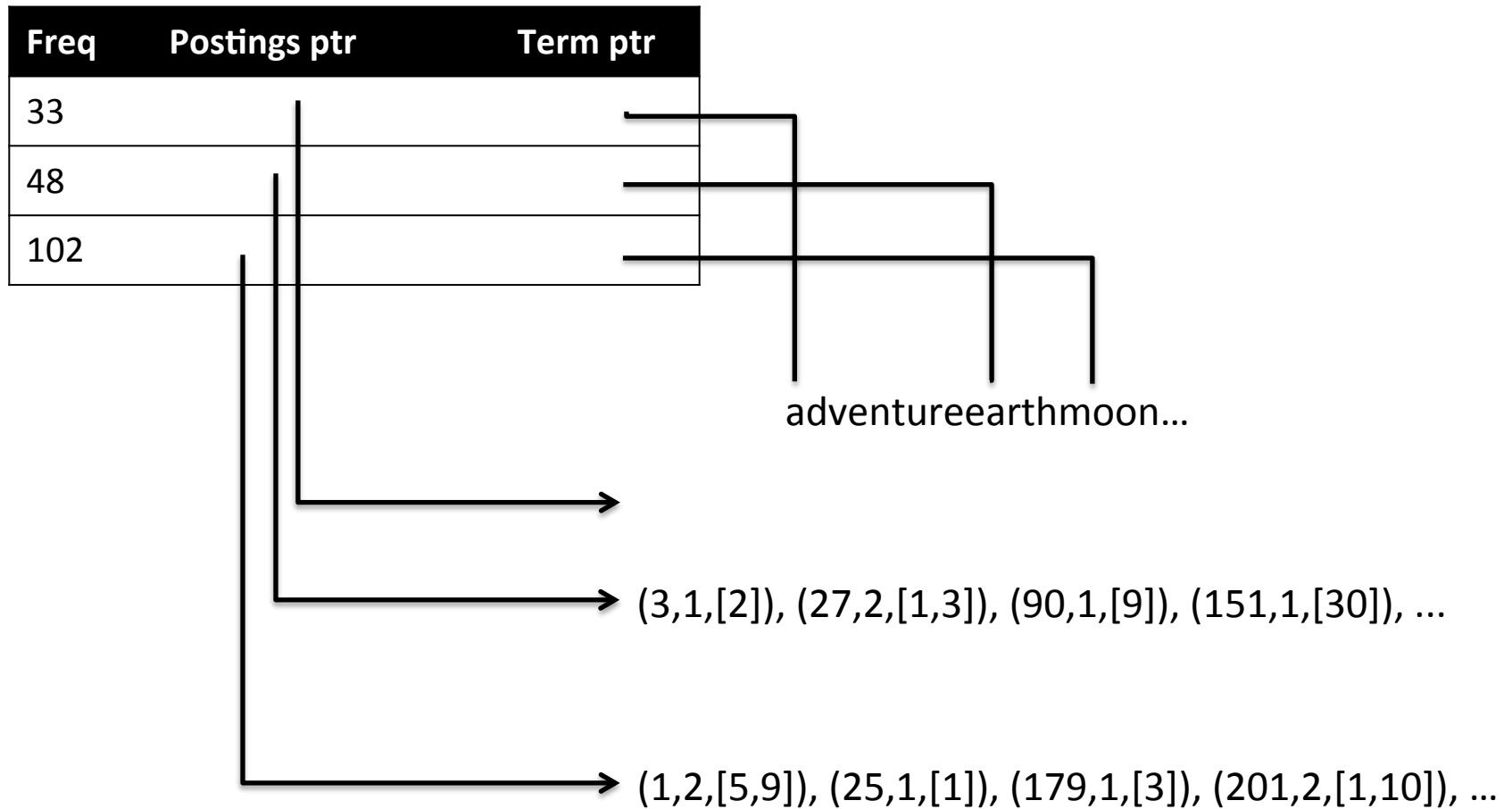
- Record the documents in which each term occurs in
 - Similar to the *Index* at the end of books



Lexicon structures

- Fixed-size entry array, allocating
 - n bytes for storing the term string
 - 8 bytes for pointing to the posting list
 - 32 bytes for storing statistics and numerical term ids
- Needs total $N \times (n+40)$ bytes per entry
 - If $n=20$, $N=50\text{M}$ we need 2.79Gb for the lexicon
- Allocating more space than needed
 - Average length of English words is approx. 4.5

Dictionary as a String



Dictionary as a String with Blocks

Store pointers every k terms

Fewer pointers: $4(k-1)$ bytes/block

Store length in 1 additional byte/term

Freq	Postings ptr	Term ptr
33		
48		
102		

9adventure5earth4moon...

→ (1,2,[3,5]), (4,1,[5]), (25,2,[3,20]), (190,1,[2]), ...

→ (3,1,[2]), (27,2,[1,3]), (90,1,[9]), (151,1,[30]), ...

→ (1,2,[5,9]), (25,1,[1]), (179,1,[3]), (201,2,[1,10]), ...

Front-coding

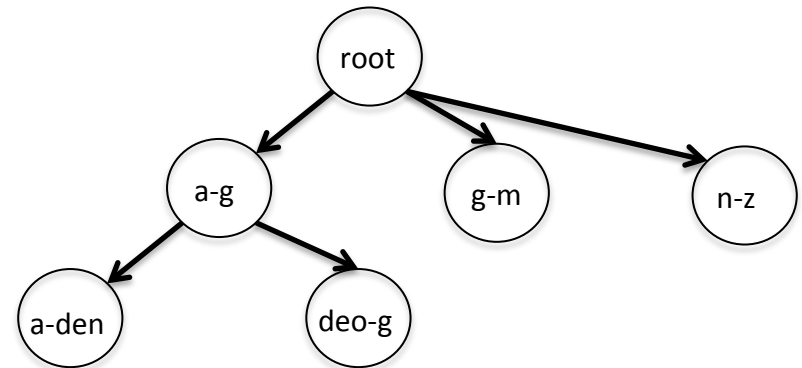
- Further compress strings within the same block
- Sorted terms share common prefix
 - store only the differences

8automata8automate9automatic10automation

→8automat*a1:e2:ic3:ion

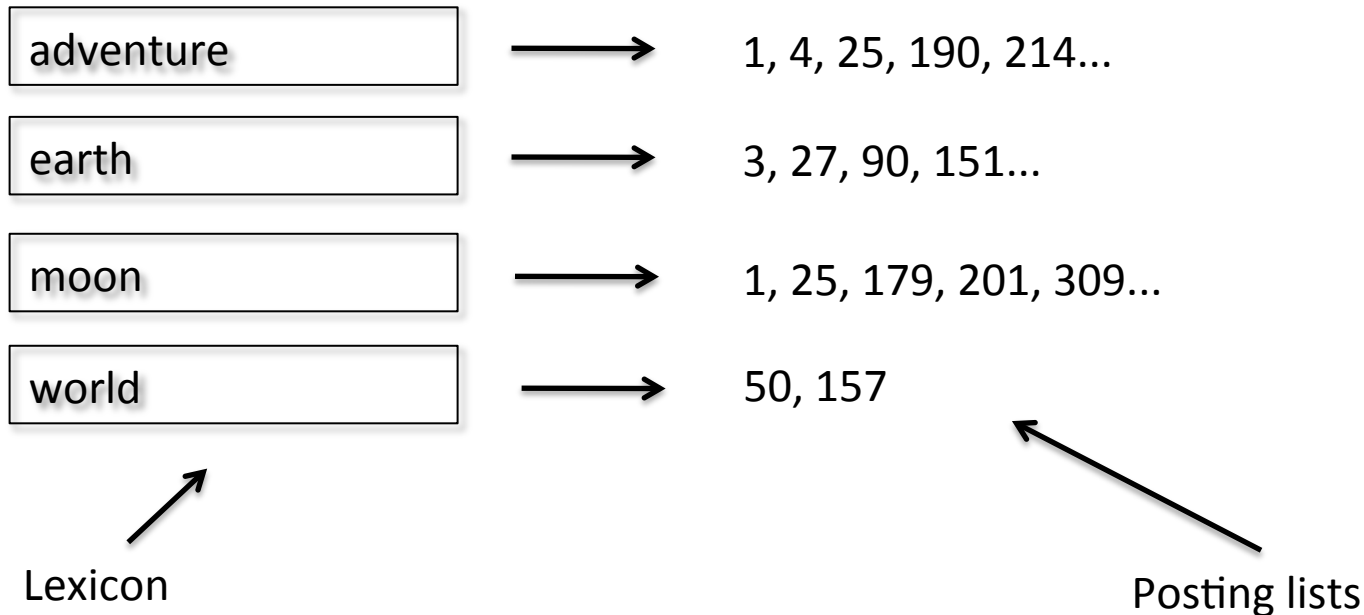
Searching Lexicons

- Hash tables
 - Each vocabulary term is hashed to an integer
 - Allow very fast lookup in constant time $O(1)$
 - Do not support finding variants of terms
 - colour / color
 - Require expensive adjustment to handle changing/growing vocabularies
- Trees: Binary trees, B-Trees
 - Trees allow prefix search
 - Slower search and rebalancing to handle growing/changing vocabularies



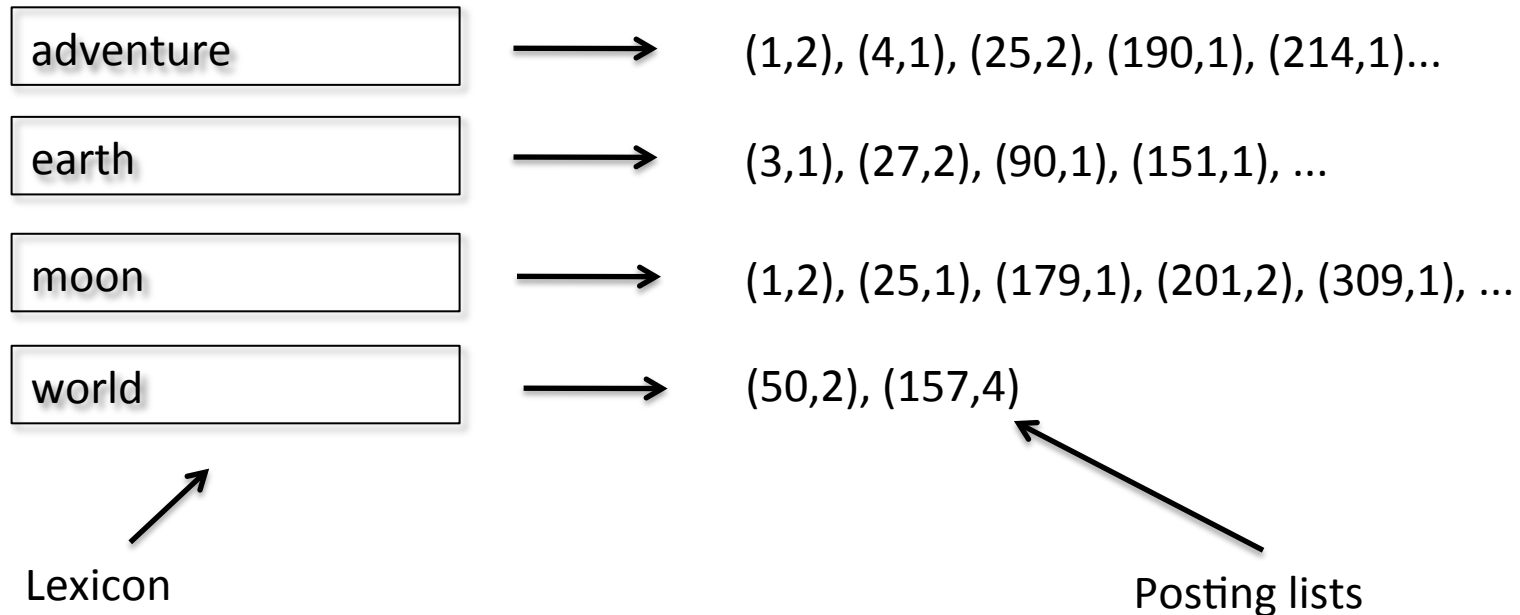
Payload in posting lists

- Simplest case to store only document identifiers (docIDs)



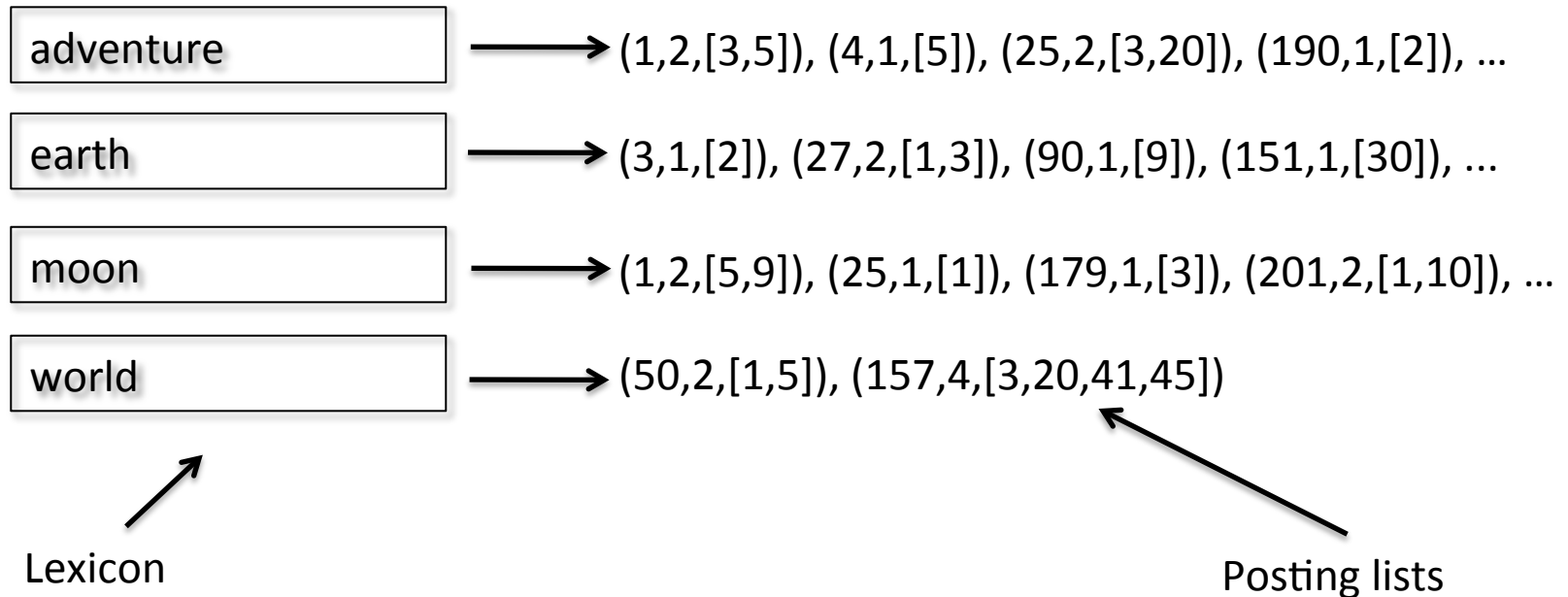
Payload in posting lists

- Store the frequency of a term in a document



Payload in posting lists

- Store the frequency of a term in a document and its positions



Payload in posting lists

- Store linguistic information in posting lists

A recent event at the National Library in Athens, drew a crowd of 300.

Payload in posting lists

- Store linguistic information in posting lists
 - Part-of-Speech information per term occurrence

A recent event at the National Library in Athens, drew a crowd of 300.
DT JJ NN IN DT NNP NNP IN NNP VDB DT NN IN CD

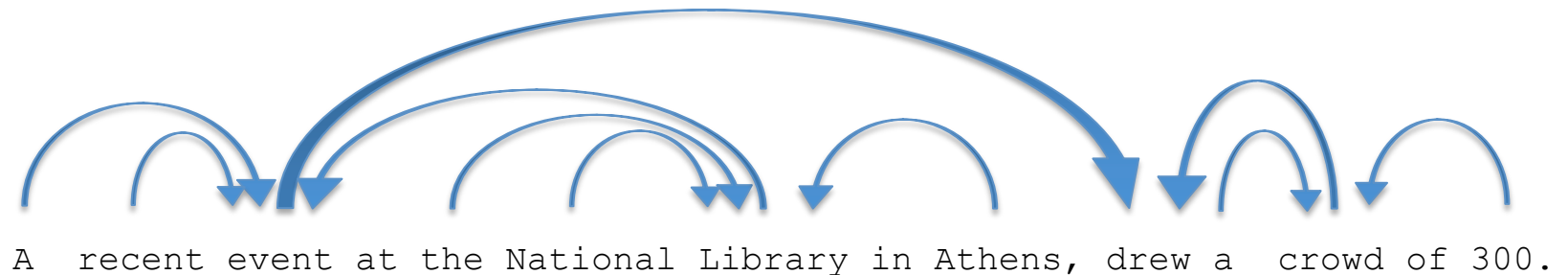
Payload in posting lists

- Store linguistic information in posting lists
 - Part-of-Speech information per term occurrence
 - References to named entities

A recent event at the National Library in Athens, drew a crowd of 300.
O O O O O **ORG** **ORG** O **LOC** O O O O O

Payload in posting lists

- Store linguistic information in posting lists
 - Part-of-Speech information per term occurrence
 - References to named entities
 - Dependency parse trees



Inverted Index construction

- Steps:
 - Document processing:
 - Parsing, Tokenization, Linguistic processing
 - Inversion of posting lists

Document parsing

- Handle different document formats
 - Html, PDF, MS Word, Flash, PowerPoint, ...
- Detect encoding of characters
 - How to translate bytes to characters?
 - Popular choices for Web pages:
 - UTF-8, ISO8859-7, Windows 1253
- Detect language of text
 - Estimate the probability of sequences of characters from a sample of documents
 - Assign the most likely language to an unseen document

Tokenization

- Split text in sequences of tokens which are candidates to be indexed
- But, pay attention to
 - Abbreviations: **U.N.** and **UN** (United Nations or 1 in French)
 - **New York** as one or two tokens
 - **c++** as one token, but not c+
 - Apostrophes
 - Hyphenation: one-man-show, Hewlett-Packard
 - Dates: 2011/05/16, May 16th, 2011
 - Numbers: (+33) 8203-911
 - Accents: Ελλάδα / Ελλαδα, Université

Stop-words

- Very frequent words that do not carry semantic information
 - the, a, an, and, or, to
- Stop-words can be removed to reduce index size requirements and to speed-up query processing
- But
 - Improvements in compression and query processing can offset the impact from stop-words
 - Stop-words are useful for certain queries submitted to Web search engines
 - “The The”, “Let it be”, “To be or not to be”

Lemmatization & Stemming

- Lemmatization
 - Reduce inflected forms of a word so that they are treated as a single term: am, were, being, been → be
 - Requires knowledge of context, grammar, part of speech
- Stemming: reduces tokens to a “root” form
 - Porter’s Stemming Algorithm
 - Applicable to texts written in English
 - Removes the longest-matching suffix from words
 - EED → EE agreed → agree, **but** feed → feed
 - ED → plastered → plaster, **but** bled → bled
 - ING → motoring → motor, **but** sing → sing
 - Limitation: resulting terms are not always readable

Sort-Based Indexing

- Steps
 - Collect all pairs term-docID from documents
 - Sort pairs on term and then docID
 - Organize the docIDs for each term in posting lists
- Optimization
 - Map each term to termID and work with pairs termID-docID
- Limitation
 - Not enough memory to hold termID-docID pairs
 - External sort algorithm

Single-Pass In-Memory Index Construction

- Main idea
 - Build intermediate complete inverted indexes
 - At the end, merge the intermediate indexes
 - No need to keep information between intermediate indexes

While more docs to process

 Initialize dictionary

 While free memory available

 Get next token

 If term(token) exists in dictionary

 Then get posting_list

 Else add new posting_list to dictionary

 Add term(token), docID to posting_list

 Sort dictionary terms

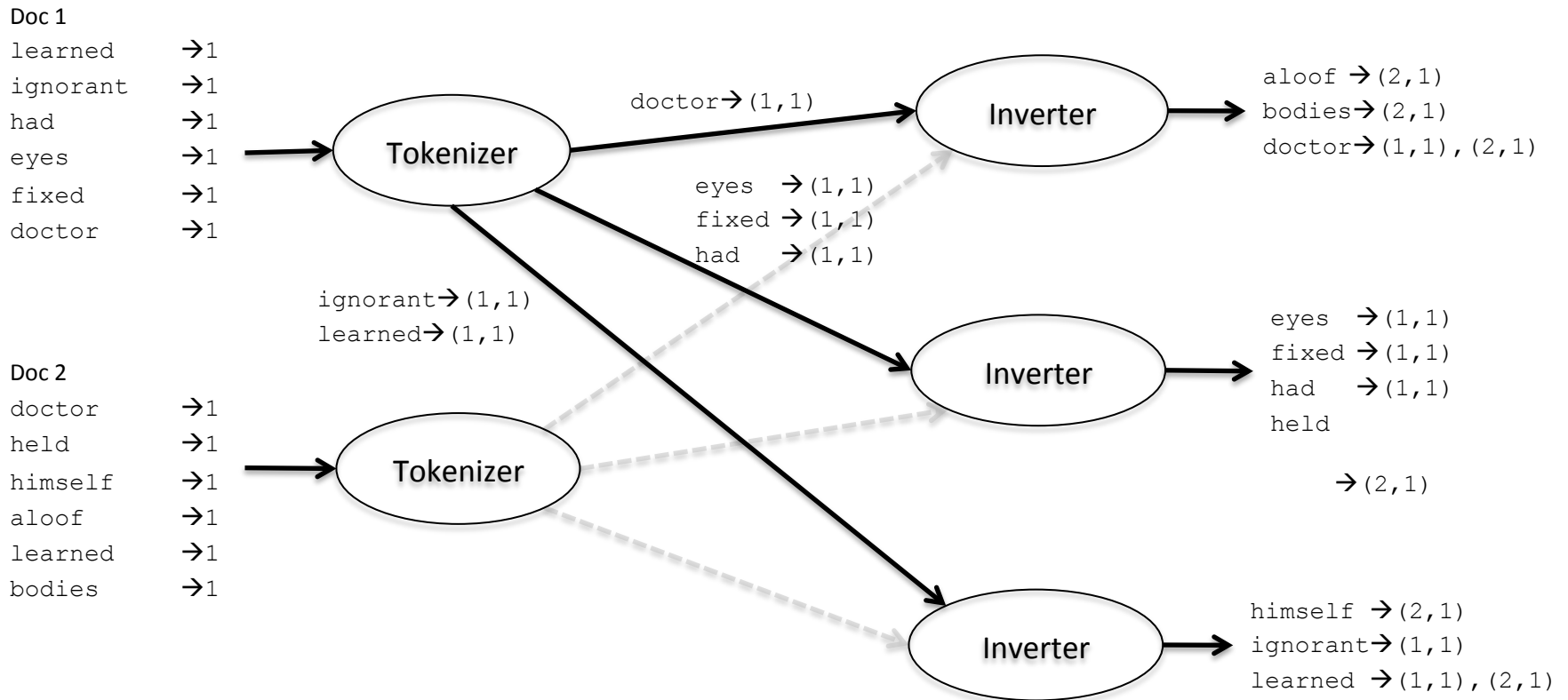
 Write block of postings to disk

Merge blocks of postings

Distributed Indexing

- Single-Pass In-Memory indexing can be applied for any number of documents
 - **But** it will take too long to index 100 billion Web pages
- Indexing can be parallelized
 - Clusters of commodity servers used to index billions of documents

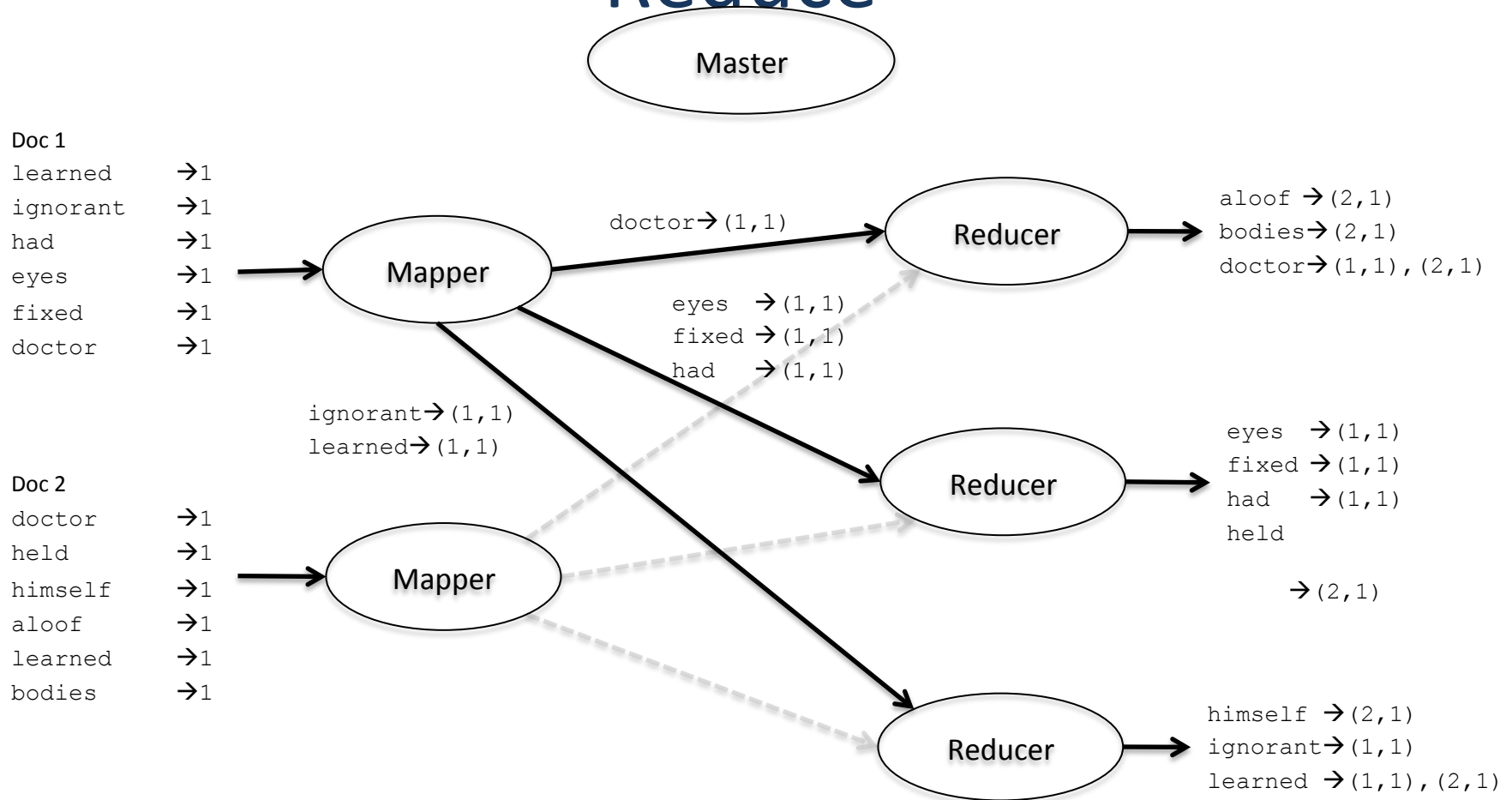
Distributed Indexing



Map-Reduce framework

- Framework for handling distribution transparently
 - provides distribution, replication, fault-tolerance
- Computation is modeled as a sequence of **Map** and **Reduce** steps
 - **Map** emit (key, value) pairs
 - **Reduce** collects (key, value) pairs for a range of keys

Distributed Indexing with Map/Reduce



Mapper emits pairs: term -> (docid, frequency)

Reducer emits pairs: term -> posting list

Queries & Document ranking

Query-document matching scores

- How do we compute the score of a query-document pair?
- one-term query: “*Sentiment*”
- If the term “*Sentiment*” does not occur in the document:
score=0.
- The more frequent the query term in the document, the higher the score
- We will look at a number of alternatives for doing this.

Jaccard coefficient

- A commonly used measure of overlap of two sets

- Let A and B be two sets

- Jaccard coefficient: $Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|}$

$$(A \neq \emptyset \text{ or } B \neq \emptyset)$$

- $Jaccard(A, A) = 1$ - $Jaccard(A, B) = 0$ if $A \cap B = \emptyset$
- A and B don't have to be the same size.

Example

What is the query-document match score that the Jaccard coefficient computes for:

- Query: "ides of March"
- Document "Caesar died in March"
- $JACCARD(q, d) = 1/6$

Issues with Jaccard?

- It doesn't consider term frequency
- Rare terms are more informative than frequent terms. Jaccard does capture this.
- We need a more sophisticated way of normalizing for the length of a document.

Binary incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth ...
ANTHONY	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
...						

Each document is represented as a binary vector $\in \{0, 1\}^{|V|}$.

Problems with Jaccard:

- doesn't consider term frequency - Rare terms are more informative than frequent terms.

Frequency incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth ...
ANTHONY	157	73	0	0	0	1
BRUTUS	4	157	0	2	0	0
CAESAR	232	227	0	2	1	0
CALPURNIA	0	10	0	0	0	0
CLEOPATRA	57	0	0	0	0	0
MERCY	2	0	3	8	5	8
WORSER	2	0	1	1	1	5
...						

Each document is now represented as a count vector $\in \mathbb{N}^{|V|}$.

Bag of words model

- We do not consider the **order** of words in a document.
- *“Mike is heavier than Molly”* and *“Molly is heavier than Mike”* are represented the same way.
- In a sense, this is a step back: The positional index was able to distinguish these two documents.

Term Frequency (1)

- **term frequency** $tf(t,d)$,
 - simplest choice: use the *raw frequency* of a term in a document, i.e. $tf(t,d) = f(t,d)$: the number of times that term t occurs in document d ;
 - Other possibilities:
- [boolean](#) "frequencies": $tf(t,d) = 1$ if t occurs in d and 0 otherwise;
- [logarithmically](#) scaled frequency: $tf(t,d) = 1 + \log f(t,d)$ (and 0 when $f(t,d) = 0$);
- [normalized frequency](#),
$$tf(t,d) = \frac{f(t,d)}{\max\{f(w,d) : w \text{ in } d\}}$$
- raw frequency divided by the maximum raw frequency of any term in the document.
 - prevent a bias towards longer documents,.

Term frequency: Log frequency weighting

- The log frequency weight of term t in d is defined as follows

$$w_{t,d} = \begin{cases} 1 + \log_{10} \text{tf}_{t,d} & \text{if } \text{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- $\text{tf}_{t,d} \rightarrow w_{t,d}$: $0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 1.3, 10 \rightarrow 2, 1000 \rightarrow 4, \dots$

- Score for a query document (q,d) pair:

$$\text{tf-matching-score}(q, d) = \sum_{t \in q \cap d} (1 + \log \text{tf}_{t,d})$$

- The score is 0 if none of the query terms is present in the document.

Frequency in document vs. frequency in collection

In addition, to term frequency we care about term frequency at **collection** level for weighting and ranking.

Rare terms are more informative than frequent terms.

- Consider a rare query term (e.g., **Hypermnesia**).
- A document containing this term is very likely to be relevant.
- We want high weights for rare terms like **Hypermnesia**

Frequent terms

- are less informative (i.e. **GOOD, INCREASE, LINE**).
- A document containing this term is more likely to be relevant than a document that doesn't . . .
- For frequent terms like **GOOD, INCREASE** and **LINE**, we assign positive weights but lower than for rare ones.

Document frequency

- high weights for rare terms like Hypermnnesia
- low (positive) weights for frequent words like GOOD, INCREASE and LINE.
- Factor document frequency into the matching score.
- The document frequency df_t is # of documents in the collection that the term occurs in.
- df_t is an inverse measure of the informativeness of term t .
- We define the idf weight of term t as: $idf_t = \log_{10} \frac{N}{df_t}$
(N : # documents in the collection.)
- idf_t is a measure of the informativeness of the term.
- $[\log N/df_t]$ instead of $[N/df_t]$ to “dampen” the effect of idf

Idf - examples

- Compute idf_t using the formula:

$$\text{idf}_t = \log_{10} \frac{1,000,000}{\text{df}_t}$$

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

Effect of idf on ranking

- idf affects the ranking of documents for **queries with at least two terms**.
- for example, in the query “*hypertension feature*”, idf weighting **increases** the relative weight of “*hypertension*” and **decreases** the relative weight of “*feature*”.
- idf has **little effect** on ranking for **one-term queries**.

tf-idf weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight.

$$w_{t,d} = (1 + \log \text{tf}_{t,d}) \cdot \log \frac{N}{\text{df}_t}$$

- increases with the number of occurrences within a document. (tf)
- increases with the rarity of the term in the collection. (idf)
- Best known weighting scheme in information retrieval

Count matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth ...
ANTHONY	157	73	0	0	0	1
BRUTUS	4	157	0	2	0	0
CAESAR	232	227	0	2	1	0
CALPURNIA	0	10	0	0	0	0
CLEOPATRA	57	0	0	0	0	0
MERCY	2	0	3	8	5	8
WORSER	2	0	1	1	1	5
...						

Each document is now represented as a count vector $\in \mathbb{N}^{|V|}$.

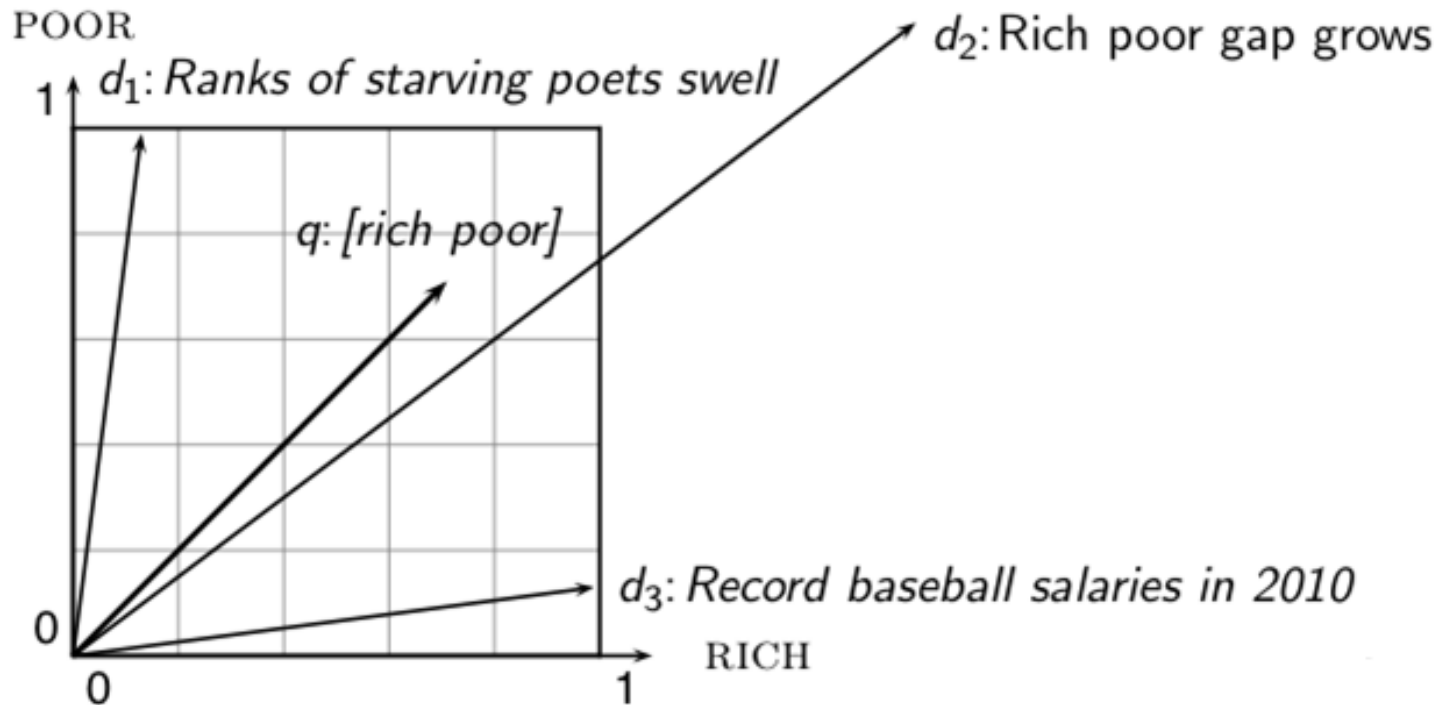
Binary \rightarrow count \rightarrow weight matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth ...
--	-----------------------------	------------------	----------------	--------	---------	----------------

ANTHONY	5.25	3.18	0.0	0.0	0.0	0.35
BRUTUS	1.21	6.10	0.0	1.0	0.0	0.0
CAESAR	8.59	2.54	0.0	1.51	0.25	0.0
CALPURNIA	0.0	1.54	0.0	0.0	0.0	0.0
CLEOPATRA	2.85	0.0	0.0	0.0	0.0	0.0
MERCY	1.51	0.0	1.90	0.12	5.25	0.88
WORSER	1.37	0.0	0.11	4.15	0.25	1.95
...						

Each document is now represented as a real-valued vector of tf x idf weights $\in \mathbb{R}^{|V|}$.

Why distance is a bad idea

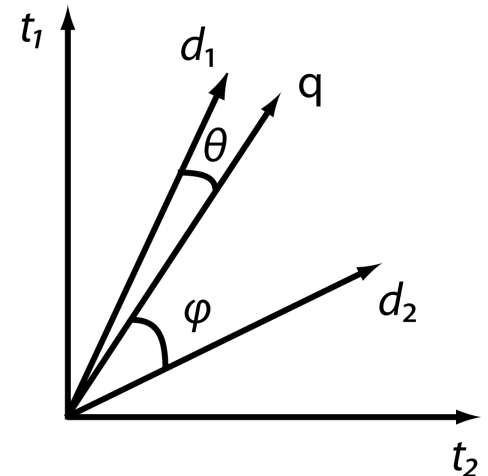


The Euclidean distance of \vec{q} and \vec{d}_2 is large although the distribution of terms in the query q and the distribution of terms in the document d_2 are very similar.

Vector Space Model

- Document ***d*** and query ***q*** are represented as k-dimensional vectors $d = (w_{1,d}, \dots, w_{k,d})$ and $q = (w_{1,q}, \dots, w_{k,q})$
 - Each dimension corresponds to a term from the collection vocabulary
 - Independence between terms
 - $w_{i,q}$ is the weight of i-th vocabulary word in ***q***
- Is Euclidean distance appropriate to measure the similarity?
 - Vector (a, b) and (10 x a, 10 x b) contain the same words but have large Euclidean distance
- Degree of similarity between ***d*** and ***q*** is the cosine of the angle between the two vectors

$$\text{sim}(d, q) = \frac{d \cdot q}{|d||q|} = \frac{\sum_{t=1}^k w_{t,d} \times w_{t,q}}{\sqrt{\sum_{t=1}^k w_{t,d}^2} \times \sqrt{\sum_{t=1}^k w_{t,q}^2}}$$



Term weighting in VSM

- Term weighting with tf-idf (and variations)

$$w_{t,d} = tf_{t,d} \times idf_t$$

- tf models the importance of a term in a document

$$tf_{t,d} = f_{t,d} \quad tf_{t,d} = \frac{f_{t,d}}{\max(f_{s,d})}$$

- $f_{t,d}$ is the frequency of term t in document d

- idf models the importance of a term in the document collection

- Logarithm base not important
 - Information content of event “term t occurs in document d ”

$$idf_t = -\log P(t \text{ occurs in } d) = -\log \frac{n_t}{N} = \log \frac{N}{n_t} \quad idf_t = \log \frac{N - n_t + 0.5}{n_t + 0.5}$$

- N is the total number of documents, n_t is the document frequency of term t

Ranked retrieval in the vector space model

- Represent the query as a weighted tf-idf vector
- Represent each document as a weighted tf-idf vector
- Compute the cosine similarity between the query vector and each document vector
- Rank documents with respect to the query
- Return the top K (e.g., $K = 10$) to the user

Example - TFIDF (1)

- Doc1: Computer Science is the scientific field that studies computers
- Doc2: Decision Support Systems support enterprises in decisions
- Doc3: Information Systems are based on Computer Science
- Dictionary/Dimensions:{computer, science, field, studies, decision, support, systems, enterprises, information, based}

TF:

computer	science	field	studies	decision	support	systems	enterprises	information	based
2/6	2/6	1/6	1/6	0	0	0	0	0	0
0	0	0	0	2/6	2/6	1/6	1/6	0	0
1/5	1/5	0	0	0	0	1/5	0	1/5	1/5

IDF:

computer	science	field	studies	decision	support	systems	enterprises s	information	based
0.301	0.301	0.602	0.602	0.602	0.602	0.301	0.602	0.602	0.602

Example - TFIDF (2)

TFIDF:

computer	science	field	studies	decision	support	systems	enterprises	information	based
0.1	0.1	0.1	0.1	0	0	0	0	0	0
0	0	0	0	0.2	0.2	0.05	0.05	0	0
0.06	0.06	0	0	0	0	0.06	0	0.12	0.12

Queries

- Each query is considered as a new document and therefore it can be represented as a vector. Then the k most similar documents are retrieved
- Query = {Information Systems}

Collection:

computer	science	field	studies	decision	support	systems	enterprises	information	based
0.1	0.1	0.1	0.1	0	0	0	0	0	0
0	0	0	0	0.2	0.2	0.05	0.05	0	0
0.06	0.06	0	0	0	0	0.06	0	0.12	0.12

Query:

computer	science	field	studies	decision	support	systems	enterprises	information	based
0	0	0	0	0	0	1	0	1	0

	Distance	I.P	$\cos(\varphi)$
Doc 1	1.43	0	0
Doc 2	1.40	0.05	0.40
Doc 3	1.34	0.18	0.64

BM25 Ranking Function

- Ranking function assuming **bag-of-words** document representation

$$score(d, q) = \sum_{t \in d \cap q} idf_t \times \frac{tf_{t,d} \cdot (k_1 + 1)}{tf_{t,d} + k_1 \cdot \left(1 - b + b \frac{len_d}{avglen} \right)}$$

- len_d is the length of document d
 - $avglen$ is the average document length in the collection
- Score depends only on query terms
- Values of parameters k_1 and b depend on collection/task
 - k_1 controls term frequency saturation
 - b controls length normalization
 - Default values: $k_1 = 1.2$ and $b = 0.75$

Text retrieval evaluation

Text retrieval evaluation

- Typical evaluation setting
 - Set of documents
 - Set of information needs, expressed as queries (typically 50 or more)
 - Relevance assessments specifying for each query the relevant and non-relevant documents

Evaluating unranked results

	Relevant	Non-relevant
Retrieved	True positives (tp)	False positives (fp)
Not retrieved	False negatives (fn)	True negatives (tn)

$$\text{Precision} = \frac{\#(\text{Relevant documents retrieved})}{\#(\text{Retrieved documents})} = \frac{tp}{(tp + fp)}$$

$$\text{Recall} = \frac{\#(\text{Relevant documents retrieved})}{\#(\text{Relevant documents})} = \frac{tp}{(tp + fn)}$$

Precision/recall trade-off

- You can increase recall by returning more docs.
- Recall is a non-decreasing function of the number of docs retrieved.
- A system that returns all docs has 100% recall!
- The converse is also true (usually): It's easy to get high precision for very low recall.

A combined measure: F

- F allows us to trade off precision against recall.

$$F = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha) \frac{1}{R}} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \qquad \beta^2 = \frac{1 - \alpha}{\alpha}$$

- $\alpha \in [0, 1]$ and thus $\beta^2 \in [0, \infty]$
- Most frequently used: **balanced F** with $\beta = 1$ or $\alpha = 0.5$
 - This is the **harmonic mean** of P and R : $\frac{1}{F} = \frac{1}{2}(\frac{1}{P} + \frac{1}{R})$

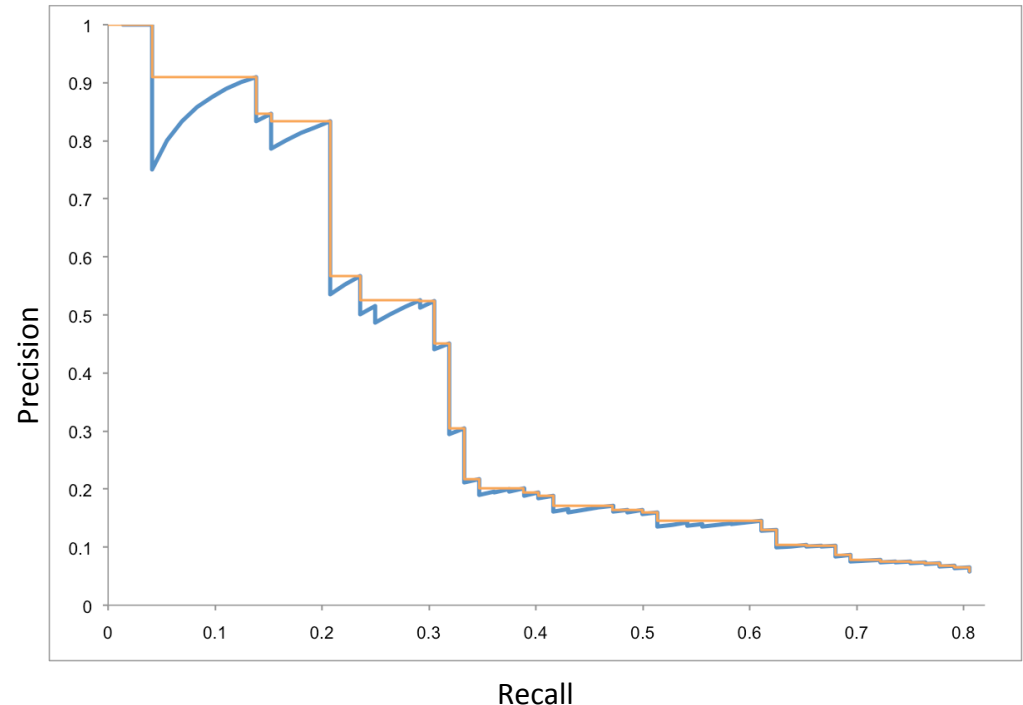
F: Example

	relevant	not relevant	
retrieved	20	40	60
not retrieved	60	1,000,000	1,000,060
	80	1,000,040	1,000,120

- $P = 20/(20 + 40) = 1/3$
- $R = 20/(20 + 60) = 1/4$
- $F_1 = 2 \frac{1}{\frac{1}{3} + \frac{1}{4}} = 2/7$

Evaluating ranked results

Rank	Relevant?	Precision	Recall	Interpolated precision
1	1	1,00	0,01	1,00
2	1	1,00	0,03	1,00
3	1	1,00	0,04	1,00
4	0	0,75	0,04	0,91
5	1	0,80	0,06	0,91
6	1	0,83	0,07	0,91
7	1	0,86	0,08	0,91
8	1	0,88	0,10	0,91
9	1	0,89	0,11	0,91
10	1	0,90	0,13	0,91
11	1	0,91	0,14	0,91
12	0	0,83	0,14	0,85
13	1	0,85	0,15	0,85
14	0	0,79	0,15	0,83
15	1	0,80	0,17	0,83
...

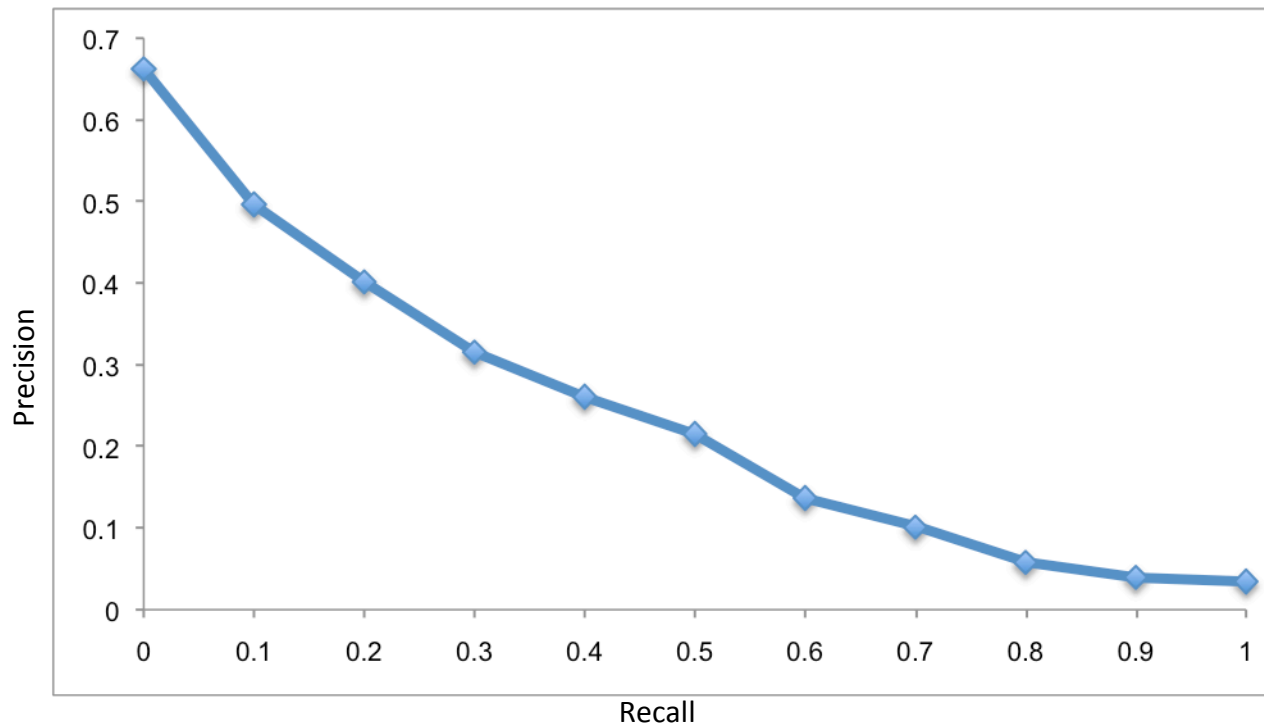


Interpolated precision at recall level r :

- maximum precision at any recall level equal or greater than r
- Defined for any recall level in $[0.0, 1.0]$

Evaluating ranked results

- Average 11-point interpolated precision for 50 queries



Evaluating ranked results

- Average Precision (AP)

- average of precision after each relevant document is retrieved
- Example: $AP = 1/1 + 2/3 + 3/5 + 4/7 = 0.7095$
- Mean Average Precision (MAP)

- Precision at K

- precision after K documents have been retrieved
- Example: Precision at 10 = $4/10 = 0.4000$

Rank	Relevant?
1	1
2	0
3	1
4	0
5	1
6	0
7	1
8	0
9	0
10	0

- R-Precision

- For a query with R relevant documents, compute precision at R
- Example: for a query with $R=7$ relevant docs,
R-Precision = $4/7 = 0.5714$

Non-binary relevance

- So far, relevance has been binary
 - a document is either relevant or non-relevant
- The degree to which a document satisfies the user's information need varies
 - Perfect match: $rel = 3$
 - Good match: $rel = 2$
 - Marginal match: $rel = 1$
 - Bad match: $rel = 0$
- Evaluate systems assuming that
 - highly relevant documents are more useful than marginally relevant documents, which are more useful than non-relevant ones
 - highly relevant documents are more useful when having higher ranks in the search engine results

Discounted Cumulative Gain

- Cumulative Gain (CG) at rank position p

$$CG_p = \sum_{i=1}^p rel_i$$

- Independent of the order of results among the top- p positions

- Discounted Cumulative Gain(DCG) at rank position p

$$DCG_p = rel_1 + \sum_{i=2}^p \frac{rel_i}{\log_2 i}$$

- Normalized Discounted Cumulative Gain (nDCG) at rank position p

- allows comparison of performance across queries
- compute ideal DCG_p ($IDCG_p$) from perfect ranking of documents in decreasing order of relevance

$$nDCG_p = \frac{DCG_p}{IDCG_p}$$

Latent Semantic Indexing

- Understand SVD
- Apply to the doc/term matrix
- Answering queries

Dim. Reduction–Eigenvectors

A nxn matrix

- eigenvalues λ : $|A-\lambda I|=0$
- Eigenvectors x : $Ax=\lambda x$
- Matrix rank: # linearly independent rows or columns
- A real symmetric table A nxn can be expressed as: $A=U\Lambda U^T$
- U's columns are A's eigenvectors
- Λ 's diagonal contains A's eigenvalues
- $A=U\Lambda U^T=\lambda_1 x_1 x_1^T + \lambda_2 x_2 x_2^T + \dots + \lambda_n x_n x_n^T$
- $x_1 x_1^T$ represents projection via x_1 (λ_i eigenvalue, x_i eigenvector)
- XX^T vs. $X^T X$

Singular Value Decomposition (SVD)

Eigen values and eigenvectors decomposition is applied to square matrices. For non square matrices we apply **Singular Value Decomposition**.

Let **X** a **$m \times n$** table, $X = U\Sigma V^T$

U : orthogonal $m \times m$, its columns are the eigenvectors of XX^T .

U, V define orthogonal basis: $U^T U = V V^T = 1$

Σ : $m \times n$ contains A 's singular values (square roots of XX^T eigenvalues)

V : $n \times n$, its columns are the eigenvectors of $X^T X$

Singular Value Decomposition (SVD) -

I

PROOF:

$$X = U\Sigma V^T, X^T = V\Sigma^T U^T \rightarrow XX^T = U\Sigma(V^T V)\Sigma^T U^T \rightarrow XX^T = U\Sigma\Sigma^T U^T$$

$$\text{Similarly: } \rightarrow X^T X = U\Sigma(V^T V)\Sigma^T U^T \text{ therefore } X^T X = V\Sigma^T \Sigma U^T$$

Hence: U : eigenvectors of XX^T , V : eigenvectors of $X^T X$ and Σ sqrt of MM^T (or $M^T M$) eigenvalues

Let X a $m \times n$ table, $X = U\Sigma V^T$ then an r rank approximation of X is:

$$Y = U_{m \times r} \text{diag}(\sigma_1, \dots, \sigma_r) V_{n \times r}^T$$

Singular Value Decomposition (SVD) - II

Matrix approximation $X_k = U_k \Sigma_k V_k^T$

- The best rank r approximation X_k of a matrix X
- minimizing the [Frobenius norm](#) distance:

$$\|X\|_F^2 = \sum_i \sum_j x_{ij}^2 = \text{trace}(\mathbf{X}^* \mathbf{X})$$

•

•

SVD application - Latent Structure in documents

- Documents are represented based on the Vector Space Model
- Vector space model consists of the keywords contained in a document.
- In many cases baseline keyword based performs poorly – not able to detect synonyms.
- Therefore document clustering is problematic
- Example where of keyword matching with the query: “IDF in computer-based information look-up”

	access	document	retrieval	information	theory	database	indexing	computer
Doc1	X	X	X			X	X	
Doc2				X	X			X
Doc3			X	X				X

Latent Semantic Indexing (LSI) -I

- Finding similarity with exact keyword matching is problematic.
- Using SVD we process the initial document-term document.
- Then we choose the k larger singular values. The resulting matrix is of order k and is the most similar to the original one based on the Frobenius norm than any other k -order matrix.

Latent Semantic Indexing (LSI) - II

- The initial matrix is SVD decomposed as: $A = ULV^T$
- Choosing the top-k singular values from L we have:

$$A_k = U_k L_k V_k^T ,$$

- L_k is square $k \times k$ containing the top-k singular values of the diagonal in matrix L,
- U_k , the $m \times k$ matrix containing the first k columns in U (left singular vectors)
- V_k^T , the $k \times n$ matrix containing the first k lines of V^T (right singular vectors)

Typical values for $k \sim 200-300$ (empirically chosen based on experiments appearing in the bibliography)

LSI capabilities

- - Term to term similarity: $A_k A_k^T$
- - Document-document similarity: $A_k^T A_k$
- - Term document similarity (as an element of the transformed – document matrix)
- - Extended query capabilities transforming initial query q to q_n : $q_n = q^T U_k L_k^{-1}$
- - Thus q_n can be regarded a line in matrix V_k

LSI – an example

LSI application on a term – document matrix

C1: Human machine Interface for Lab ABC computer application

C2: A survey of user opinion of computer system response time

C3: The EPS user interface management system

C4: System and human system engineering testing of EPS

C5: Relation of user-perceived response time to error measurements

M1: The generation of random, binary unordered trees

M2: The intersection graph of path in trees

M3: Graph minors IV: Widths of trees and well-quasi-ordering

M4: Graph minors: A survey

- The dataset consists of 2 classes, 1st: “human – computer interaction” (c1-c5)
2nd: related to graph (m1-m4). After feature extraction the titles are represented as follows.

LSI – an example

$$A = ULV^T$$

U=

0.22	-0.11	0.29	-0.41	-0.11	-0.34	0.52	-0.06	-0.41	0	0	0
0.20	-0.07	0.14	-0.55	0.28	0.50	-0.07	-0.01	-0.11	0	0	0
0.24	0.04	-0.16	-0.59	-0.11	-0.25	-0.30	0.06	0.49	0	0	0
0.40	0.06	-0.34	0.10	0.33	0.38	0.00	0.00	0.01	0	0	0
0.64	-0.17	0.36	0.33	-0.16	-0.21	-0.17	0.03	0.27	0	0	0
0.27	0.11	-0.43	0.07	0.08	-0.17	0.28	-0.02	-0.05	0	0	0
0.27	0.11	-0.43	0.07	0.08	-0.17	0.28	-0.02	-0.05	0	0	0
0.30	-0.14	0.33	0.19	0.11	0.27	0.03	-0.02	-0.17	0	0	0
0.21	0.27	-0.18	-0.03	-0.54	0.08	-0.47	-0.04	-0.58	0	0	0
0.01	0.49	0.23	0.03	0.59	-0.39	-0.29	0.25	-0.23	0	0	0
0.04	0.62	0.22	0.00	-0.07	0.11	0.16	-0.68	0.23	0	0	0
0.03	0.45	0.14	-0.01	-0.30	0.28	0.34	0.68	0.18	0	0	0

LSI – an example

$$A=ULV^T$$

$V=$

0.20	-0.06	0.11	-0.95	0.05	-0.08	0.18	-0.01	-0.06
0.61	0.17	-0.50	-0.03	-0.21	-0.26	-0.43	0.05	0.24
0.46	-0.13	0.21	0.04	0.38	0.72	-0.24	0.01	0.02
0.54	-0.23	0.57	0.27	-0.21	-0.37	0.26	-0.02	-0.08
0.28	0.11	-0.51	0.15	0.33	0.03	0.67	-0.06	-0.26
0.00	0.19	0.10	0.02	0.39	-0.30	-0.34	0.45	-0.62
0.01	0.44	0.19	0.02	0.35	-0.21	-0.15	-0.76	0.02
0.02	0.62	0.25	0.01	0.15	0.00	0.25	0.45	0.52
0.08	0.53	0.08	-0.03	-0.60	0.36	0.04	-0.07	-0.45

LSI – an example

Choosing the 2 largest singular values we have

$$U_k =$$

0.22	-0.11
0.20	-0.07
0.24	0.04
0.40	0.06
0.64	-0.17
0.27	0.11
0.27	0.11
0.30	-0.14
0.21	0.27
0.01	0.49
0.04	0.62
0.03	0.45

$$L_k =$$

3.34	0
0	2.54

$$V_k^T =$$

0.20	0.61	0.46	0.54	0.28	0.00	0.02	0.02	0.08
-0.06	0.17	-0.13	-0.23	0.11	0.19	0.44	0.62	0.53

LSI (2 singular values)

$A_k =$

	C1	C2	C3	C4	C5	M1	M2	M3	M4
human	0.16	0.40	0.38	0.47	0.18	-0.05	-0.12	-0.16	-0.09
Interface	0.14	0.37	0.33	0.40	0.16	-0.03	-0.07	-0.10	-0.04
Computer	0.15	0.51	0.36	0.41	0.24	0.02	0.06	0.09	0.12
User	0.26	0.84	0.61	0.70	0.39	0.03	0.08	0.12	0.19
System	0.45	1.23	1.05	1.27	0.56	-0.07	-0.15	-0.21	-0.05
Response	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
Time	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
EPS	0.22	0.55	0.51	0.63	0.24	-0.07	-0.14	-0.20	-0.11
Survey	0.10	0.53	0.23	0.21	0.27	0.14	0.31	0.44	0.42
Trees	-0.06	0.23	-0.14	-0.27	0.14	0.24	0.55	0.77	0.66
Graph	-0.06	0.34	-0.15	-0.30	0.20	0.31	0.69	0.98	0.85
Minors	-0.04	0.25	-0.10	-0.21	0.15	0.22	0.50	0.71	0.62

LSI Example

- Query: “human computer interaction” retrieves documents: c_1, c_2, c_4 but *not* c_3 and c_5 .
- submit the same query (based on the transformation shown before) to the transformed matrix we retrieve (using cosine similarity) all c_1 - c_5 even if c_3 and c_5 have no common keyword to the query.
- According to the transformation for the queries we have:

Query transformation

	query
human	1
Interface	0
computer	1
User	0
System	0
Response	0
Time	0
EPS	0
Survey	0
Trees	0
Graph	0
Minors	0

 $q =$ [illegible]

Query transformation

$$q^T = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$U_k = \begin{bmatrix} 0.22 & -0.11 \\ 0.20 & -0.07 \\ 0.24 & 0.04 \\ 0.40 & 0.06 \\ 0.64 & -0.17 \\ 0.27 & 0.11 \\ 0.27 & 0.11 \\ 0.30 & -0.14 \\ 0.21 & 0.27 \\ 0.01 & 0.49 \\ 0.04 & 0.62 \\ 0.03 & 0.45 \end{bmatrix}$$

$$L_k = \begin{bmatrix} 0.3 & 0 \\ 0 & 0.39 \end{bmatrix}$$

$$q_n = q^T U_k L_k = \begin{bmatrix} 0.138 & -0.0273 \end{bmatrix}$$

Query transformation

Map
docs to
the 2
dim
space
 $V_k L_k =$

0.20	-0.06
0.61	0.17
0.46	-0.13
0.54	-0.23
0.28	0.11
0.00	0.19
0.01	0.44
0.02	0.62
0.08	0.53

3.34	0
0	2.54

=

0.67	-0.15
2.04	0.43
1.54	-0.33
1.80	-0.58
0.94	0.28
0.00	0.48
0.03	1.12
0.07	1.57
0.27	1.35

$q_n L_k =$

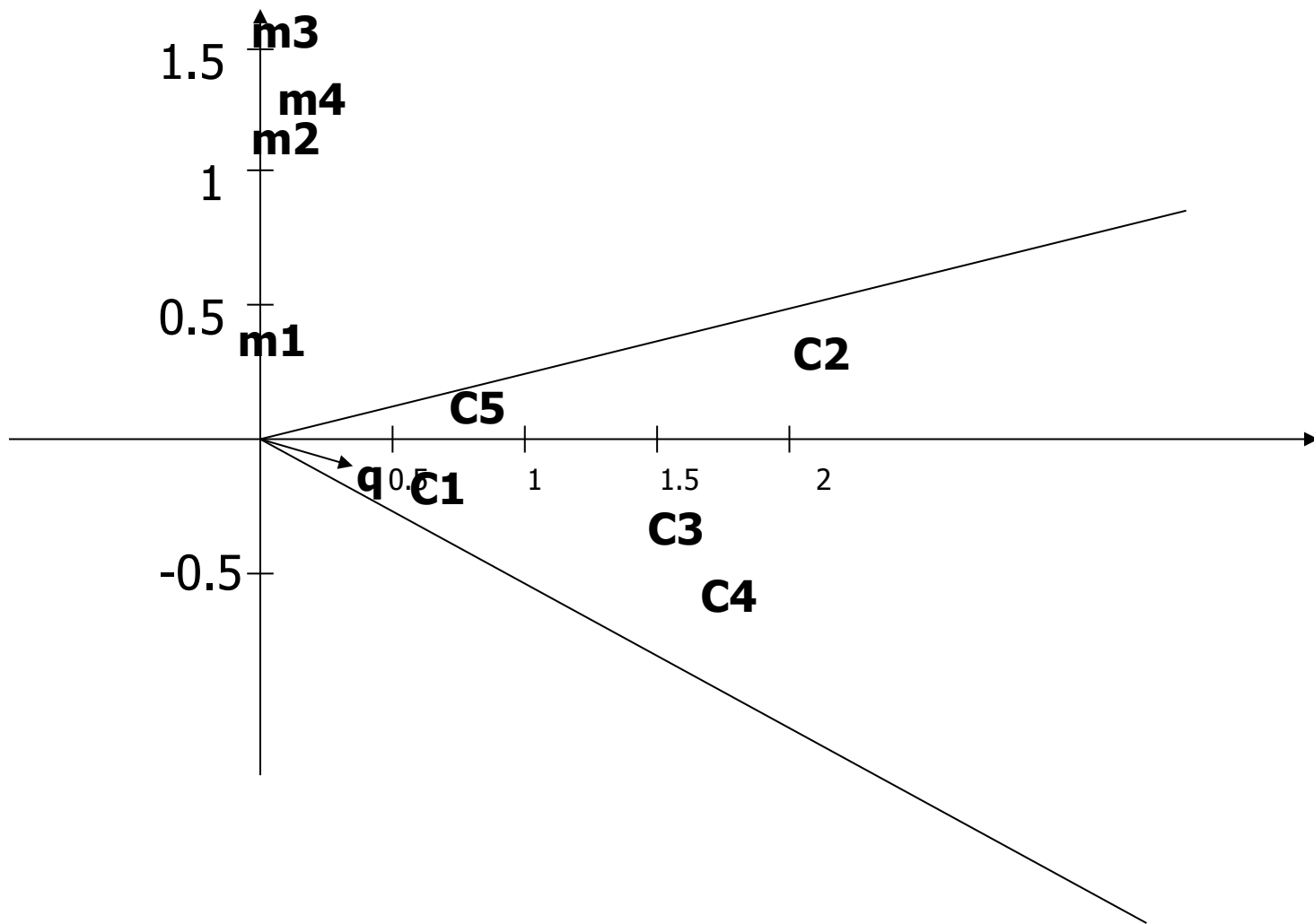
0.138	-0.0273
-------	---------

3.34	0
0	2.54

=

0.46	-0.069
------	--------

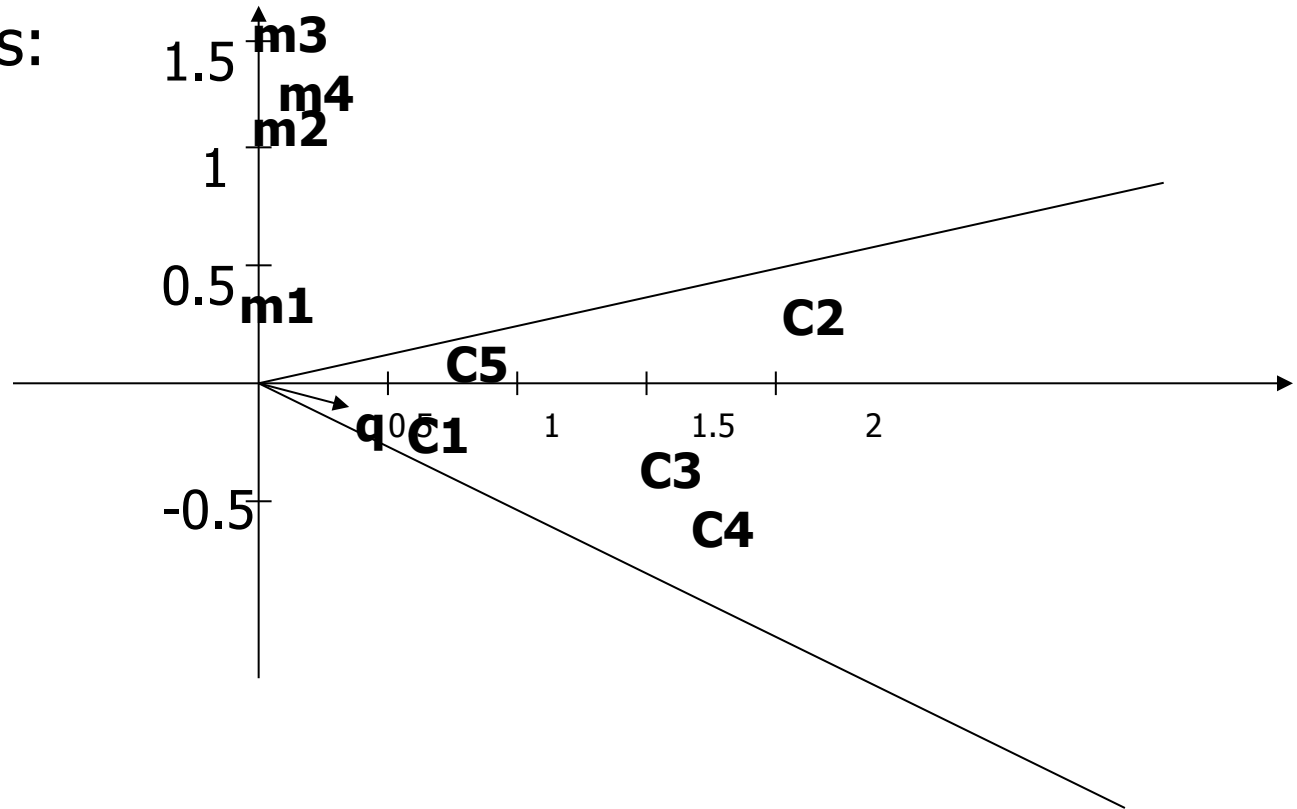
Query transformation



Query transformation

- The cosine similarity matrix of query vector to the documents is:

	query
C1	0.99
C2	0.94
C3	0.99
C4	0.99
C5	0.90
M1	-0.14
M2	-0.13
M3	-0.11
M4	0.05



References

- Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008.
<http://www-nlp.stanford.edu/IR-book/>
- “Indexing by Latent Semantic Analysis”, S.Deerwester, S.Dumais, T.Landauer, G.Fumas, R.Harshman, Journal of the Society for Information Science, 1990
- “Mining the Web: Discovering Knowledge from Hypertext Data”, Soumen Chakrabarti

GoW - Relevant publications

- ① Rousseau, François and Michalis Vazirgiannis (2013a). Composition of TF Normalizations: New Insights on Scoring Functions for Ad Hoc IR. In *Proceedings of the 36th annual international ACM SIGIR conference on Research and development in Information Retrieval*. SIGIR '13, pp. 917–920.
- ② Rousseau, François and Michalis Vazirgiannis (2013b). Graph-of-word and TW-IDF: New Approach to Ad Hoc IR. In *Proceedings of the 22nd ACM International conference on Information and Knowledge Management*. CIKM '13, pp. 59–68.
- ③ Rousseau, François and Michalis Vazirgiannis (2015). Main Core Retention on Graph-of-words for Single-Document Keyword Extraction. In *Proceedings of the 37th European Conference on Information Retrieval*. ECIR '15, To appear.
- ④ Degeneracy-based Real-Time Sub-Event Detection in Twitter Stream. Polykarpos Meladianos, Giannis Nikolentzos, François Rousseau, Yannis Stavrakas and Michalis Vazirgiannis. *Proceedings of the 9th AAAI International Conference on Web and Social Media (ICWSM '15)*.