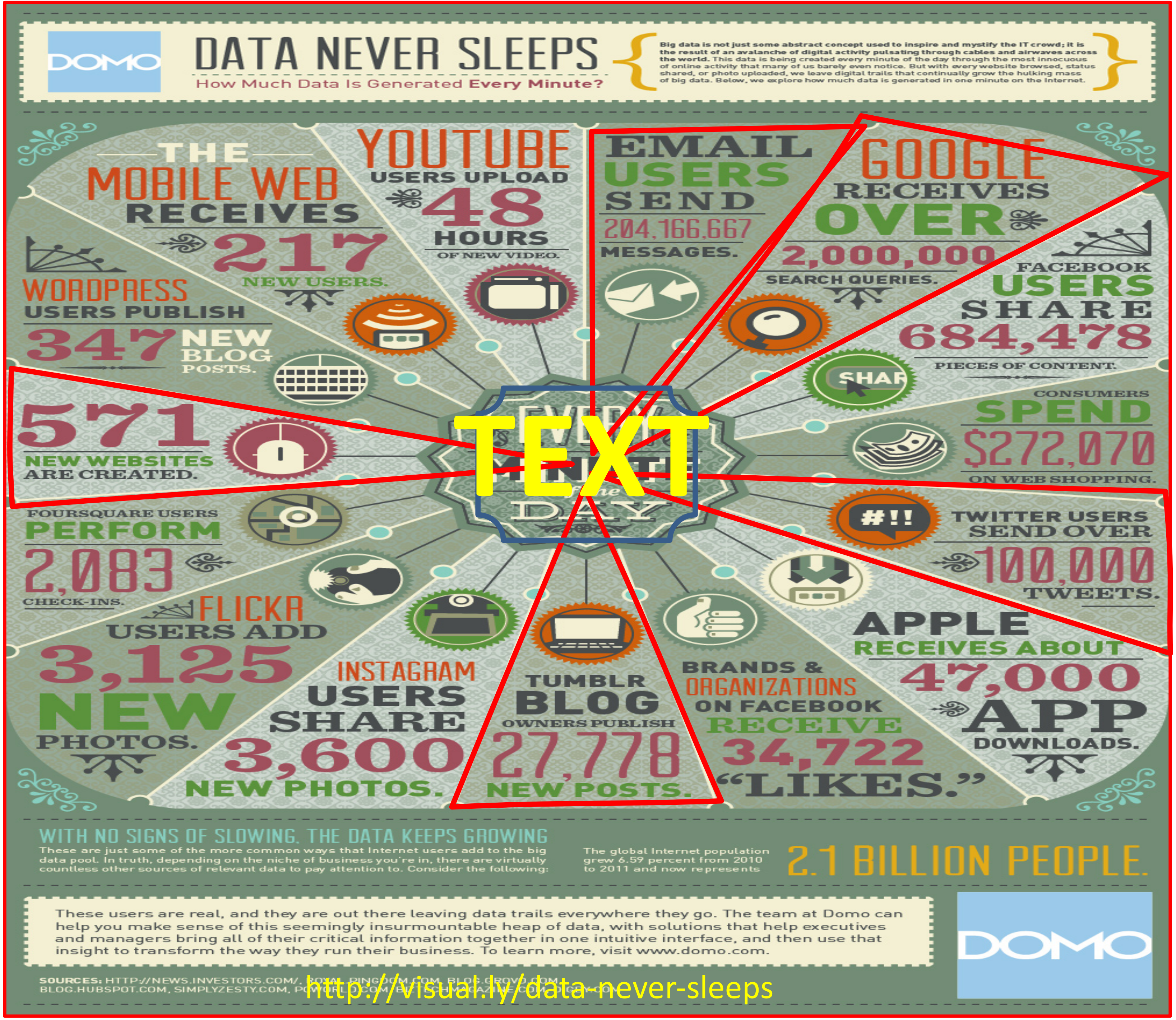
Text Mining – an introduction

Michalis Vazirgiannis

LIX @ Ecole Polytechnique

January 2017



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.

THE MOBILE WEB RECEIVES 217 NEW USERS.

YOUTUBE USERS UPLOAD 48 HOURS OF NEW VIDEO.

EMAIL USERS SEND 204,166,667 MESSAGES.

GOOGLE RECEIVES OVER 2,000,000 SEARCH QUERIES.

FACEBOOK USERS SHARE 684,478 PIECES OF CONTENT.

CONSUMERS SPEND $272,070 ON WEB SHOPPING.

TWITTER USERS SEND OVER 100,000 TWEETS.

APPLE RECEIVES ABOUT 47,000 APP DOWNLOADS.

BRANDS & ORGANIZATIONS ON FACEBOOK RECEIVE 34,722 "LIKES."

TUMBLR BLOG OWNERS PUBLISH 27,778 NEW POSTS.

INSTAGRAM USERS SHARE 3,600 NEW PHOTOS.

FLICKR USERS ADD 3,125 NEW PHOTOS.

FOURSQUARE USERS PERFORM 2,083 CHECK-INS.

571 NEW WEBSITES ARE CREATED.

WORDPRESS USERS PUBLISH 347 NEW BLOG POSTS.

EVERY MINUTE of the DAY

TEXT

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.

SOURCES: HTTP://NEWS.INVESTORS.COM/, ROYAL.PINGDOM.COM, BLOG.GROVO.COM, BLOG.HUBSPOT.COM, SIMPLYZESTY.COM, PCWORLD.COM

http://visual.ly/data-never-sleeps

Outline

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

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

	This	is	a	the	database	lab	of	IS	master	course
Text 1:	1	1	0	1	1	1	1	1	1	1
Text 2:	1	1	1	0	1	0	0	0	0	1

Vector Space Model

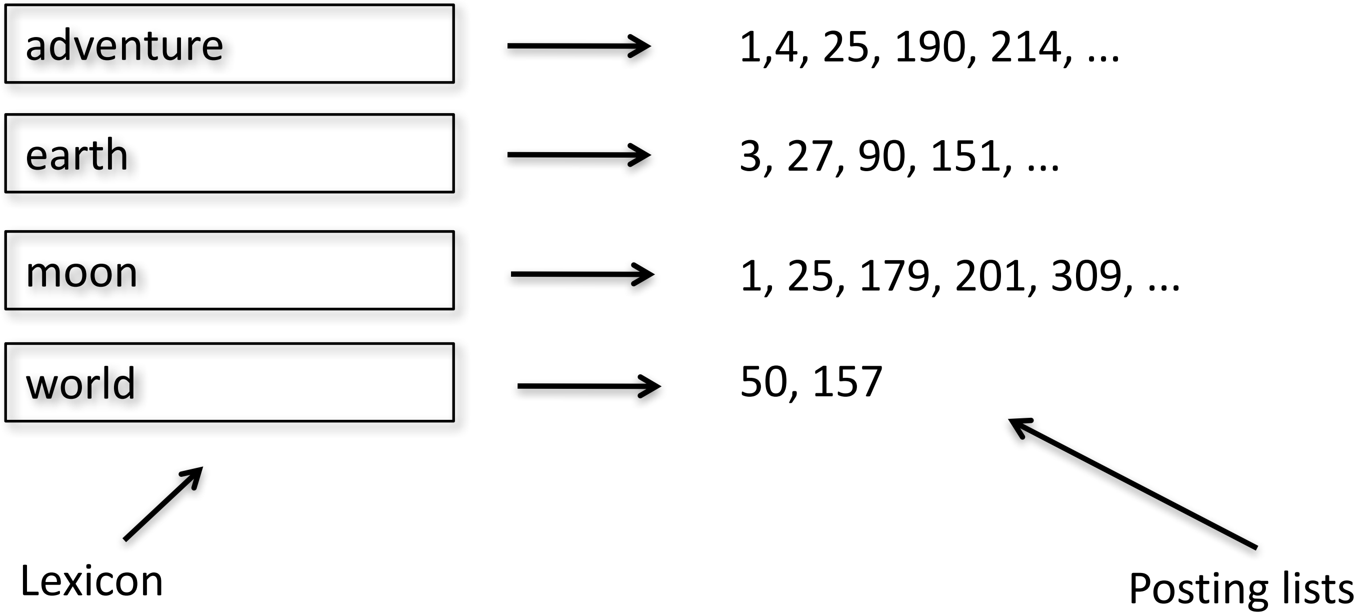
- Vector Space Model:
  - To VSM represents the significance of each term for each document
    - The cell values are computed based on the terms' frequency
    - Most common approach is TF/IDF

Feature Selection

	This	is	a	the	database	lab	of	IS	master	course
Text 1:	0.1	0.1	0	0.2	0.1	0.1	0.1	0.1	0.1	0.1
Text 2:	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



adventure → 1,4, 25, 190, 214, ...

earth → 3, 27, 90, 151, ...

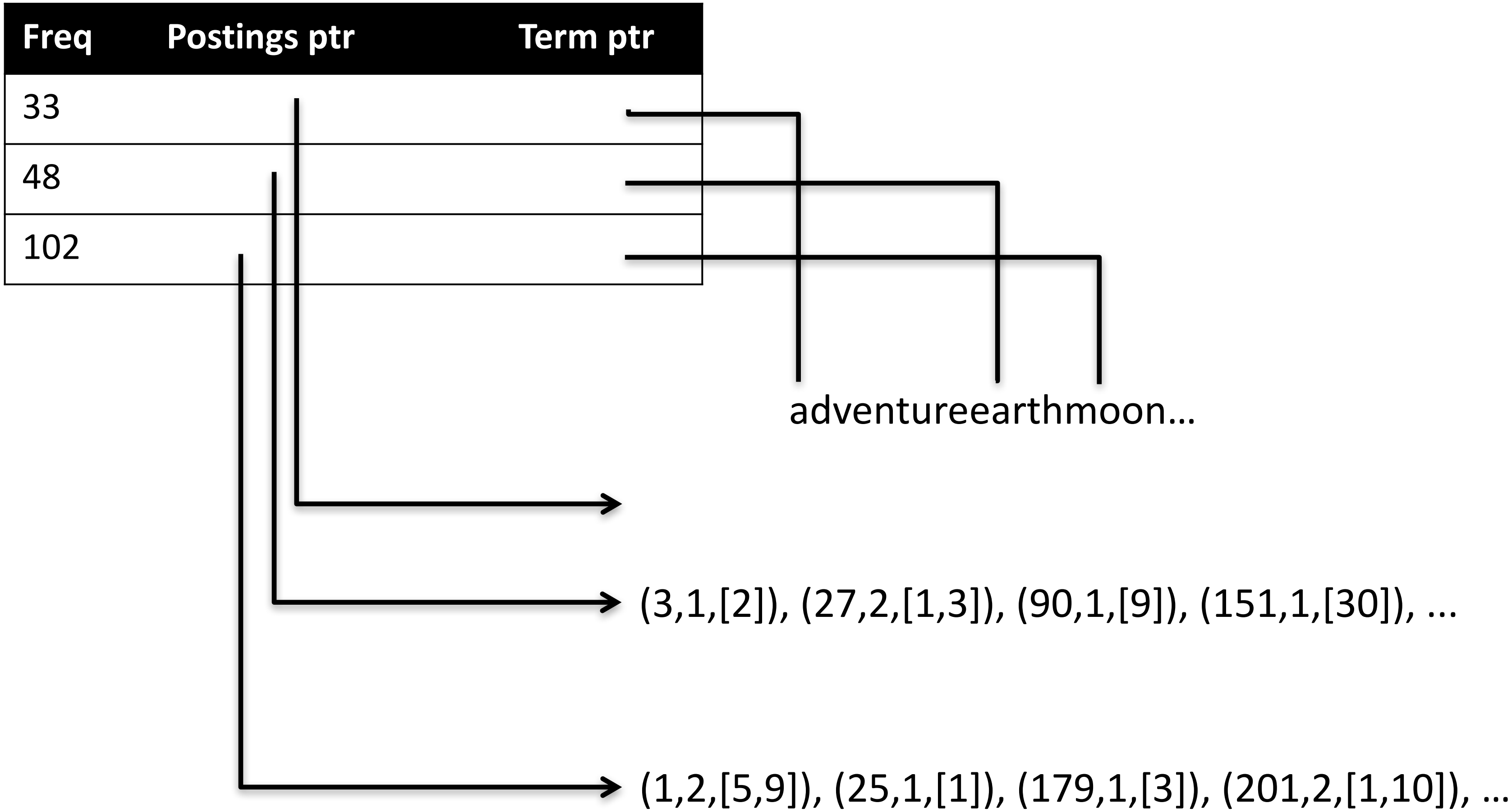
moon → 1, 25, 179, 201, 309, ...

world → 50, 157

Lexicon

Posting lists

Dictionary as a String



Freq	Postings ptr	Term ptr
33		
48		
102		

adventureearthmoon…

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

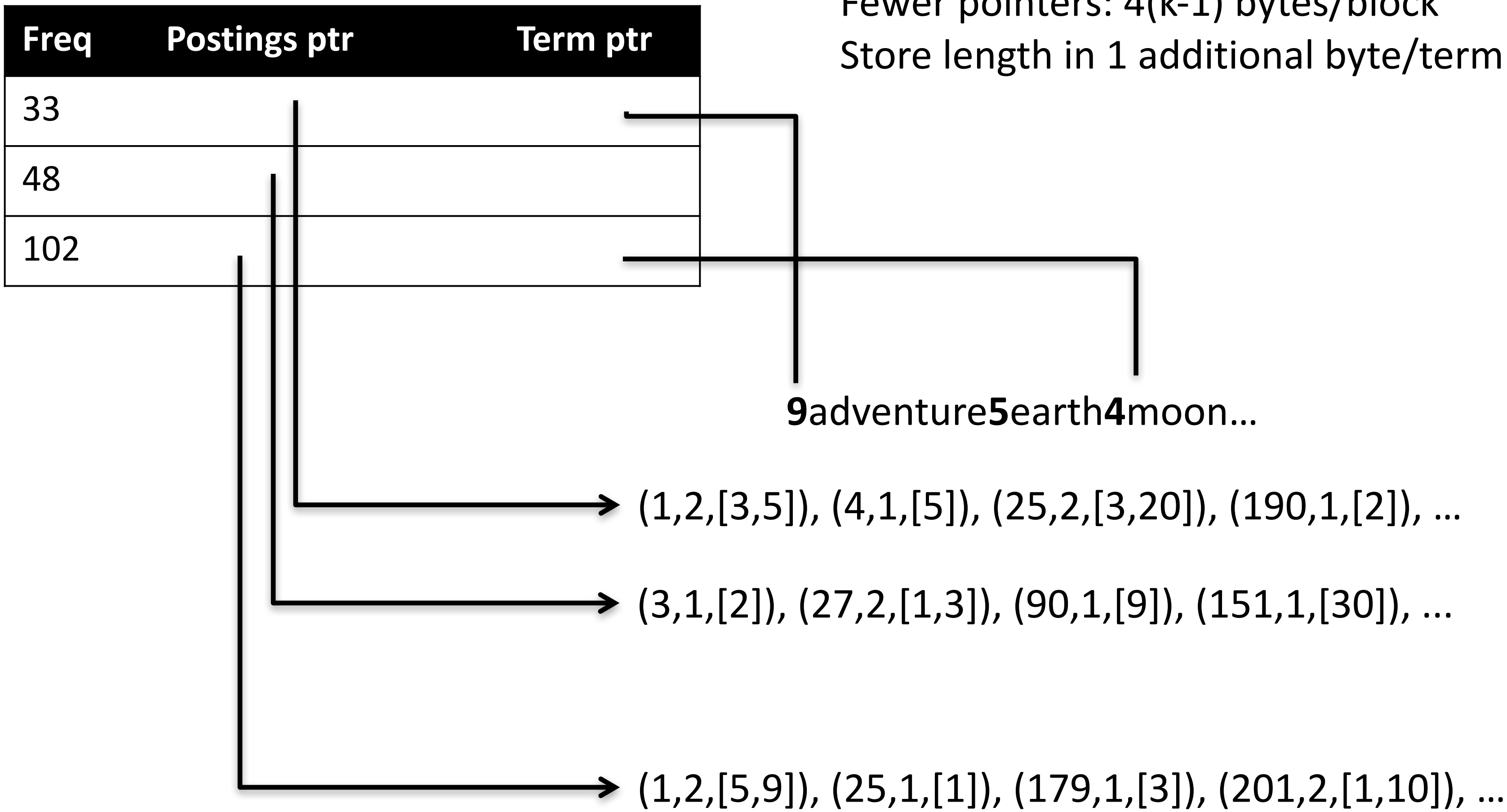
(1,2,[5,9]), (25,1,[1]), (179,1,[3]), (201,2,[1,10]), …

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		

9adventure**5**earth**4**moon…

(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

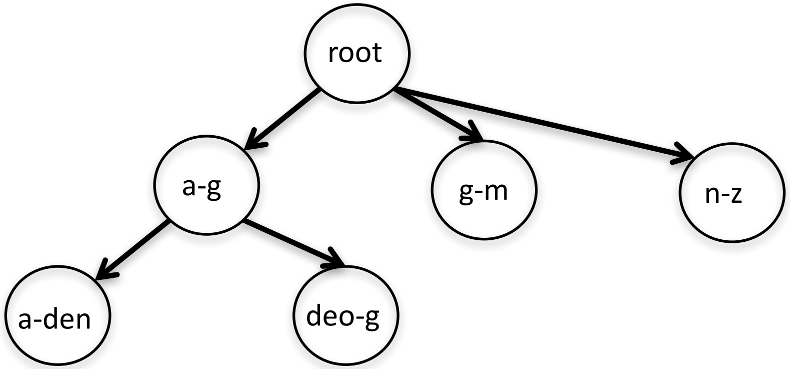
8automata8automate9automatic10automation

→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

- 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
```

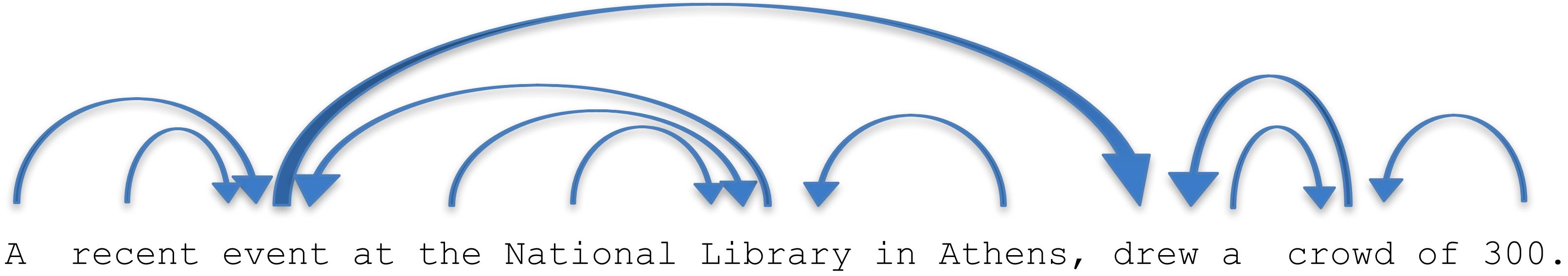
Payload in posting lists

- 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


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

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

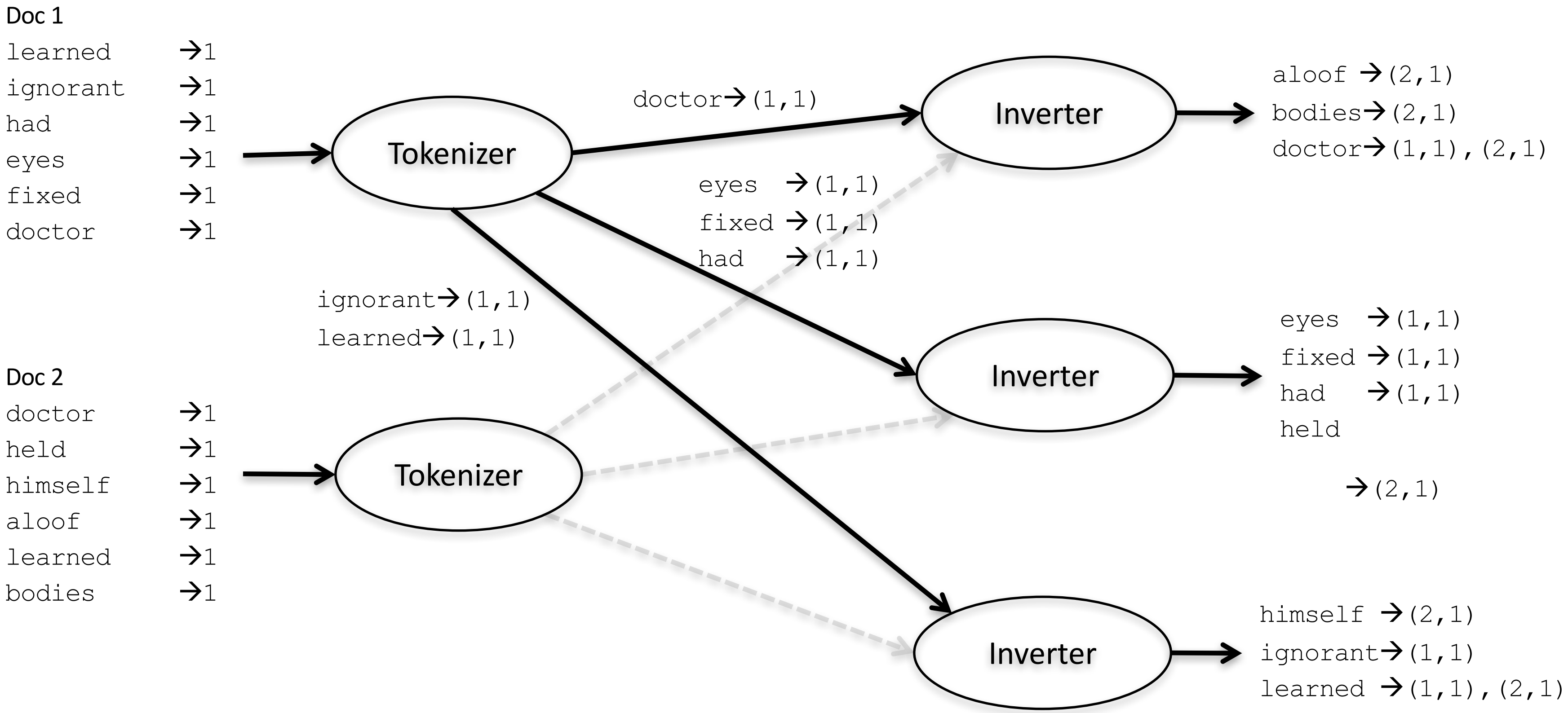
- Main idea
  - Build intermediate complete inverted indexes
  - At the end, merge the intermediate indexes
  - No need to keep information between intermediate indexes

```
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



Doc 1

learned →1
ignorant →1
had →1
eyes →1
fixed →1
doctor →1

Doc 2

doctor →1
held →1
himself →1
aloof →1
learned →1
bodies →1

Tokenizer

Tokenizer

doctor→(1,1)

eyes →(1,1)
fixed →(1,1)
had →(1,1)

ignorant→(1,1)
learned→(1,1)

Inverter

Inverter

Inverter

aloof →(2,1)
bodies→(2,1)
doctor→(1,1),(2,1)

eyes →(1,1)
fixed →(1,1)
had →(1,1)
held

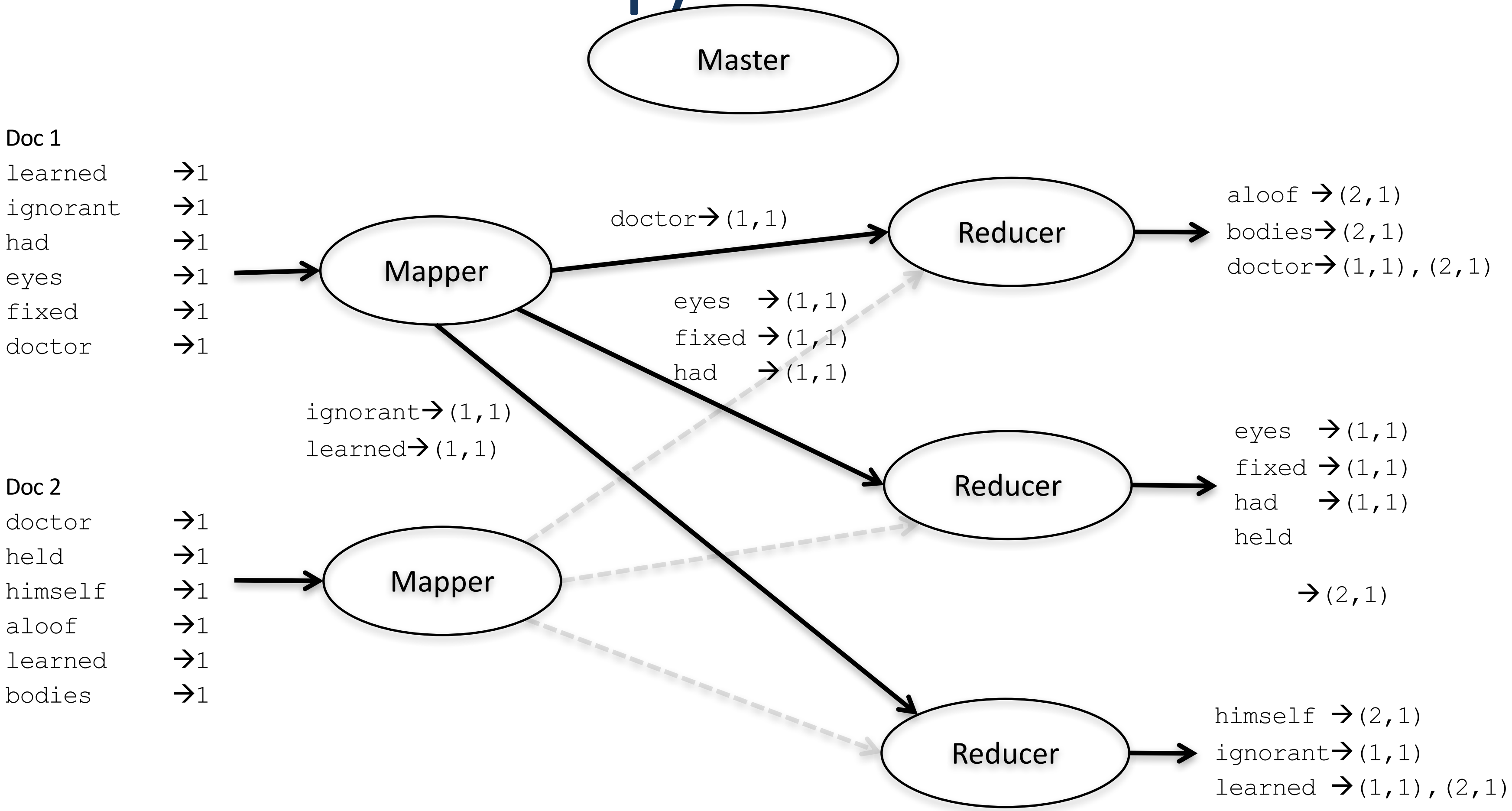
→(2,1)

himself →(2,1)
ignorant→(1,1)
learned →(1,1),(2,1)

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



Master

Doc 1

learned →1
ignorant →1
had →1
eyes →1
fixed →1
doctor →1

Doc 2

doctor →1
held →1
himself →1
aloof →1
learned →1
bodies →1

Mapper

Mapper

doctor→(1,1)

eyes →(1,1)
fixed →(1,1)
had →(1,1)

ignorant→(1,1)
learned→(1,1)

Reducer

aloof →(2,1)
bodies→(2,1)
doctor→(1,1),(2,1)

Reducer

eyes →(1,1)
fixed →(1,1)
had →(1,1)
held

→(2,1)

Reducer

himself →(2,1)
ignorant→(1,1)
learned →(1,1),(2,1)

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* ∩ *B* = 0
- 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 $\in \mathrm{N}^{|V|}$.

Bag of words model

- We do not consider the order of words and their distance in the document.
- *“Paris is the capital of France”* and *“France is the capital of Paris”* 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, $tf(t,d) = \dfrac{f(t,d)}{\max\{f(w,d) : wind\}}$
  - 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

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

- $tf_{t,d} \rightarrow w_{t,d}$: 0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, ...
- Score for a query document (q,d) pair:

$$tf\,(q, d) = \sum\nolimits_{t \in q \cap d} (1 + log\ tf_{t,d}\ )$$

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

Frequency in document vs. frequency in collection

- 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 Hypermnesia
- 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: $\mathrm{idf}_t = \log_{10} \frac{N}{\mathrm{df}_t}$

  (*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*

Examples for idf

- Compute idf_t using the formula:

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

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 ranking of documents for queries with at least two terms.
- For example, in the query "*hypermensia feature*",
  - idf weighting increases the relative weight of "*hypermensia*" 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.

$$w_{t,d} = (1 + log tf_{t,d}) * log \frac{N}{df_t}$$

- 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
. . .						

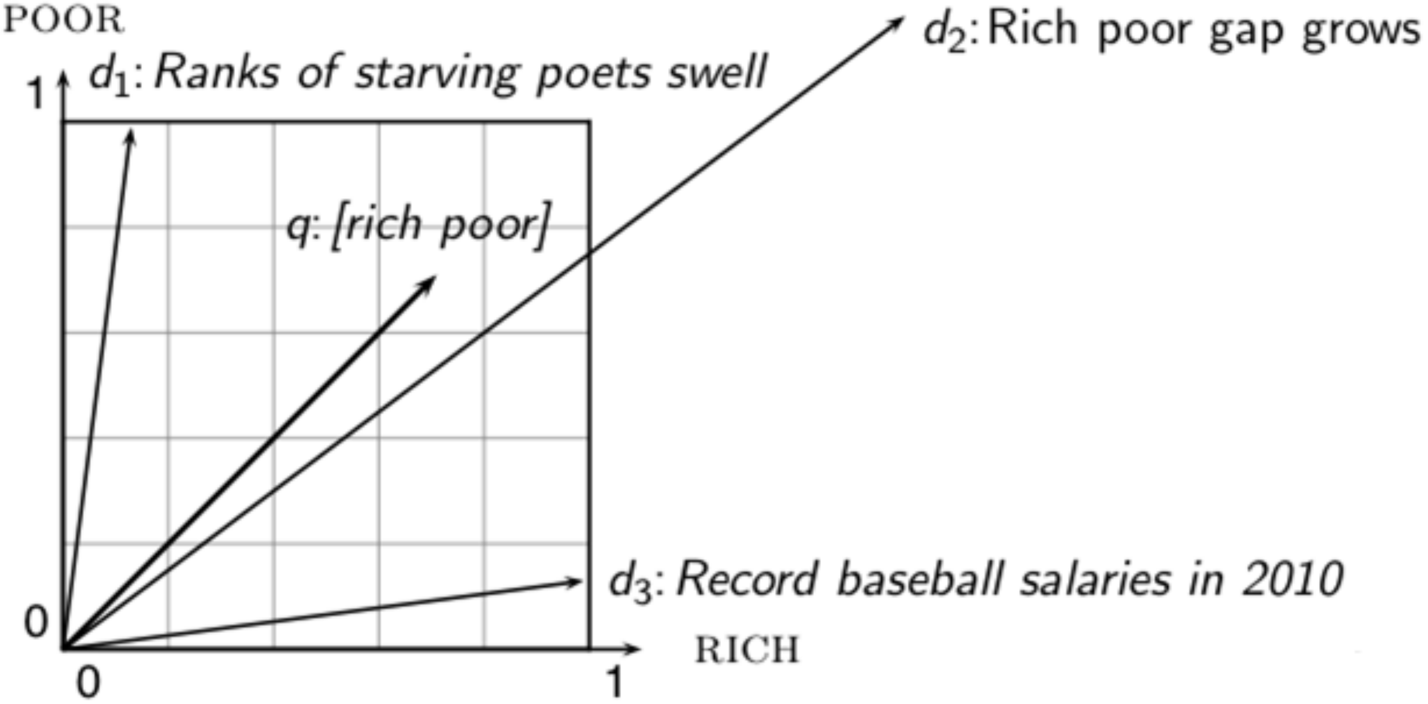
Each document is now represented as a count vector $\in N^{|V|}$.

Binary → count → 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 $\in R^{|V|}$.

Why vector distance may be a bad idea

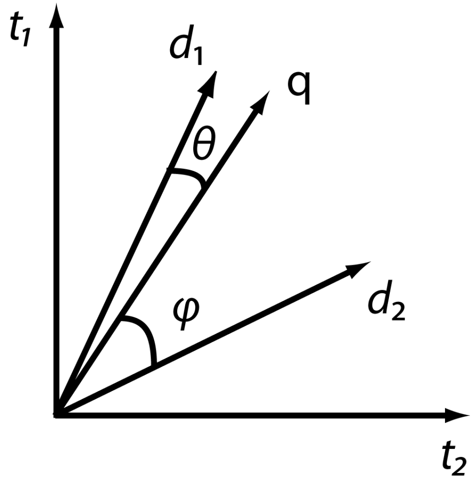

POOR
d_1: Ranks of starving poets swell
d_2: Rich poor gap grows
q: [rich poor]
d_3: Record baseball salaries in 2010
RICH
1
0
0
1

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 $d = (w_{1,d}, \ldots, w_{k,d})$ and $q = (w_{1,q}, \ldots, w_{k,q})$
  - 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

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



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

$$tf_{t,d} = f_{t,d} \qquad tf_{t,d} = \frac{f_{t,d}}{\max\left(f_{s,d}\right)}$$

  - $f_{t,d}$ 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* ”

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

  - *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	scienc e	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	scienc e	field	studies	decision	support	systems	enterprise 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

- query considered as a new document there fore represented as a vector.
- 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(φ)
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 tradeoff

- 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*: $\frac{1}{F} = \frac{1}{2}(\frac{1}{P} + \frac{1}{R})$

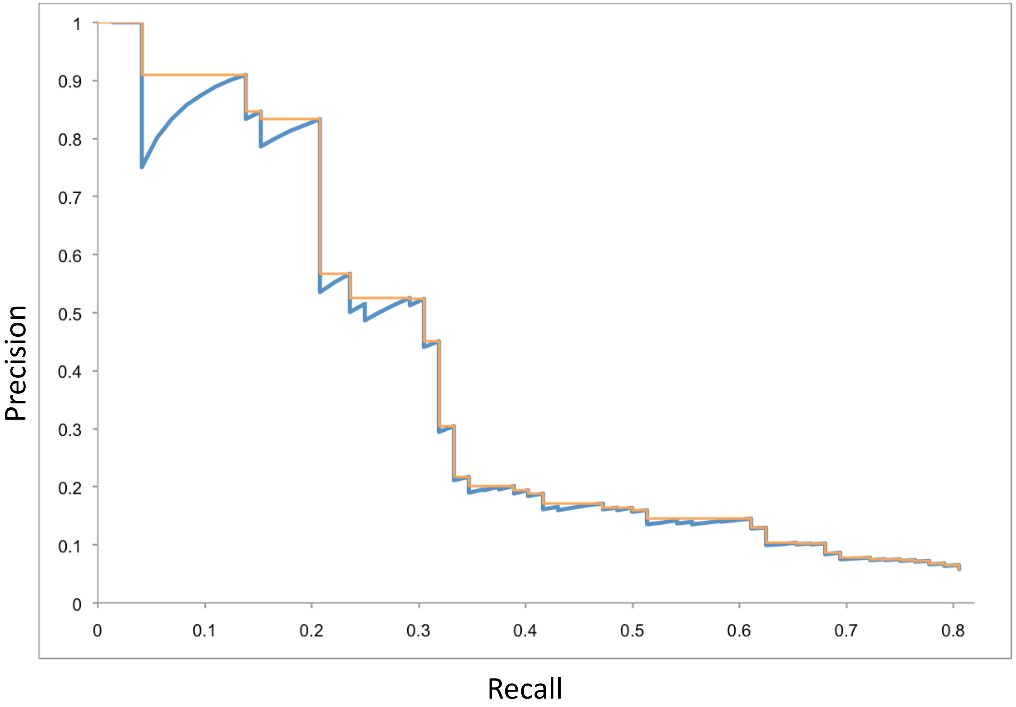
F: Example

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

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

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
…	…	…	…	…

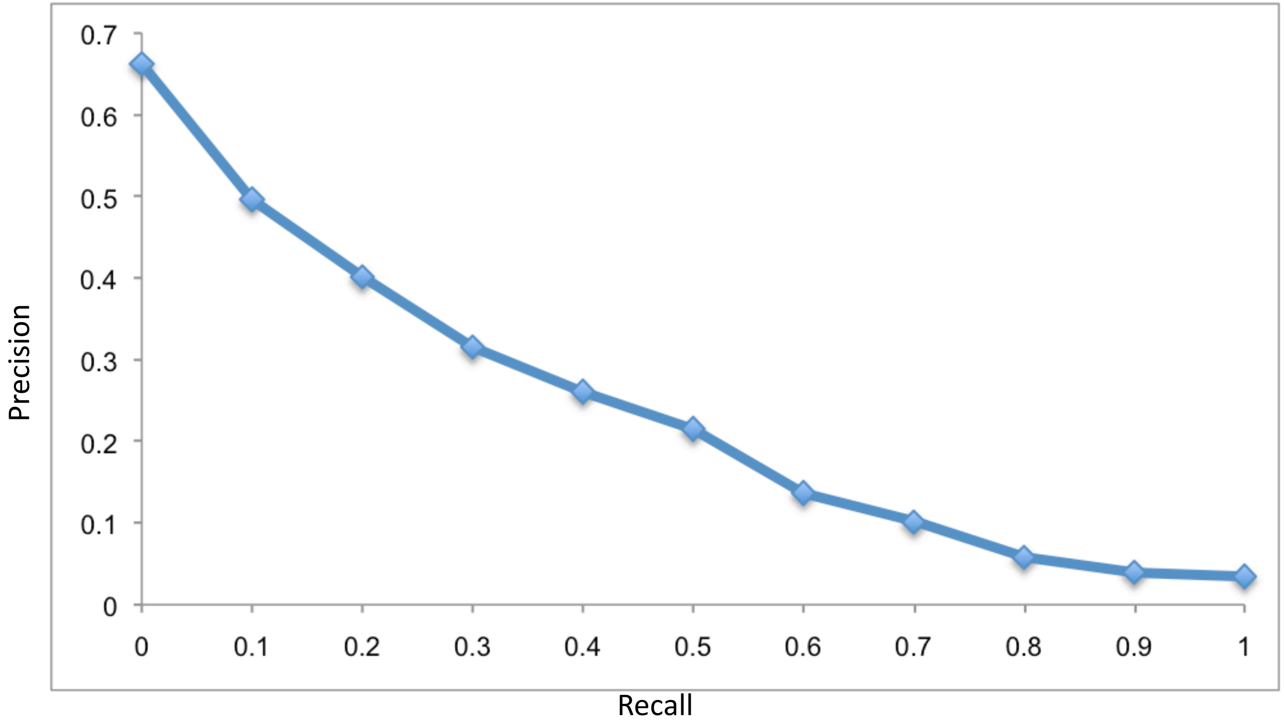


Interpolated precision at recall level *r*:

- 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
- 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

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

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

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

Graph of Word – a novel approach for text mining

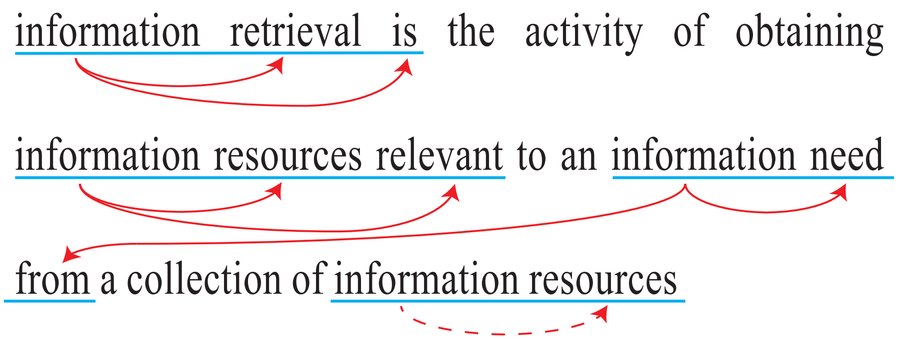
- For efficiency reasons, we used to make the term independence assumption through the bag-of-word representation and the term frequency weighting

- We challenge this assumption by taking into account word dependence and word order through a graph-based representation of a document at indexing time (graph-of-word)

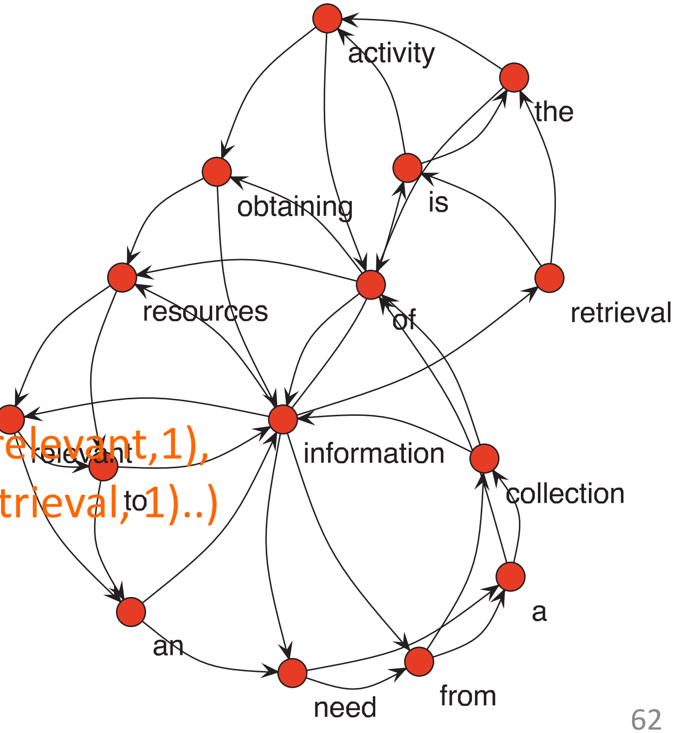
- Graphs have been successfully used in IR to encompass relations between entities and propose meaningful weights (e.g. PageRank[Page et al., 1999])

Graph-based Text Mining

- Instead of the traditional *bag-of-words* (i.e. multiset of terms), we represent a document as a *graph-of-words* to capture *word order* and *word dependency*.



information retrieval is the activity of obtaining information resources relevant to an information need from a collection of information resources



Bag of words: ((activity,1), (collection,1) (information,4), (relevant,1), (resources, 2), (retrieval, 1)..)

Graph of Word – a novel approach for text mining

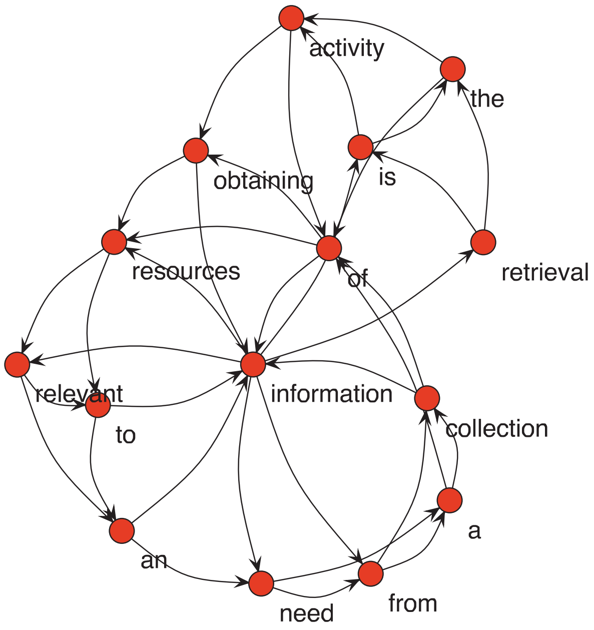
- For efficiency reasons, we used to make the term independence assumption through the bag-of-word representation and the term frequency weighting

- We challenge this assumption by taking into account word dependence and word order through a graph-based representation of a document at indexing time (graph-of-word)

- Graphs have been successfully used in IR to encompass relations between entities and propose meaningful weights (e.g. PageRank[Page et al., 1999])

Graph-based Ad Hoc IR I

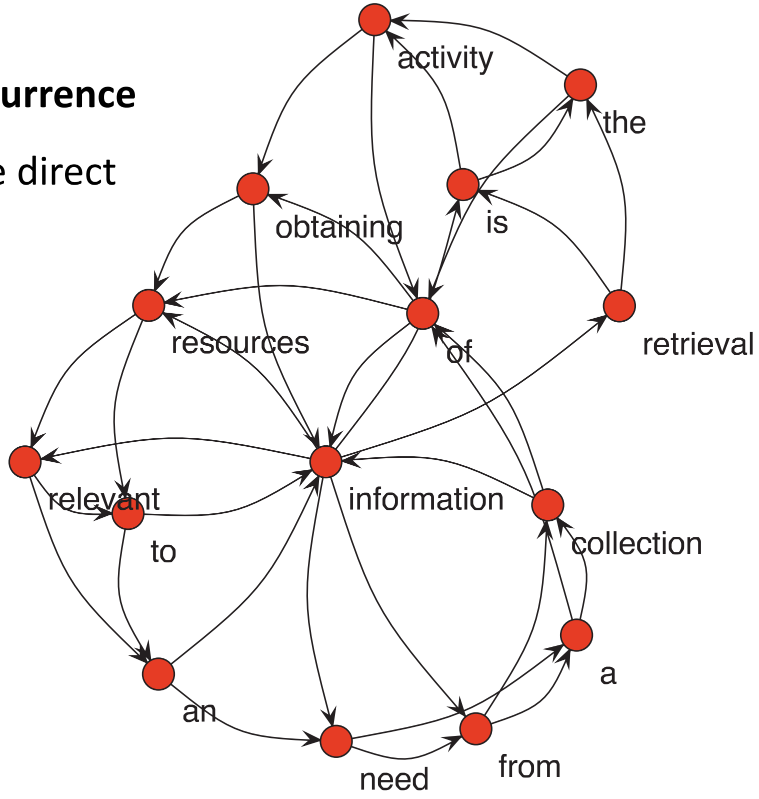
- Ad Hoc Information Retrieval [1,2]
  - ➢ Unweighted directed graph
  - ➢ The weight of a word in the document: number of neighbors in the graph => favor words that occur with many different other words
  - ➢ Robust to varying document length: weight of a word increases only with **new context** of co-occurrences as opposed to the word frequency that increases with any new co-occurrence.



Indegree-BASED TW

- The weight of a term in a document is its **indegree** (numbers of incoming edges) in the **graph-of-word**
- It represents the **number of distinct contexts of occurrence**
- We store the document as a vector of weights in the direct index and similarly in the inverted index
- For example:

information	5
retrieval	1
is	2
the	2
activity	2
of	3
obtaining	2
resources	3
relevant	2
to	2
an	2
need	2
from	2
a	2
collection	2



TF-IDF and BM25

- Term t, document d, collection of size N, term frequency *tf(t, d)*, document frequency *df(t)*, document length |*d*|, average document length *avdl*, asymptotical marginal gain k_1 (1.2), slope parameter *b*

- ## TF-IDF [Singhal et al., TREC-7]

TF-IDF(t, d) = TF$_{pol}$-IDF(t, d) = TF$_p$oTF$_l$(t, d) x IDF(t) =

$$\left(\frac{1+\log\left(1+\log\left(tf(t,d)\right)\right)}{1-b+b\times\frac{|d|}{avdl}}\right)\times\log\left(\frac{N+1}{df(t)}\right)$$

- ## BM25 [Lv and Zhai, CIKM ’11]

BM25(t, d) =

$$\left(\frac{(k_1+1)\times tf(t,d)}{k_1\times\left(1-b+b\times\frac{|d|}{avdl}\right)+tf(t,d)}\right)\times\log\left(\frac{N+1}{df(t)}\right)$$

TW-IDF

- Term t, document *d*, collection of size *N*, term weight *tw(t, d)*, document frequency *df(t)*, document length *|d|*, average document length *avdl*, asymptotical marginal gain k_1 (1.2), slope parameter *b*
- TW-IDF(t, d) = $$\left(\frac{tw(t,d)}{1-b+b\times\frac{|d|}{avdl}}\right)\times\log\left(\frac{N+1}{df(t)}\right)$$
- In the bag-of-word representation, *tw* is usually defined as the term frequency or sometimes just the presence/absence of a term (binary tf)
- In our graph-of-word representation, *tw* is the indegree of the vertex representing the term in the graph

EXPERIMENTS

- DATASETS
- PLATFORM
- EVALUATION
- RESULTS

Datasets

- Disks 1 & 2 (TREC)

  741,856 news articles from Wall Street Journal (1987-1992), Federal Register (1988-1989), Associated Press (1988-1989 and Information from the Computer Select disks (1989-1990)

- Disks 4 & 5 (TREC, minus the Congressional Record)

  528,155 news releases from Federal Register (1994), Financial Times (1991-1994), Foreign Broadcast Information Service (1996) and Los Angeles Times (1989-1990)

- WT10G (TREC)

  1,692,096 crawled pages from a snapshot of the Web in 1997

- .GOV2 (TREC)

  25,205,179 crawled Web pages from .gov sites in early 2004

Datasets (cont.)

Dataset Statistic	Disks 1 & 2	Disks 4 & 5	WT10G	.GOV2
# of documents	741,856	528,155	1,692,096	25,205,179
# of unique terms	535,001	520,423	3,135,780	15,324,292
average # of terms (*avdl*)	237	272	398	645
average # of vertices	125	157	165	185
average # of edges	608	734	901	1,185

Table 1: Statistics on the four TREC datasets used; Disks 4&5 excludes the Congressional Record. The average values are computed per document.

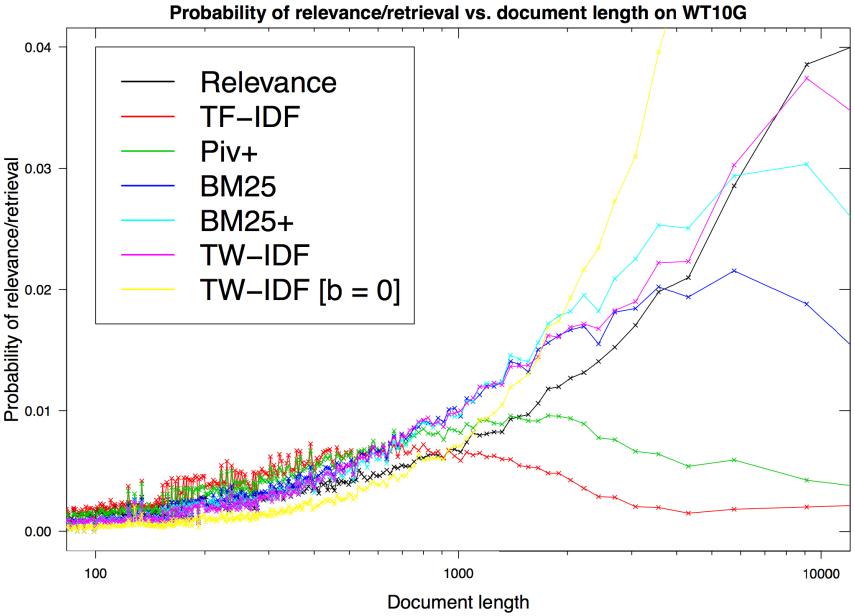
EVALUATION

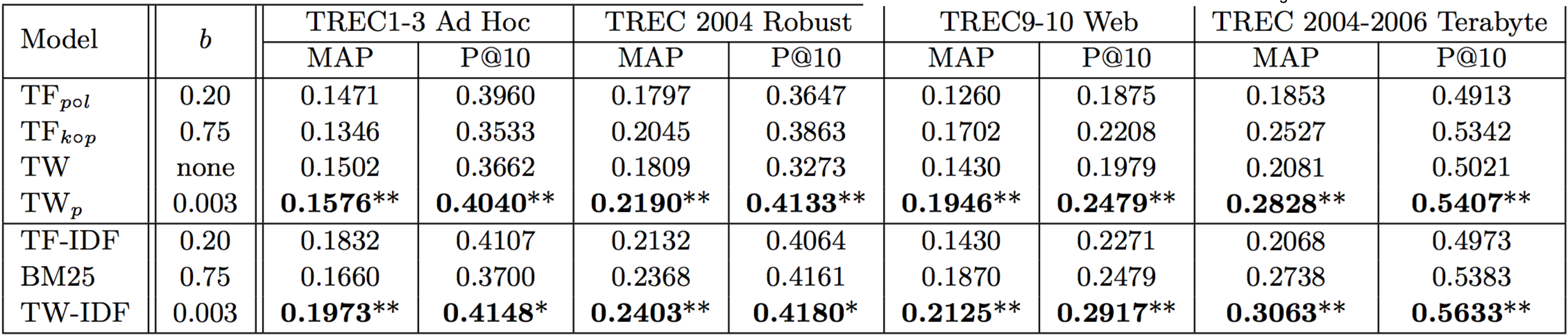
- Mean Average Precision (MAP) and Precision at 10 (P@10)
  considering only the top-ranked 1000 documents for each run
- Statistical significance of improvement was assessed using the Student's paired t-test
  - R implementation (t.test {stats} package), trec_eval output as input
  - Two-sided p-values less than 0.05 and 0.01 to reject the null hypothesis
- Likelihood of relevance vs. likelihood of retrieval [Singhal et al., SIGIR '96]

- 4 baseline models: TF-IDF, BM25, Piv+ and BM25+
  - Tuned slope parameter *b* for pivoted document length normalization (2-fold cross-validation, odd vs. even topic ids, MAP maximization)
  - Default (1.0) lower-bounding gap ∂ [Lv and Zhai, CIKM '11]

Graph-based Ad Hoc IR II

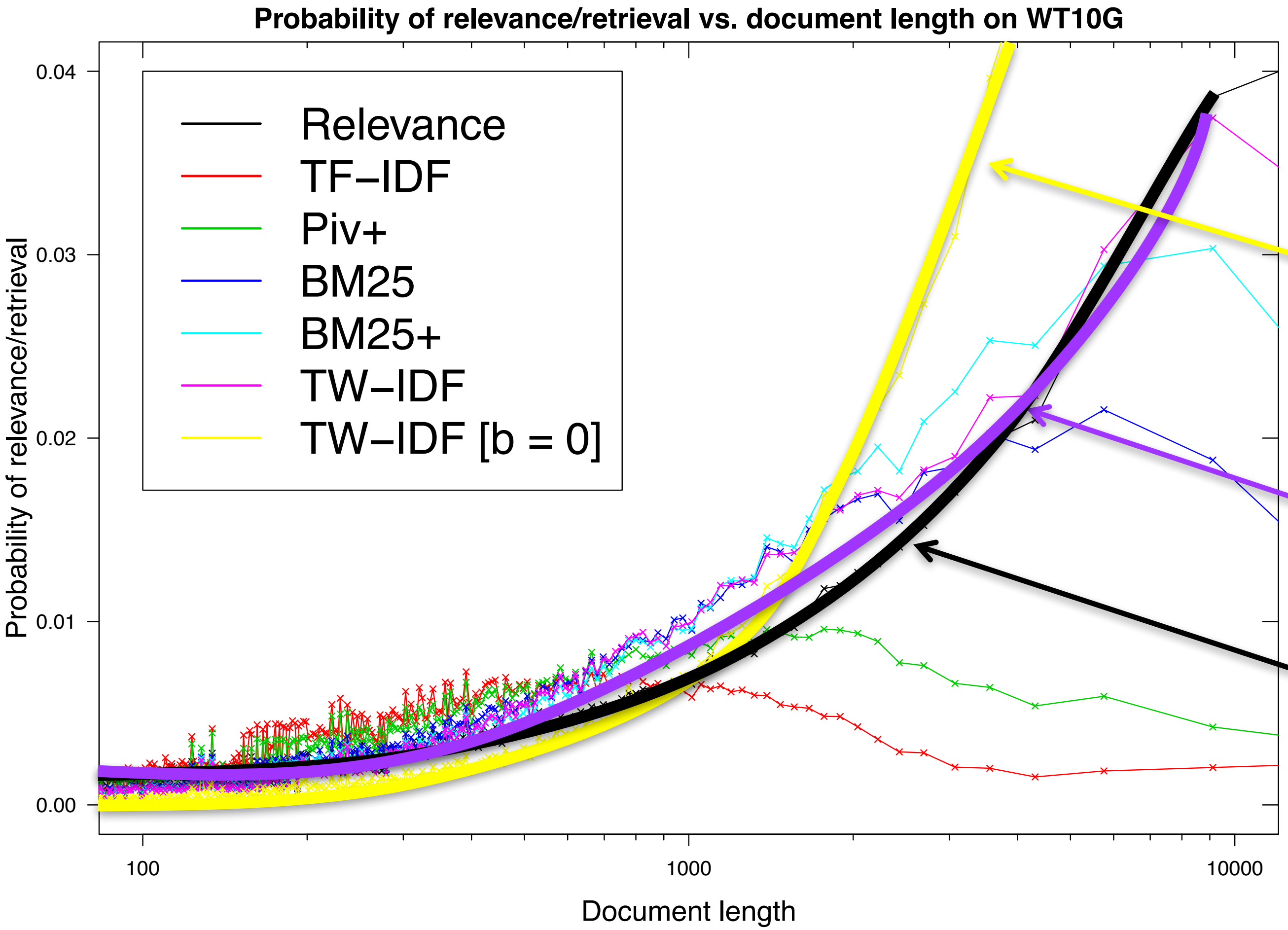
- Evaluation in terms of:
  - ➢Mean Average Precision
  - ➢Precision@10
  - ➢Probability of relevance vs. probability of retrieval





Model	b	TREC1-3 Ad Hoc		TREC 2004 Robust		TREC9-10 Web		TREC 2004-2006 Terabyte	
		MAP	P@10	MAP	P@10	MAP	P@10	MAP	P@10
TF_{pol}	0.20	0.1471	0.3960	0.1797	0.3647	0.1260	0.1875	0.1853	0.4913
TF_{kop}	0.75	0.1346	0.3533	0.2045	0.3863	0.1702	0.2208	0.2527	0.5342
TW	none	0.1502	0.3662	0.1809	0.3273	0.1430	0.1979	0.2081	0.5021
TW_p	0.003	**0.1576****	**0.4040****	**0.2190****	**0.4133****	**0.1946****	**0.2479****	**0.2828****	**0.5407****
TF-IDF	0.20	0.1832	0.4107	0.2132	0.4064	0.1430	0.2271	0.2068	0.4973
BM25	0.75	0.1660	0.3700	0.2368	0.4161	0.1870	0.2479	0.2738	0.5383
TW-IDF	0.003	**0.1973****	**0.4148***	**0.2403****	**0.4180***	**0.2125****	**0.2917****	**0.3063****	**0.5633****

Likelihood of relevance vs. likelihood of retrieval



Probability of retrieval (TW-IDF, b = 0)

Probability of retrieval (TW-IDF)

Probability of relevance (ground truth)

GoW for keyword extraction for documents

Keywords are used everywhere:

- looking up information on the Web (e. g., via a search engine bar)
- finding similar posts on a blog (e. g., tag cloud)
- for ads matching (e. g., AdWords' keyword planner)
- for research paper indexing and retrieval (e. g., SpringerLink)
- for research paper reviewer assignment (e. g., ECIR '15!)

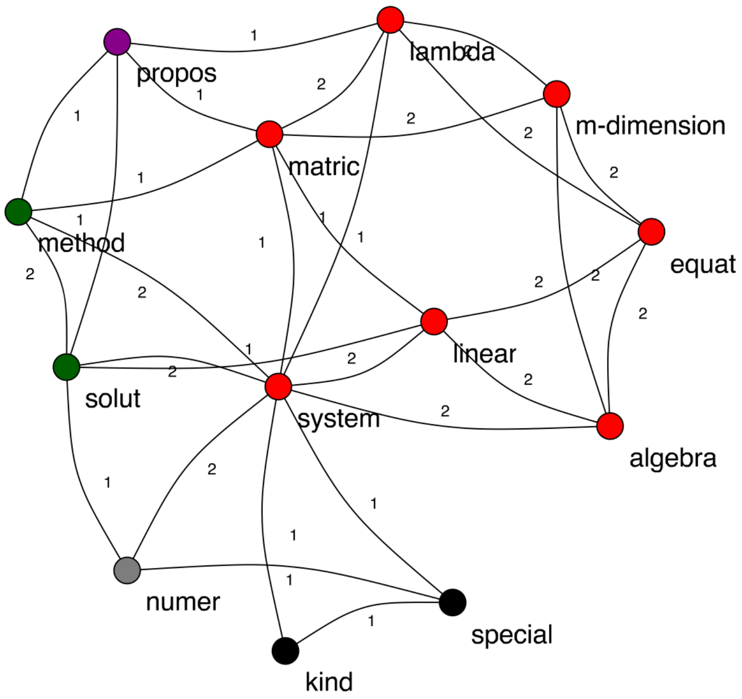
Applications are numerous:

- **summarization** (to get a gist of the content of a document)
- **information filtering** (to select specific documents of interest)
- **indexing** (to answer keyword-based queries)
- **query expansion** (using additional keywords from top results)

Graph-based Keyword Extraction

- Single-Document Keyword Extraction [3]
  - Elect the most cohesive sets of words in the graph as keywords
  - Use k-core decomposition to extract the main core of the graph
  - Weighted edges as opposed to Ad Hoc IR (single-document => no normalization)

A method for solution of systems of linear algebraic equations with m-dimensional lambda matrices.
A system of linear algebraic equations with m-dimensional lambda matrices is considered. The proposed method of searching for the solution of this system lies in reducing it to a numerical system of a special kind.



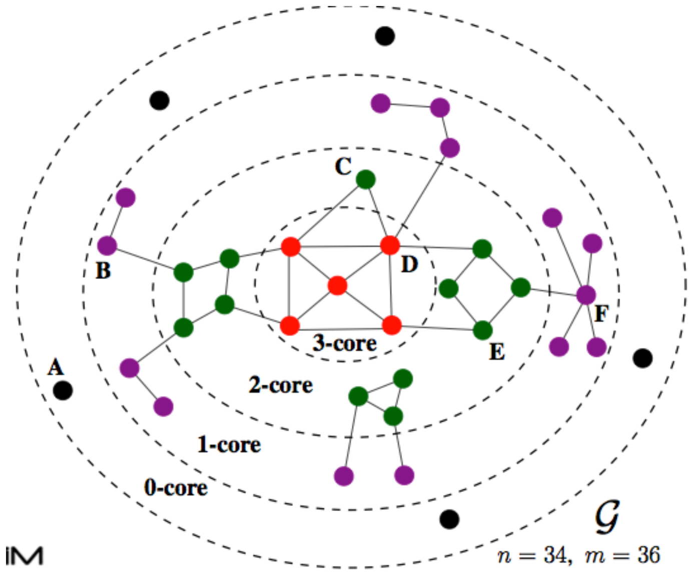
Keywords manually assigned by human annotators
linear algebra equat; numer system; m-dimension lambda matric

Graph-based keyword extraction

Two existing graph-based keyword extractors: PageRank [Mihalcea and Tarau, 2004] HITS [Litvak and Last, 2008]

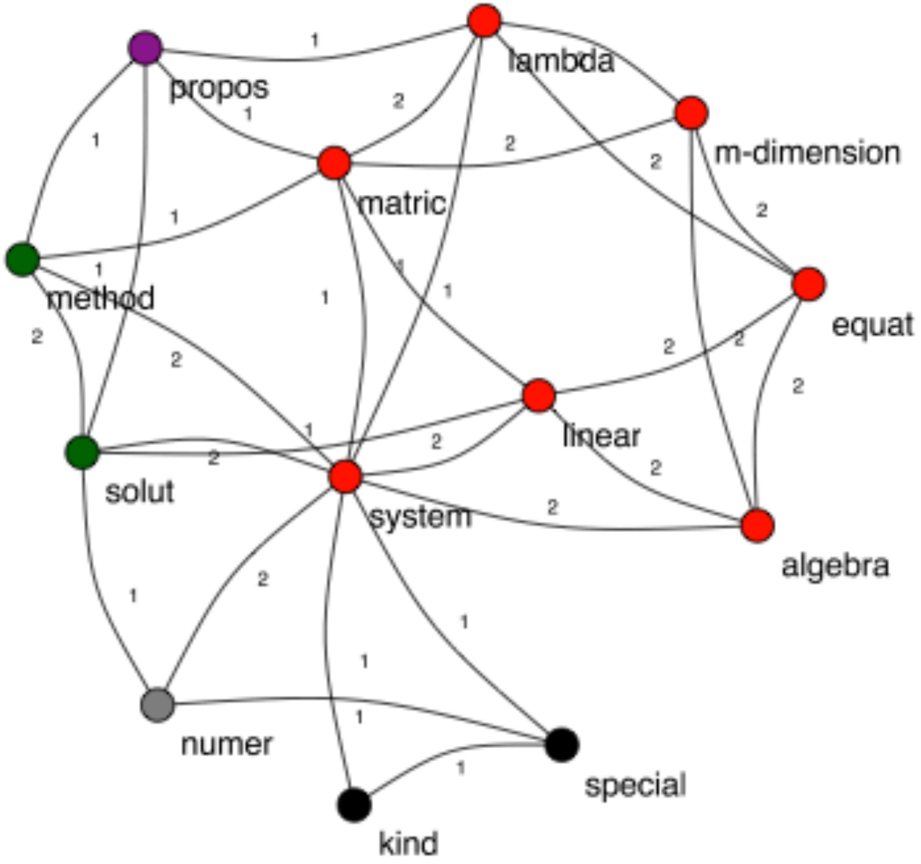
Two iterative graph algorithms that assign an influence score to a node based on its centrality (top ones supposedly the most representative).

⇒ we propose to retain the main core of the graph to extract the nodes based on their centrality and cohesiveness.



K-core decomposition of the graph

Pagerank vs. main core

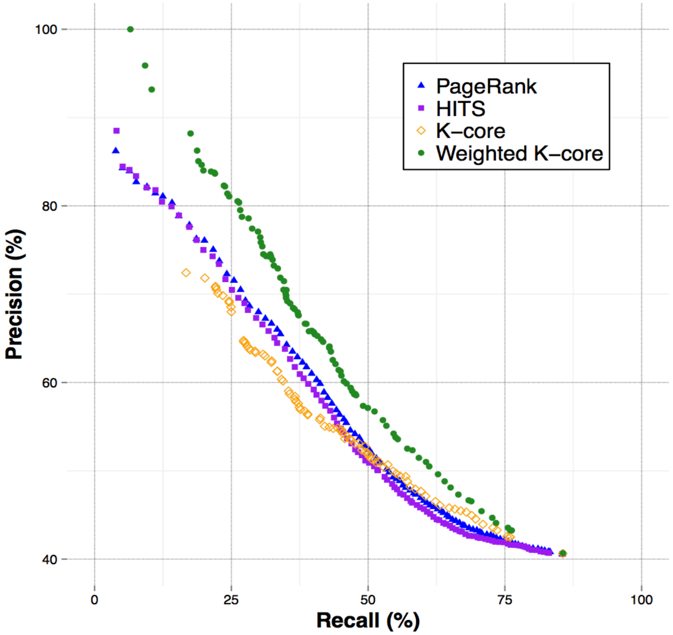


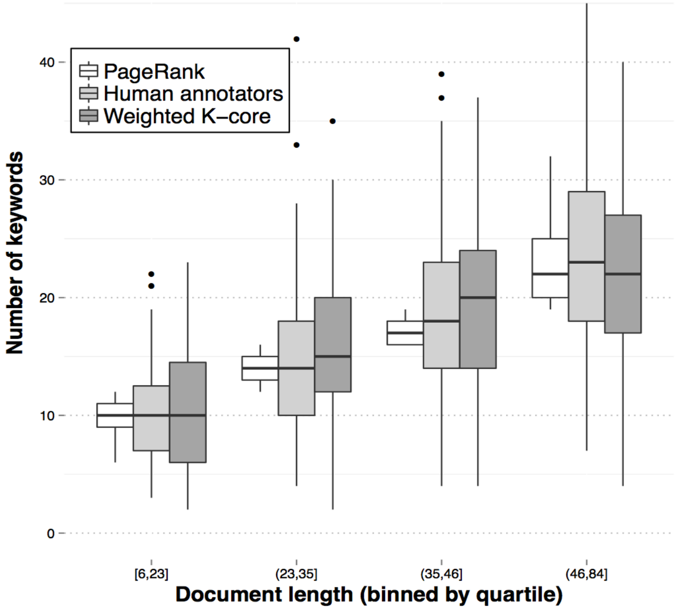
Keywords manually assigned by human annotators
linear algebra equat; numer system; m-dimension lambda matric

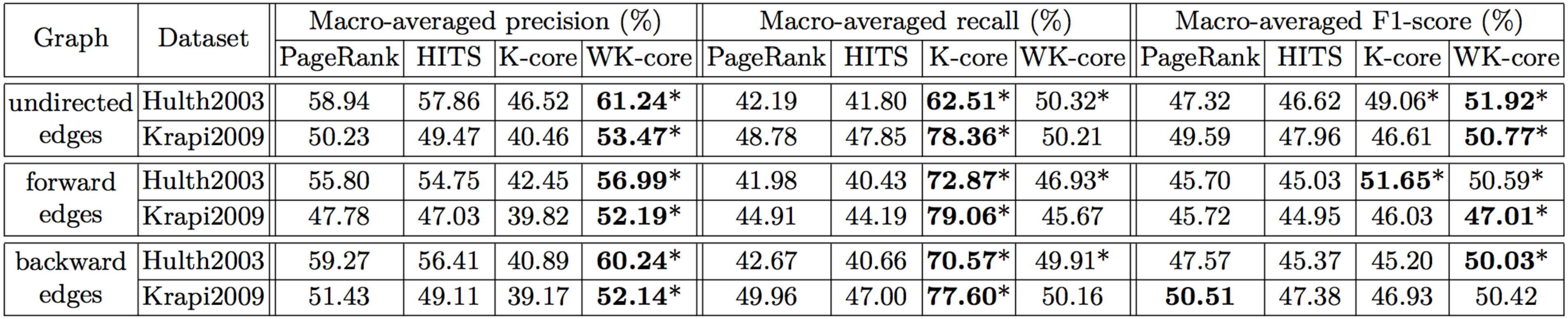
WK-core		PageRank	
system	6	**system**	1.93
matric	6	**matric**	1.27
lambda	6	solut	1.10
linear	6	**lambda**	1.08
equat	6	**linear**	1.08
algebra	6	**equat**	0.90
m-dim...	6	**algebra**	0.90
method	5	**m-dim...**	0.90
solut	5	propos	0.89
propos	4	method	0.88
numer	3	special	0.78
specia	2	**numer**	0.74
kind	2	kind	0.55

Graph-based Keyword Extraction II

- Evaluation in terms of:
  - Precision
  - Recall
  - F1-score
  - Precision/recall
  - # of keywords



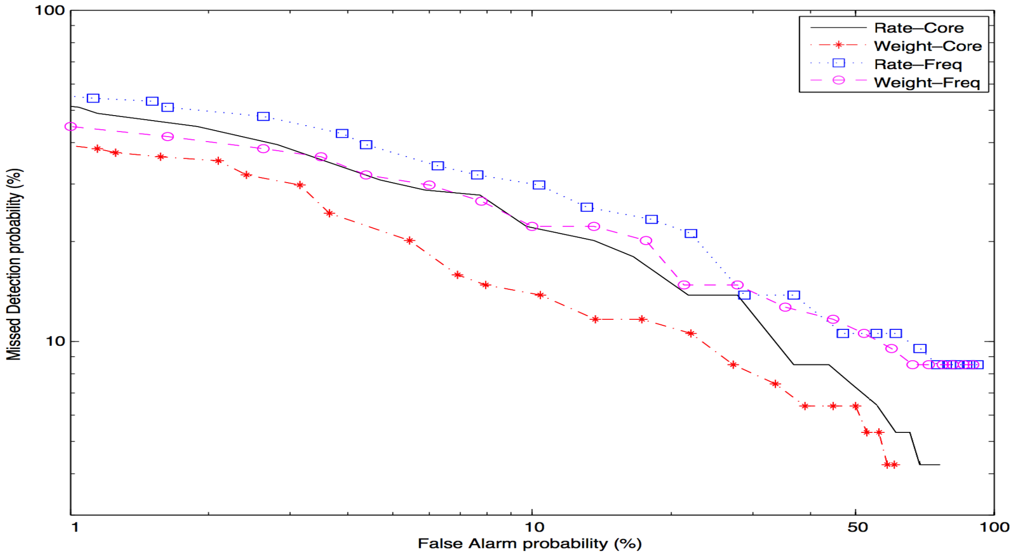


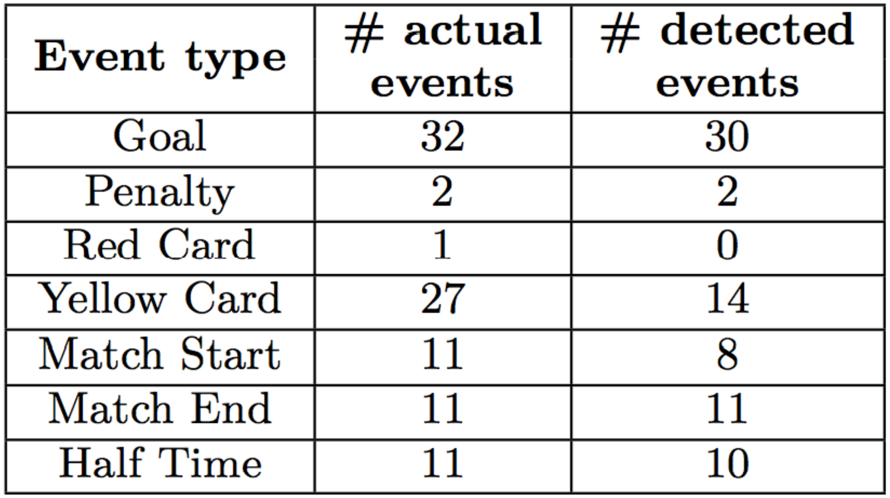


Graph	Dataset	Macro-averaged precision (%)				Macro-averaged recall (%)				Macro-averaged F1-score (%)			
		PageRank	HITS	K-core	WK-core	PageRank	HITS	K-core	WK-core	PageRank	HITS	K-core	WK-core
undirected edges	Hulth2003	58.94	57.86	46.52	**61.24***	42.19	41.80	**62.51***	50.32*	47.32	46.62	49.06*	**51.92***
	Krapi2009	50.23	49.47	40.46	**53.47***	48.78	47.85	**78.36***	50.21	49.59	47.96	46.61	**50.77***
forward edges	Hulth2003	55.80	54.75	42.45	**56.99***	41.98	40.43	**72.87***	46.93*	45.70	45.03	**51.65***	50.59*
	Krapi2009	47.78	47.03	39.82	**52.19***	44.91	44.19	**79.06***	45.67	45.72	44.95	46.03	**47.01***
backward edges	Hulth2003	59.27	56.41	40.89	**60.24***	42.67	40.66	**70.57***	49.91*	47.57	45.37	45.20	**50.03***
	Krapi2009	51.43	49.11	39.17	**52.14***	49.96	47.00	**77.60***	50.16	**50.51**	47.38	46.93	50.42

Graph-based Sub-event Detection

- Sub-event Detection in Textual Streams
  - Identify newly-formed tight communities of words as novel sub-event
- Evaluation in terms of:
  - Precision
  - Recall
  - DET curve





Event type	# actual events	# detected events
Goal	32	30
Penalty	2	2
Red Card	1	0
Yellow Card	27	14
Match Start	11	8
Match End	11	11
Half Time	11	10

Current Work

- Text classification with GoW – graph kernels
- Sentiment analysis
- Text stream summarization

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).