Information Retrieval

CS 547/DS 547

Worcester Polytechnic Institute

Department of Computer Science

Instructor: Prof. Kyumin Lee

Quiz

- There will be a quiz next Wednesday in class.
  - Yes/No questions, multiple choice questions or short answer questions.

Project Team

- Bo Yu, Jin Yang, Kangjian Wu
- Vignesh Sundaram, Amey More, Akanksha Pawar, Padmesh Naik
- Xiaofan Zhou, Yiming Liu, Yuyuan Liu
- Sidhant jain, Parth dhruv, Darshan Swami, Aditya Ramesh
- Chandra Rachabathuni, Ayush Shinde, Chao Wang, Anthony Chen
- Joe Scheufele, Alex Alvarez and Matt Suyer, Jason Dykstra
- Adityavikram Gurao, Snehith Varma Datla, Sree Likhith Dasari
- Samar, Bao, Michael, Megan
- Adhiraj, Supreeth, Akhil Daphara

So far, 31 students formed teams.

Previous Class…

Preprocessing Documents ➔Tokenization, Normalization, Stemming, Stop words

Initial stages of text processing

- Tokenization
  - Cut character sequence into word tokens
    - Deal with ***"John's"***, ***a state-of-the-art solution***
- Normalization
  - Map text and query term to same form
    - You want ***U.S.A.*** and ***USA*** to match
- Stemming
  - We may wish different forms of a root to match
    - ***authorize***, ***authorization***
- Stop words
  - We may omit very common words (or not)
    - ***the, a, to, of***

Previous Class…

Preprocessing Documents ➔Tokenization, Normalization, Stemming, Stop words

Skip pointers & Positional index

Previous Class…

Wild-card queries
➔ Permuterm Index

Permuterm index

- For term ***hello***, index under:
  - ***hello$, ello$h, llo$he, lo$hel, o$hell, $hello***

  where $ is a special symbol.

Query = ***hel*o***

X=*hel*, Y=*o*

Lookup ***o$hel****

Vector Space Retrieval

Take-away today

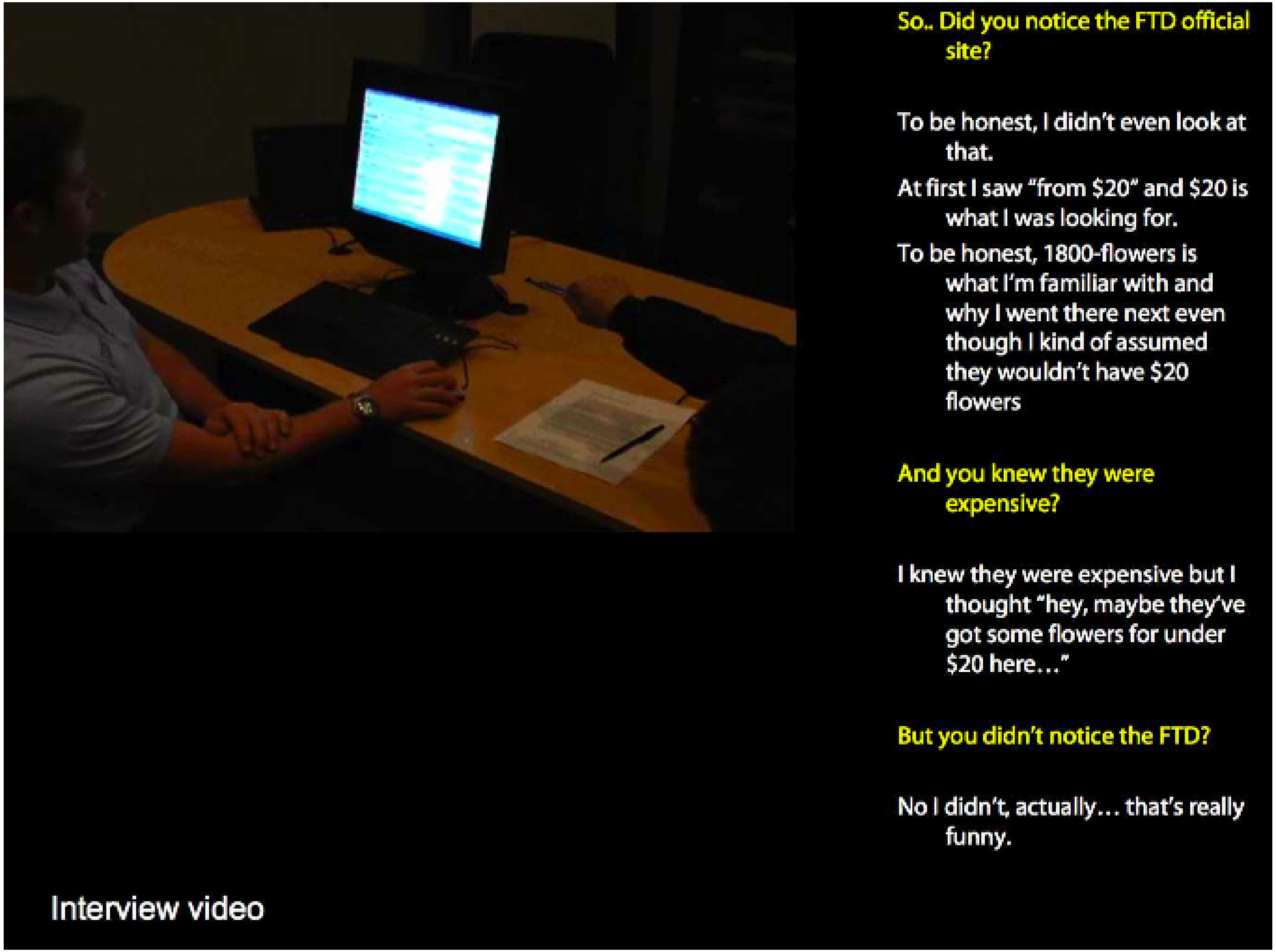
- Ranking search results: why it is important (as opposed to just presenting a set of unordered Boolean results)
- Term frequency: This is a key ingredient for ranking.
- Tf-idf ranking: best known traditional ranking scheme
- Vector space model: One of the most important formal models for information retrieval (along with Boolean and probabilistic models)

Ranked retrieval

- Thus far, our queries have all been Boolean.
  - Documents either match or don’t.
- Good for expert users with precise understanding of their needs and of the collection.
- Also good for applications: Applications can easily consume 1000s of results.
- Not good for the majority of users
- Most users are not capable of writing Boolean queries . . .
  - . . . or they are, but they think it’s too much work.
- Most users don’t want to wade through 1000s of results.
- This is particularly true of web search.

Empirical investigation of the effect of ranking

- How can we measure how important ranking is?
- Observe what searchers do when they are searching in a controlled setting
    - Videotape them
    - Ask them to “think aloud”
    - Interview them
    - Eye-track them
    - Time them
    - Record and count their clicks
- The following slides are from Dan Russell’s JCDL talk
- Dan Russell is a senior research scientist for “Search Quality & User Happiness” at Google.



So.. Did you notice the FTD official site?

To be honest, I didn't even look at that.

At first I saw "from $20" and $20 is what I was looking for.

To be honest, 1800-flowers is what I'm familiar with and why I went there next even though I kind of assumed they wouldn't have $20 flowers

And you knew they were expensive?

I knew they were expensive but I thought "hey, maybe they've got some flowers for under $20 here…"

But you didn't notice the FTD?

No I didn't, actually… that's really funny.

Interview video

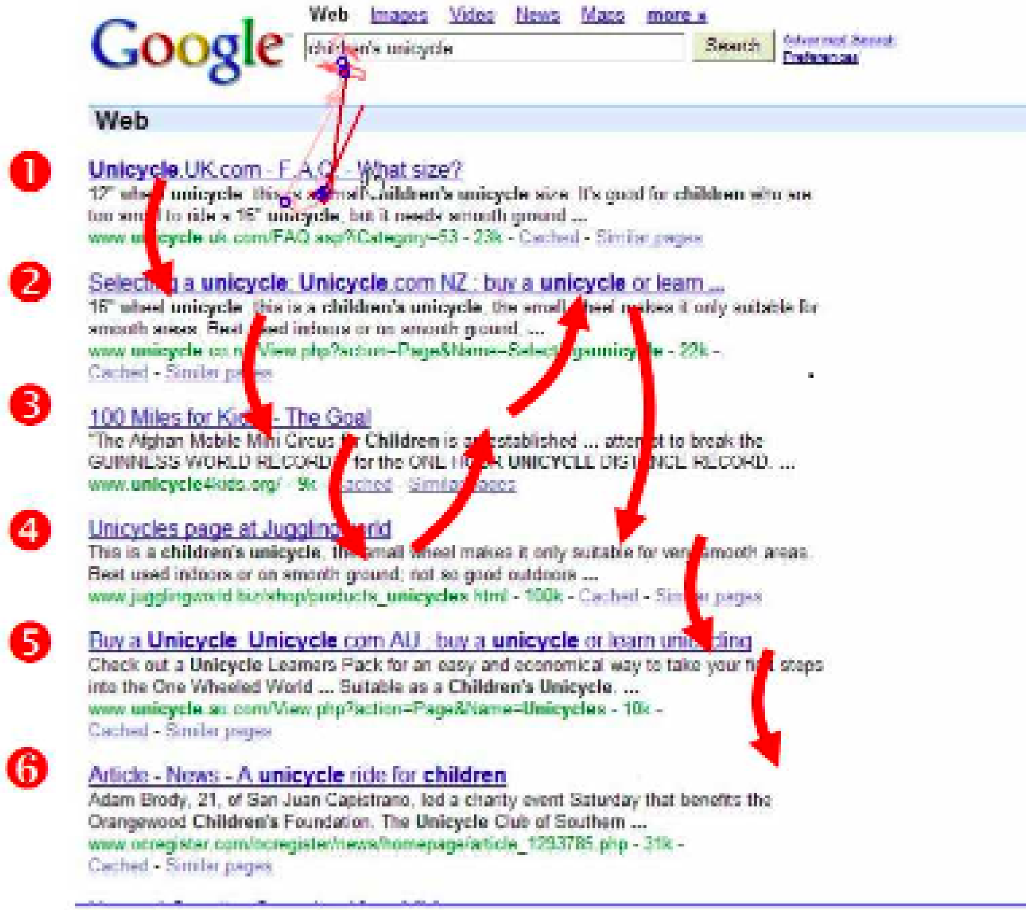
Rapidly scanning the results

Note scan pattern:

Page 3:
Result 1
Result 2
Result 3
Result 4
Result 3
Result 2
Result 4
Result 5
Result 6 <click>

Q: Why do this?

A: What's learned later
influences judgment
of earlier content.



Unicycle.UK.com - F.A.Q. - What size?
12" wheel unicycle: this is a small children's unicycle size. It's good for children who are too small to ride a 16" unicycle, but it needs smooth ground ...
www.unicycle.uk.com/FAQ.asp?Category=53 - 23k - Cached - Similar pages

Selecting a unicycle: Unicycle.com NZ : buy a unicycle or learn ...
16" wheel unicycle: this is a children's unicycle, the small wheel makes it only suitable for smooth areas. Best used indoors or on smooth ground. ...
www.unicycle.co.nz/View.php?action=Page&Name=Selectingunicycle - 22k - Cached - Similar pages

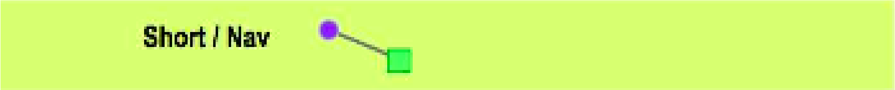
100 Miles for Kids - The Goal
"The Afghan Mobile Mini Circus for Children is an established ... attempt to break the GUINNESS WORLD RECORD for the ONE HOUR UNICYCLE DISTANCE RECORD. ...
www.unicycle4kids.org/ - 9k - Cached - Similar pages

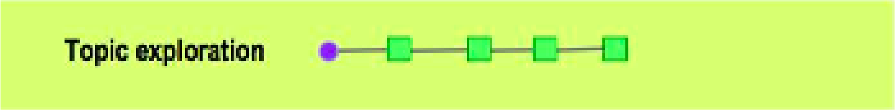
Unicycles page at Jugglingworld
This is a children's unicycle, the small wheel makes it only suitable for very smooth areas. Best used indoors or on smooth ground; not so good outdoors ...
www.jugglingworld.biz/shop/products_unicycles.html - 100k - Cached - Similar pages

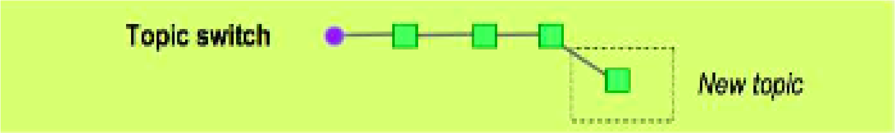
Buy a Unicycle. Unicycle.com AU : buy a unicycle or learn unicycling
Check out a Unicycle Learners Pack for an easy and economical way to take your first steps into the One Wheeled World ... Suitable as a Children's Unicycle. ...
www.unicycle.au.com/View.php?action=Page&Name=Unicycles - 10k - Cached - Similar pages

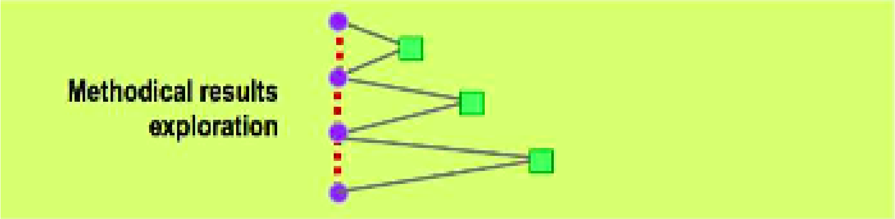
Article - News - A unicycle ride for children
Adam Brody, 21, of San Juan Capistrano, led a charity event Saturday that benefits the Orangewood Children's Foundation. The Unicycle Club of Southern ...
www.ocregister.com/ocregister/news/homepage/article_1293785.php - 31k - Cached - Similar pages

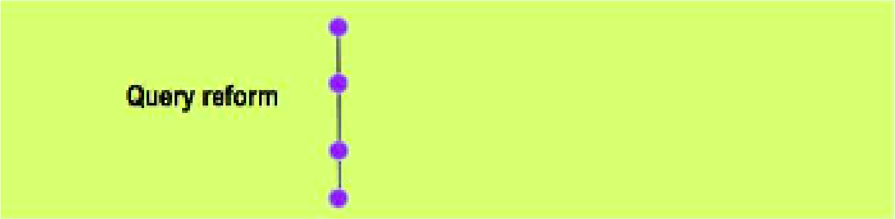
Kinds of behaviors we see in the data

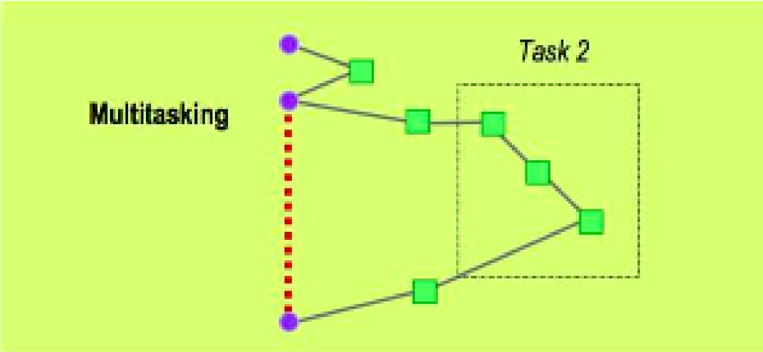


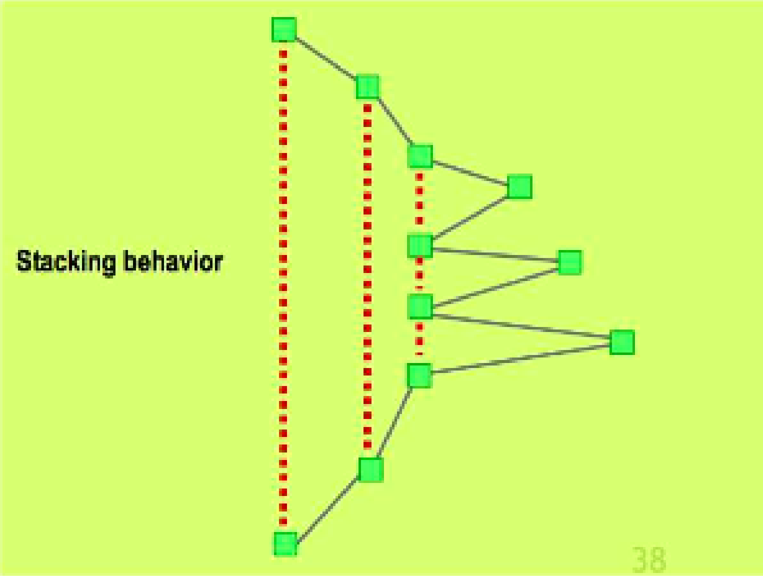




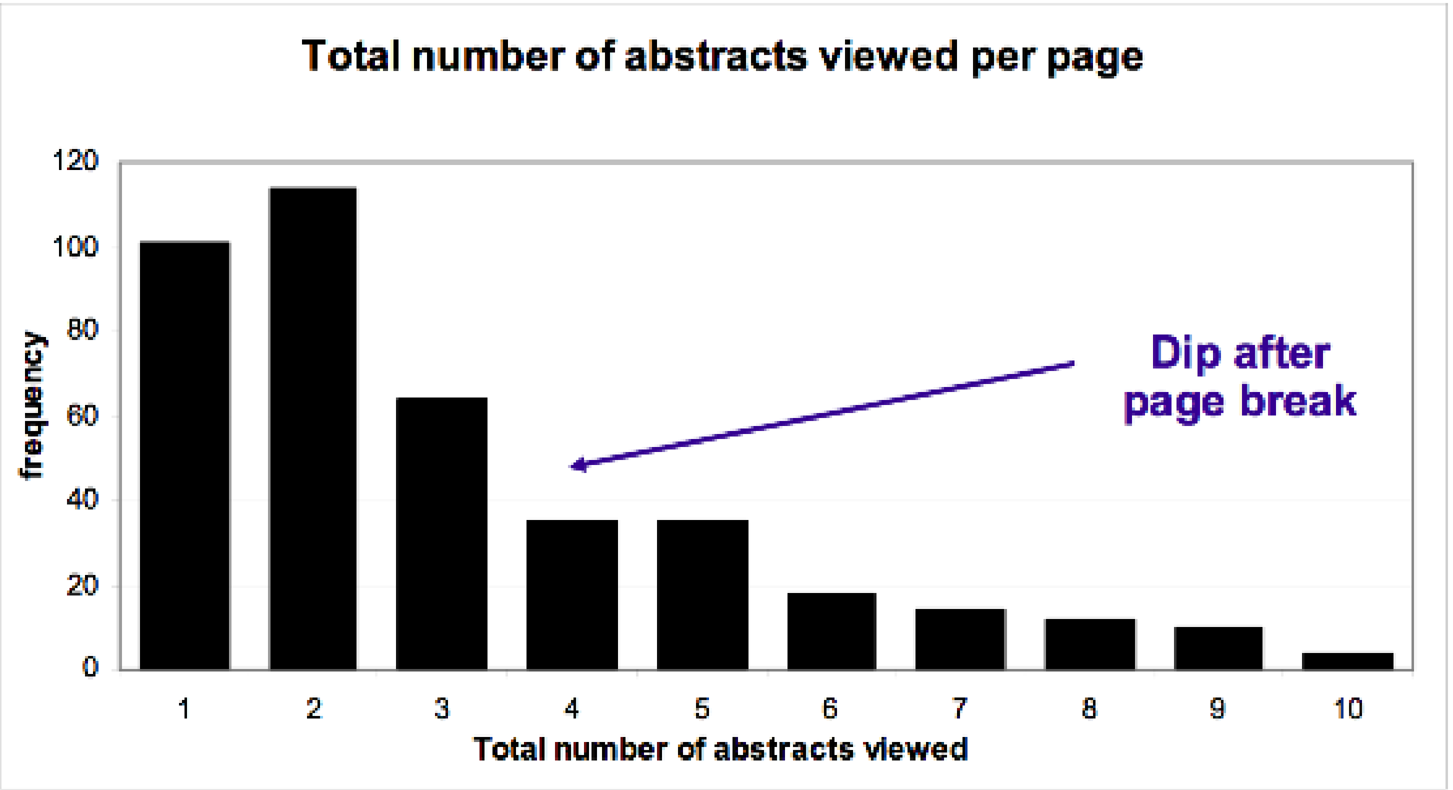






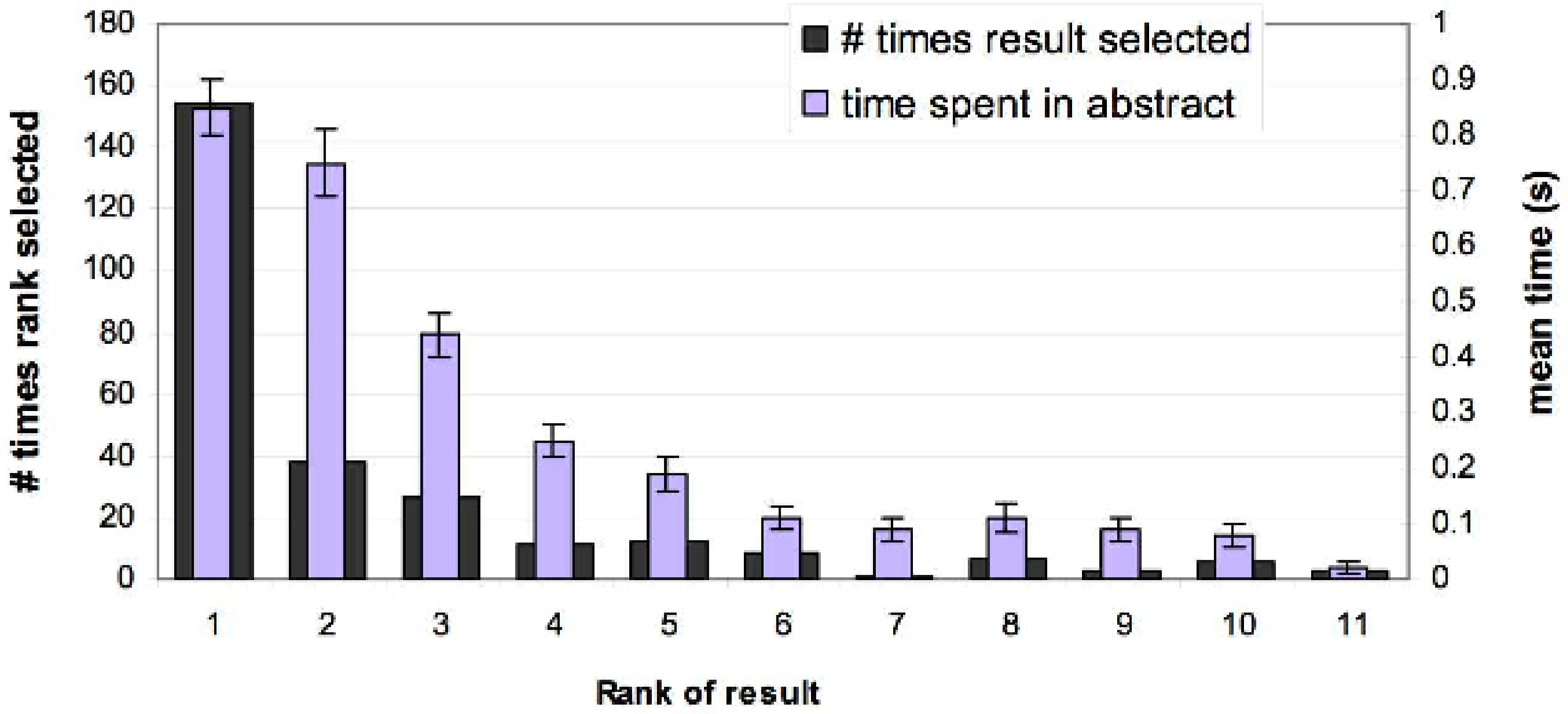


How many links do users view?



Mean: 3.07   Median/Mode: 2.00

Looking vs. Clicking

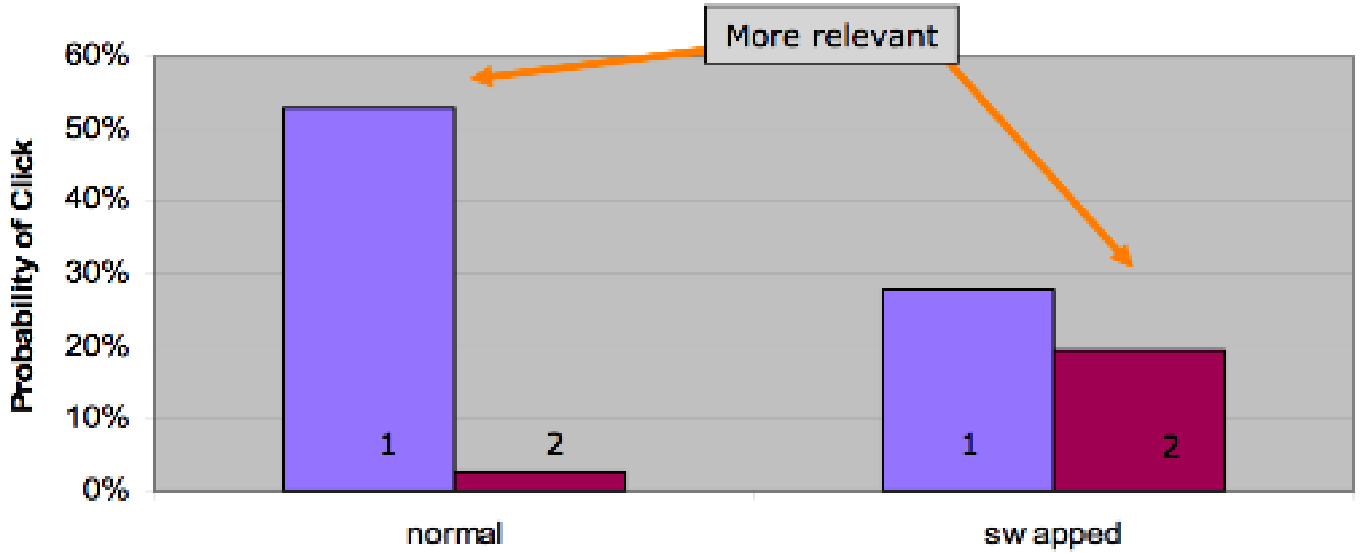


- Users view results one and two more often / thoroughly
- Users click most frequently on result one

Google

Presentation bias – reversed results

- Order of presentation influences where users look **AND** where they click



Importance of ranking: Summary

- Viewing abstracts: Users are a lot more likely to read the abstracts of the top-ranked pages (1, 2, 3, 4) than the abstracts of the lower ranked pages (7, 8, 9, 10).
- Clicking: Distribution is even more skewed for clicking
- In 1 out of 2 cases, users click on the top-ranked page.
- Even if the top-ranked page is not relevant, 30% of users will click on it.
- → Getting the ranking right is very important.
- → Getting the top-ranked page right is most important.

Scoring as the basis of ranked retrieval

Scoring as the basis of ranked retrieval

- We wish to rank documents that are more relevant higher than documents that are less relevant.
- How can we accomplish such a ranking of the documents in the collection with respect to a query?
- Assign a score to each query-document pair, say in [0, 1].
- This score measures how well document and query "match".

Factors impacting query-document score

- …
- …
- …

Query-document matching scores

- How do we compute the score of a query-document pair?
- Let’s start with a one-term query.
- If the query term does not occur in the document: score should be 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.

Take 1: Jaccard coefficient

- A commonly used measure of overlap of two sets
- Let *A* and *B* be two sets
- Jaccard coefficient:

$$\textsc{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

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

- JACCARD $(A, A) = 1$
- JACCARD $(A, B) = 0$ if $A \cap B = 0$
- A and B don’t have to be the same size.
- Always assigns a number between 0 and 1.

Jaccard coefficient: 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

What’s wrong with Jaccard?

- It doesn’t consider term frequency (how many occurrences a term has).
- Rare terms are more informative than frequent terms. Jaccard does not consider this information.
- We need a more sophisticated way of normalizing for the length of a document.

Term Frequency

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 $\in \{0, 1\}^{|V|.}$

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 $\in N^{|V|.}$

Bag of words model

- We do not consider the order of words in a document.
- *“John is quicker than Mary”* and *“Mary is quicker than John”* are represented the same way.
- This is called a bag of words model.

Term frequency tf

- The term frequency $\mathrm{tf}_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.
- We want to use tf when computing query-document match scores.
- But how?
- Raw term frequency is not what we want because:
- A document with tf = 10 occurrences of the term is more relevant than a document with tf = 1 occurrence of the term.
- But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Instead of raw frequency: Log frequency weighting

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

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

- $\mathrm{tf}_{t,d} \rightarrow \mathrm{w}_{t,d}$:
  0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, etc.
- Score for a document-query pair: sum over terms t in both *q* and *d*:
  tf-matching-score(*q*, *d*) = $\sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$
- The score is 0 if none of the query terms is present in the document.

$$\textsc{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

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

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

Compute the Jaccard matching score and the tf matching score for the following query-document pairs.

- q: [information on cars] d: “all you’ve ever wanted to know about cars”
- q: [information on cars] d: “information on trucks, information on planes, information on trains”

TF-IDF Weighting

Frequency in document vs. frequency in collection

- In addition, to term frequency (the frequency of the term in the document) . . .
- . . .we also want to use the frequency of the term in the collection for weighting and ranking.

Desired weight for rare terms

- Rare terms are more informative than frequent terms.
- Consider a term in the query that is rare in the collection (e.g., ARACHNOCENTRIC).
- A document containing this term is very likely to be relevant.
- → We want high weights for rare terms like ARACHNOCENTRIC.

Desired weight for frequent terms

- Frequent terms are less informative than rare terms.
- Consider a term in the query that is frequent in the collection (e.g., GOOD, INCREASE, LINE).
- A document containing this term is more likely to be relevant than a document that doesn't . . .
- . . . but words like GOOD, INCREASE and LINE are not sure indicators of relevance.
- → For frequent terms like GOOD, INCREASE and LINE, we want positive weights . . .
- . . . but lower weights than for rare terms.

Document frequency

- We want high weights for rare terms like ARACHNOCENTRIC.
- We want low (positive) weights for frequent words like GOOD, INCREASE and LINE.
- We will use document frequency to factor this into computing the matching score.
- The document frequency is the number of documents in the collection that the term occurs in.

idf weight

- df$_t$ is the document frequency, the number of documents that *t* occurs in.
- df$_t$ is an inverse measure of the informativeness of term *t*.
- We define the idf weight of term t as follows:

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

(*N* is the number of 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
- Note that we use the log transformation for both term frequency and document frequency.

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 the ranking of documents for queries with at least two terms.
- For example, in the query “arachnocentric line”, idf weighting increases the relative weight of ARACHNOCENTRIC and decreases the relative weight of LINE.
- idf has little effect on ranking for one-term queries.

Collection frequency vs. Document frequency

word	collection frequency	document frequency
INSURANCE	10440	3997
TRY	10422	8760

- Collection frequency of *t*: number of tokens of *t* in the collection
- Document frequency of *t*: number of documents *t* occurs in
- Why these numbers?
- Which word is a better search term (and should get a higher weight)?
- This example suggests that df (and idf) is better for weighting than cf (and “icf”).

tf-idf weighting

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

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

- tf-weight
- idf-weight
- Best known weighting scheme in information retrieval
- Note: the “-” in tf-idf is a hyphen, not a minus sign!
- Alternative names: tf.idf, tf x idf

Summary: tf-idf

- Assign a tf-idf weight for each term t in each document *d*:

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

- The tf-idf weight . . .
  - . . . increases with the number of occurrences within a document. (term frequency)
  - . . . increases with the rarity of the term in the collection. (inverse document frequency)

The Vector Space Model

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 $\in \{0, 1\}^{|V|.}$

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 ∈ $N^{|V|}$.

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 idf weights ∈ $R^{|V|}$.

Documents as vectors

- Each document is now represented as a real-valued vector of tf-idf weights ∈ R$^{|V|}$.
- So we have a $|V|$-dimensional real-valued vector space.
- Terms are axes of the space.
- Documents are points or vectors in this space.
- Very high-dimensional: tens of millions of dimensions when you apply this to web search engines
- Each vector is very sparse - most entries are zero.

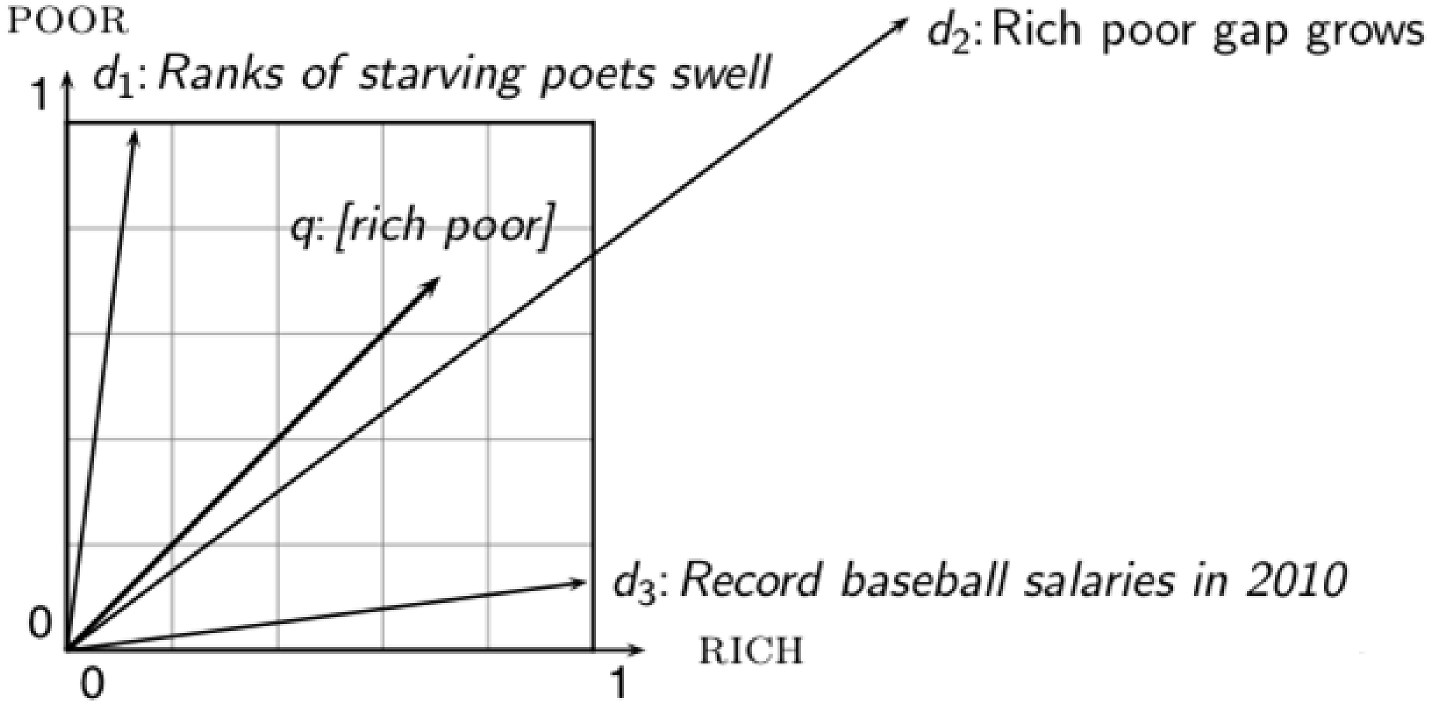
Queries as vectors

- Key idea 1: do the same for queries: represent them as vectors in the high-dimensional space
- Key idea 2: Rank documents according to their proximity to the query
- proximity = similarity
- proximity ≈ negative distance
- Recall: We’re doing this because we want to get away from the you’re-either-in-or-out, feast-or-famine Boolean model.
- Instead: rank relevant documents higher than nonrelevant documents

How do we formalize vector space similarity?

- First cut: (negative) distance between two points
- (= distance between the end points of the two vectors)
- Euclidean distance?
- Euclidean distance is a bad idea . . .
- . . . because Euclidean distance is large for vectors of different lengths.

Why distance is a bad idea


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

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.

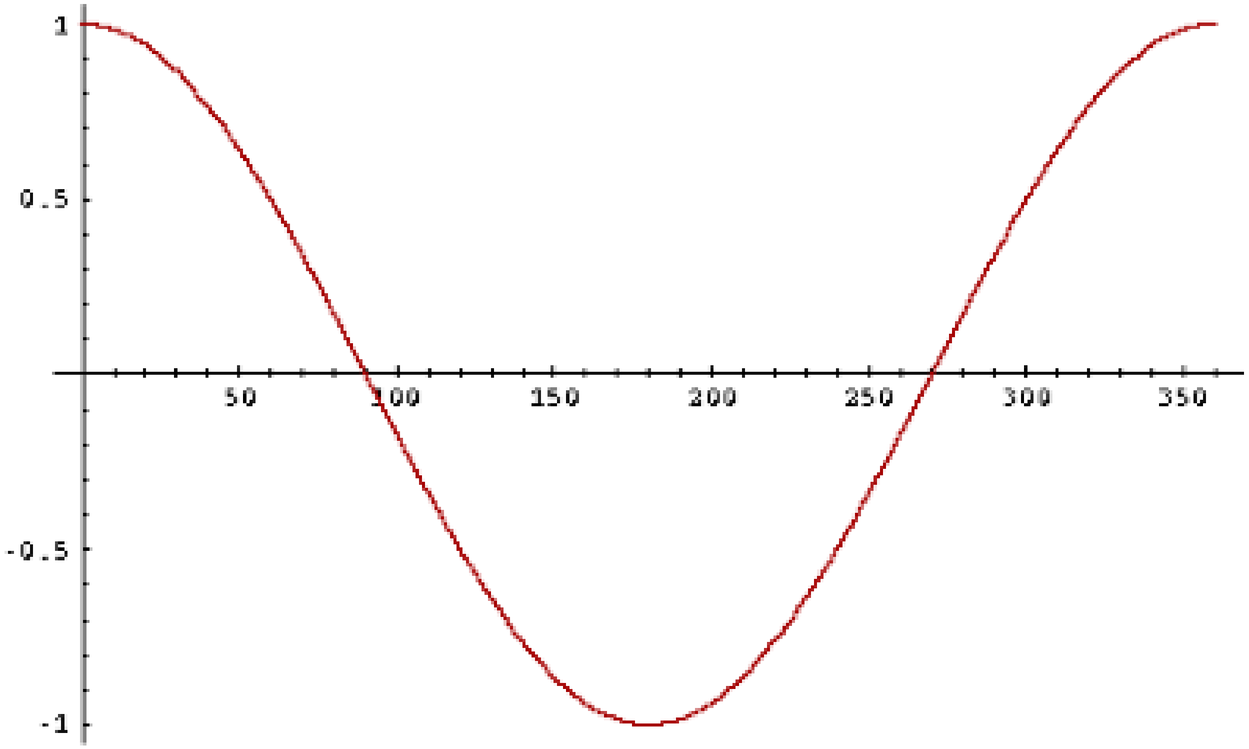
Use angle instead of distance

- Rank documents according to angle with query
- Thought experiment: take a document d and append it to itself. Call this document *d′*. *d′* is twice as long as *d*.
- “Semantically” *d* and *d′* have the same content.
- The angle between the two documents is 0, corresponding to maximal similarity . . .
- . . . even though the Euclidean distance between the two documents can be quite large.

From angles to cosines

- The following two notions are equivalent.
  - Rank documents according to the angle between query and document in decreasing order
  - Rank documents according to cosine(query,document) in increasing order
- Cosine is a monotonically decreasing function of the angle for the interval [0°, 180°]

Cosine



Length normalization

- How do we compute the cosine?
- A vector can be (length-) normalized by dividing each of its components by its length – here we use the L_2 norm: $||x||_2 = \sqrt{\sum_i x_i^2}$
- This maps vectors onto the unit sphere . . .
- . . . since after normalization: $||x||_2 = \sqrt{\sum_i x_i^2} = 1.0$
- As a result, longer documents and shorter documents have weights of the same order of magnitude.
- Effect on the two documents *d* and *d′* (*d* appended to itself) from earlier slide: they have identical vectors after length-normalization.

Cosine similarity between query and document

$$\cos(\vec{q},\vec{d}) = \textsc{sim}(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

- q_i is the tf-idf weight of term *i* in the query.
- d_i is the tf-idf weight of term *i* in the document.
- | $\vec{q}$ | and | $\vec{d}$ | are the lengths of $\vec{q}$ and $\vec{d}$.
- This is the cosine similarity of $\vec{q}$ and $\vec{d}$. . . . . or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

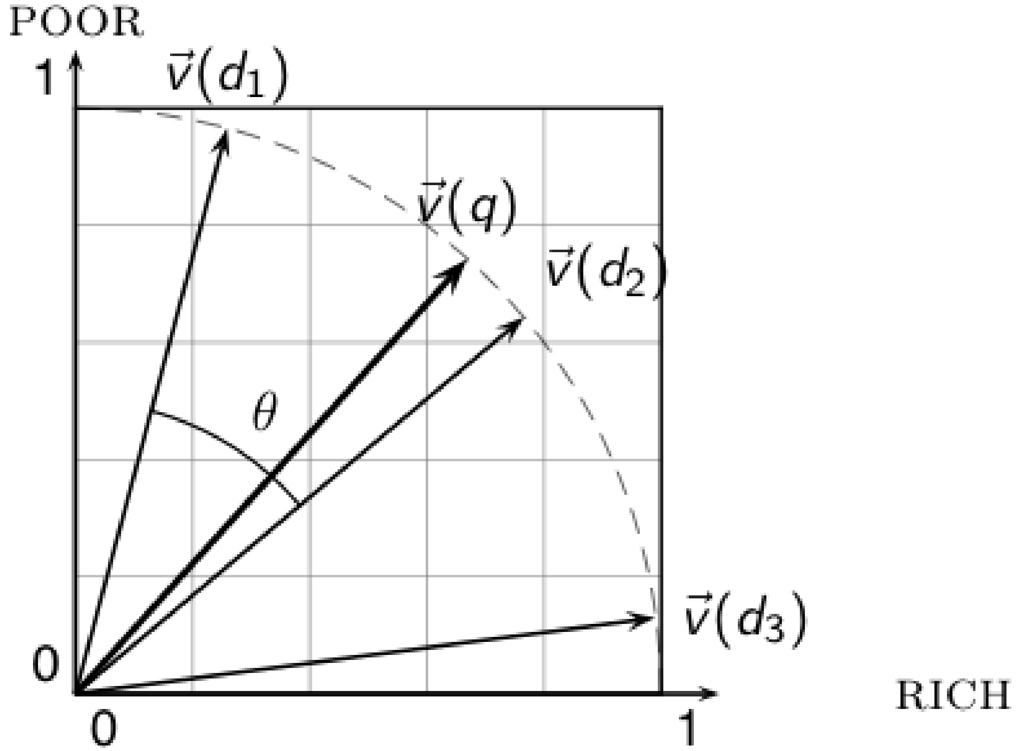
Cosine for normalized vectors

- For normalized vectors, the cosine is equivalent to the dot product or scalar product.

$$\cos(\vec{q}, \vec{d}) = \vec{q} \cdot \vec{d} = \sum_i q_i \cdot d_i$$

  - (if $\vec{q}$ and $\vec{d}$ are length-normalized).

Cosine similarity illustrated



Cosine: Example

How similar are these novels?

SaS: Sense and Sensibility

PaP: Pride and Prejudice

WH: Wuthering Heights

term frequencies (counts)

term	SaS	PaP	WH
AFFECTION	115	58	20
JEALOUS	10	7	11
GOSSIP	2	0	6
WUTHERING	0	0	38

Cosine: Example

term frequencies (counts)

term	SaS	PaP	WH
AFFECTION	115	58	20
JEALOUS	10	7	11
GOSSIP	2	0	6
WUTHERING	0	0	38

log frequency weighting

term	SaS	PaP	WH
AFFECTION	3.06	2.76	2.30
JEALOUS	2.0	1.85	2.04
GOSSIP	1.30	0	1.78
WUTHERING	0	0	2.58

(To simplify this example, we don't do idf weighting.)

Cosine: Example

log frequency weighting

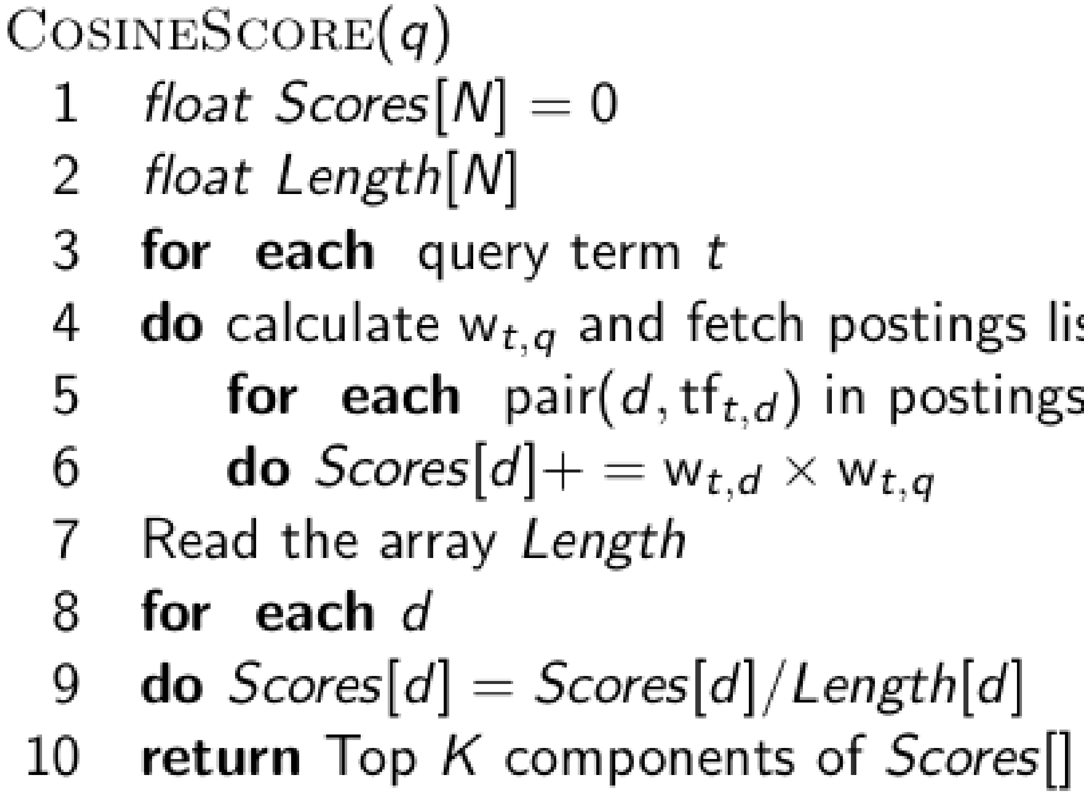
term	SaS	PaP	WH
AFFECTION	3.06	2.76	2.30
JEALOUS	2.0	1.85	2.04
GOSSIP	1.30	0	1.78
WUTHERING	0	0	2.58

log frequency weighting &
cosine normalization

term	SaS	PaP	WH
AFFECTION	0.789	0.832	0.524
JEALOUS	0.515	0.555	0.465
GOSSIP	0.335	0.0	0.405
WUTHERING	0.0	0.0	0.588

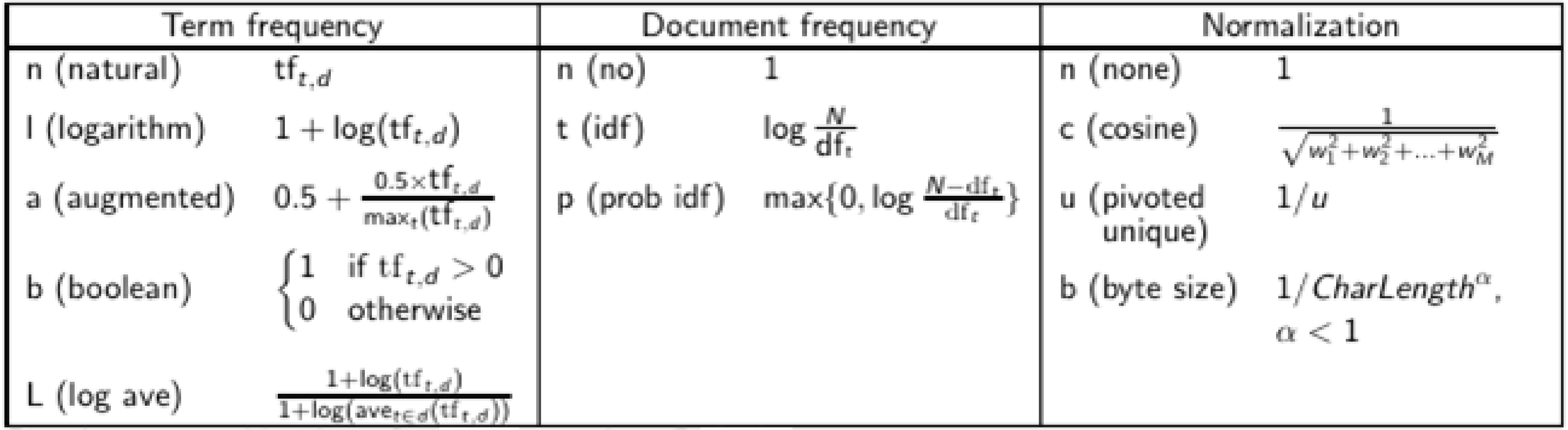
- cos(SaS,PaP) ≈
  $0.789 * 0.832 + 0.515 * 0.555 + 0.335 * 0.0 + 0.0 * 0.0 \approx 0.94.$
- cos(SaS,WH) ≈ 0.79
- cos(PaP,WH) ≈ 0.69
- Why do we have cos(SaS,PaP) > cos(SAS,WH)?

Computing the cosine score



```
CosineScore($q$)
float $Scores[N] = 0$
float $Length[N]$
for each query term $t$
do calculate $w_{t,q}$ and fetch postings list for $t$
    for each pair($d$, $tf_{t,d}$) in postings list
    do $Scores[d] += w_{t,d} \times w_{t,q}$
Read the array $Length$
for each $d$
do $Scores[d] = Scores[d] / Length[d]$
return Top $K$ components of $Scores[]$
```

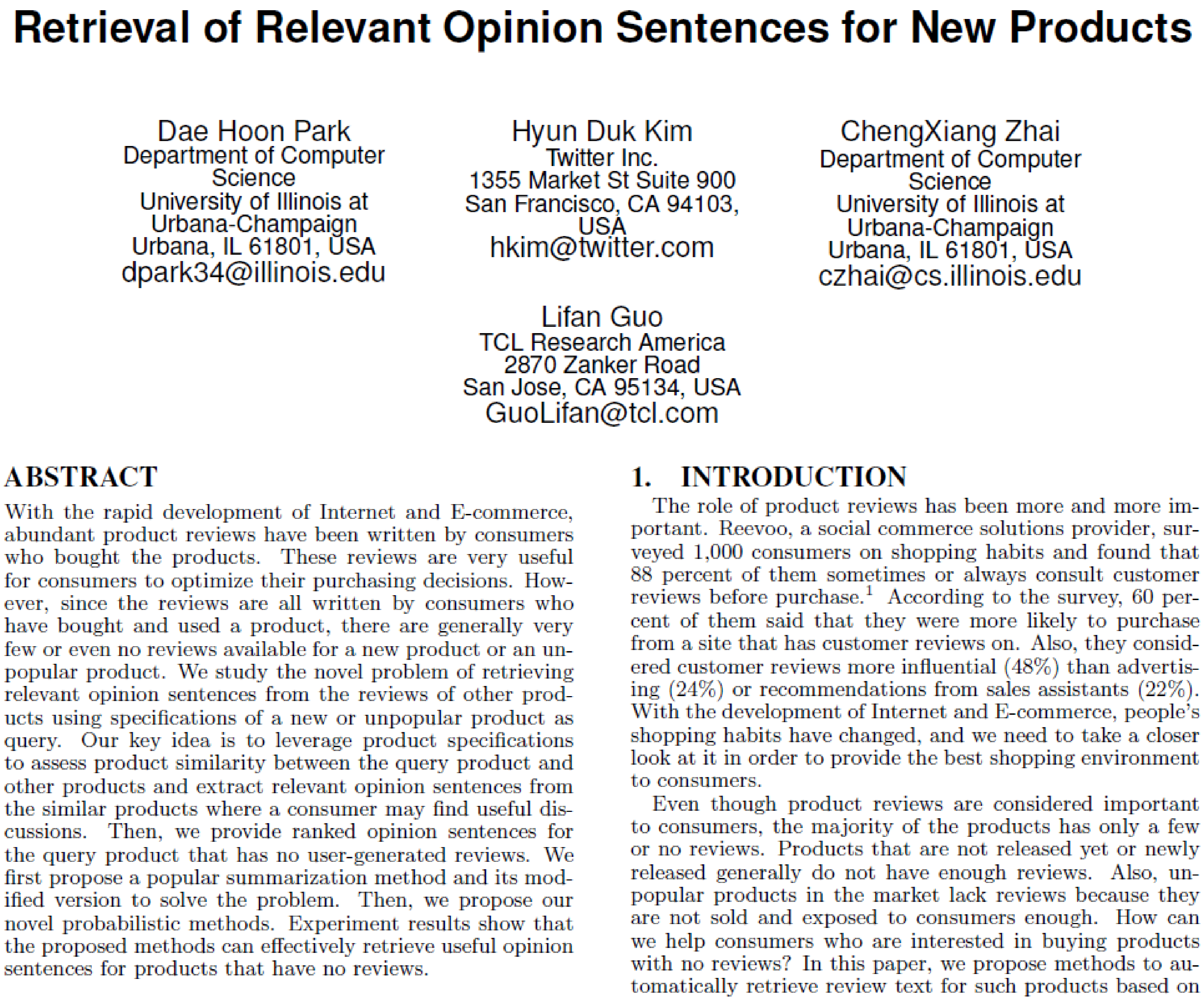
Components of tf-idf weighting



Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+...+w_M^2}}$
a (augmented)	$0.5 + \frac{0.5\times\mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

Summary: 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



Retrieval of Relevant Opinion Sentences for New Products

Dae Hoon Park
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
dpark34@illinois.edu

Hyun Duk Kim
Twitter Inc.
1355 Market St Suite 900
San Francisco, CA 94103, USA
hkim@twitter.com

ChengXiang Zhai
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
czhai@cs.illinois.edu

Lifan Guo
TCL Research America
2870 Zanker Road
San Jose, CA 95134, USA
GuoLifan@tcl.com

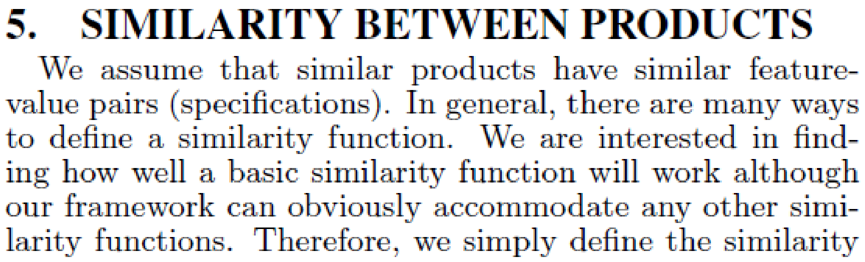
ABSTRACT

With the rapid development of Internet and E-commerce, abundant product reviews have been written by consumers who bought the products. These reviews are very useful for consumers to optimize their purchasing decisions. However, since the reviews are all written by consumers who have bought and used a product, there are generally very few or even no reviews available for a new product or an unpopular product. We study the novel problem of retrieving relevant opinion sentences from the reviews of other products using specifications of a new or unpopular product as query. Our key idea is to leverage product specifications to assess product similarity between the query product and other products and extract relevant opinion sentences from the similar products where a consumer may find useful discussions. Then, we provide ranked opinion sentences for the query product that has no user-generated reviews. We first propose a popular summarization method and its modified version to solve the problem. Then, we propose our novel probabilistic methods. Experiment results show that the proposed methods can effectively retrieve useful opinion sentences for products that have no reviews.

1. INTRODUCTION

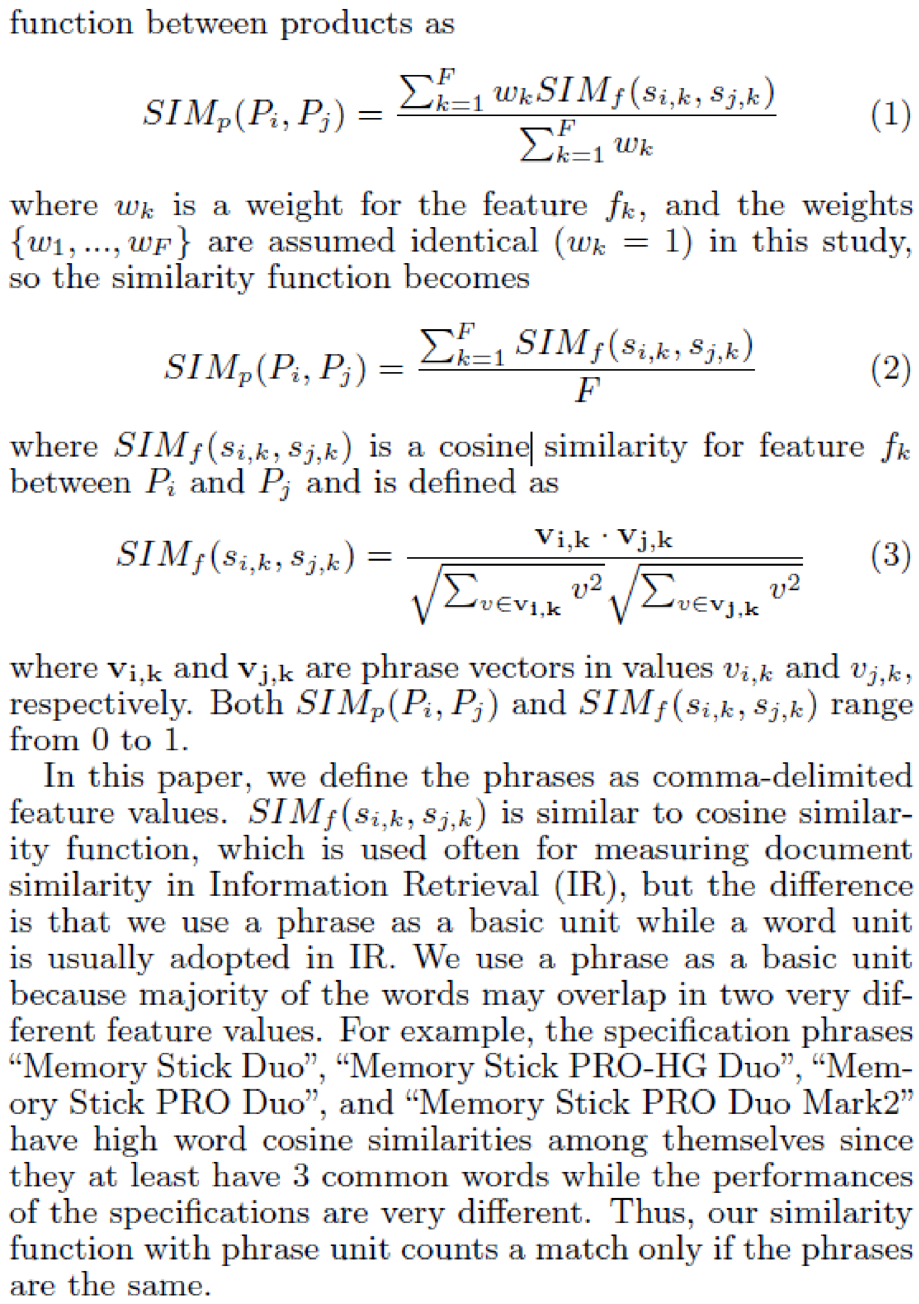
The role of product reviews has been more and more important. Reevoo, a social commerce solutions provider, surveyed 1,000 consumers on shopping habits and found that 88 percent of them sometimes or always consult customer reviews before purchase.[1] According to the survey, 60 percent of them said that they were more likely to purchase from a site that has customer reviews on. Also, they considered customer reviews more influential (48%) than advertising (24%) or recommendations from sales assistants (22%). With the development of Internet and E-commerce, people's shopping habits have changed, and we need to take a closer look at it in order to provide the best shopping environment to consumers.

Even though product reviews are considered important to consumers, the majority of the products has only a few or no reviews. Products that are not released yet or newly released generally do not have enough reviews. Also, unpopular products in the market lack reviews because they are not sold and exposed to consumers enough. How can we help consumers who are interested in buying products with no reviews? In this paper, we propose methods to automatically retrieve review text for such products based on



5. SIMILARITY BETWEEN PRODUCTS

We assume that similar products have similar feature-value pairs (specifications). In general, there are many ways to define a similarity function. We are interested in finding how well a basic similarity function will work although our framework can obviously accommodate any other similarity functions. Therefore, we simply define the similarity



function between products as

$$SIM_p(P_i, P_j) = \frac{\sum_{k=1}^{F} w_k SIM_f(s_{i,k}, s_{j,k})}{\sum_{k=1}^{F} w_k} \quad (1)$$

where w_k is a weight for the feature f_k, and the weights $\{w_1, ..., w_F\}$ are assumed identical ($w_k = 1$) in this study, so the similarity function becomes

$$SIM_p(P_i, P_j) = \frac{\sum_{k=1}^{F} SIM_f(s_{i,k}, s_{j,k})}{F} \quad (2)$$

where $SIM_f(s_{i,k}, s_{j,k})$ is a cosine similarity for feature f_k between P_i and P_j and is defined as

$$SIM_f(s_{i,k}, s_{j,k}) = \frac{\mathbf{v_{i,k}} \cdot \mathbf{v_{j,k}}}{\sqrt{\sum_{v \in \mathbf{v_{i,k}}} v^2}\sqrt{\sum_{v \in \mathbf{v_{j,k}}} v^2}} \quad (3)$$

where $\mathbf{v_{i,k}}$ and $\mathbf{v_{j,k}}$ are phrase vectors in values $v_{i,k}$ and $v_{j,k}$, respectively. Both $SIM_p(P_i, P_j)$ and $SIM_f(s_{i,k}, s_{j,k})$ range from 0 to 1.

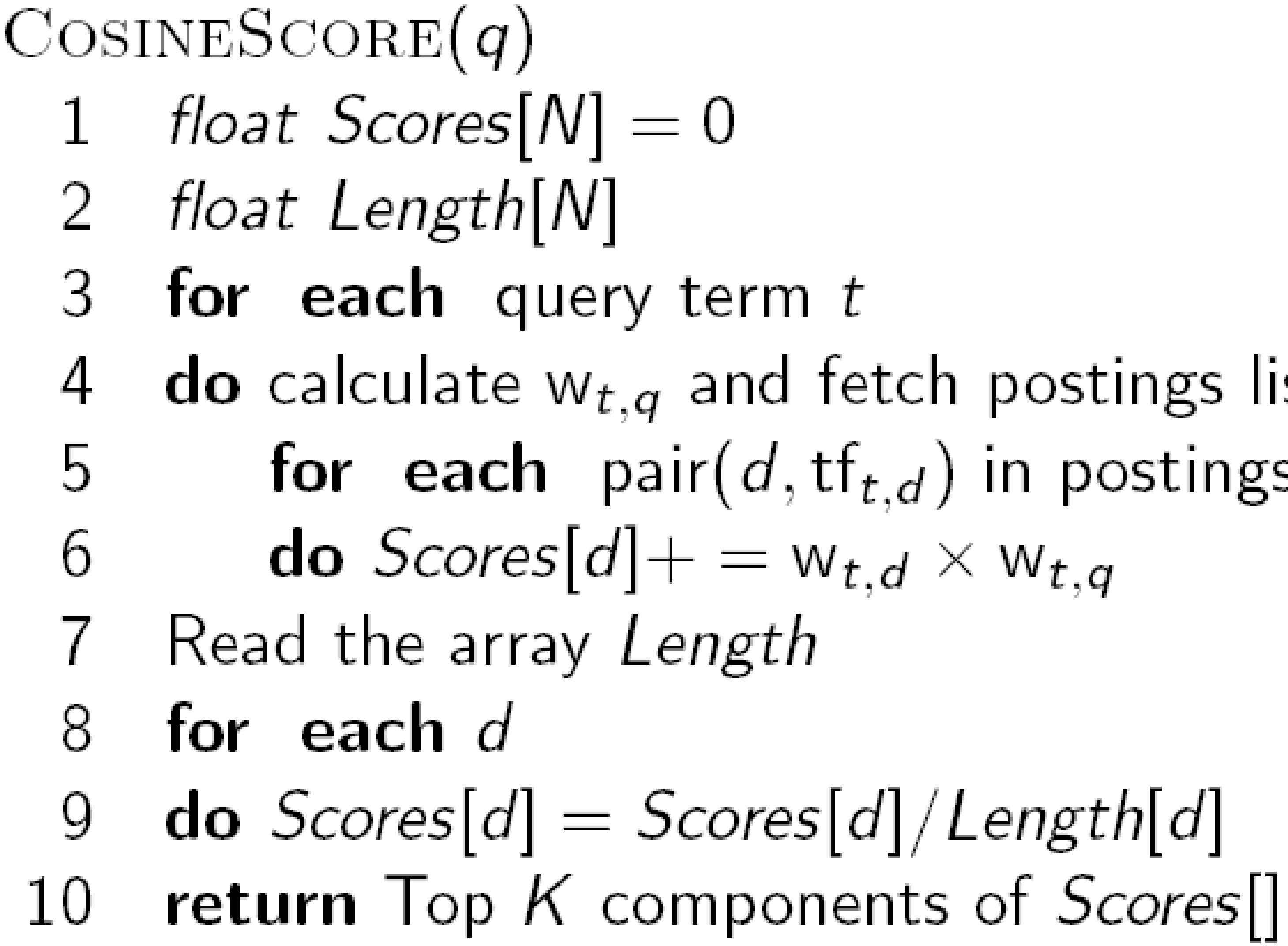
In this paper, we define the phrases as comma-delimited feature values. $SIM_f(s_{i,k}, s_{j,k})$ is similar to cosine similarity function, which is used often for measuring document similarity in Information Retrieval (IR), but the difference is that we use a phrase as a basic unit while a word unit is usually adopted in IR. We use a phrase as a basic unit because majority of the words may overlap in two very different feature values. For example, the specification phrases "Memory Stick Duo", "Memory Stick PRO-HG Duo", "Memory Stick PRO Duo", and "Memory Stick PRO Duo Mark2" have high word cosine similarities among themselves since they at least have 3 common words while the performances of the specifications are very different. Thus, our similarity function with phrase unit counts a match only if the phrases are the same.

Computing scores in a complete search system

This lecture

- Speeding up vector space ranking
- Putting together a complete search system
  - Will require learning about a number of miscellaneous topics and heuristics

Computing cosine scores



```
CosineScore($q$)
float $Scores[N] = 0$
float $Length[N]$
for each query term $t$
do calculate $\mathrm{w}_{t,q}$ and fetch postings list for $t$
    for each pair($d$, $\mathrm{tf}_{t,d}$) in postings list
    do $Scores[d] += \mathrm{w}_{t,d} \times \mathrm{w}_{t,q}$
Read the array $Length$
for each $d$
do $Scores[d] = Scores[d]/Length[d]$
return Top $K$ components of $Scores[]$
```

Efficient cosine ranking

- Find the *K* docs in the collection “nearest” to the query ⇒ *K* largest query-doc cosines.
- Efficient ranking:
  - Computing a single cosine efficiently.
  - Choosing the *K* largest cosine values efficiently.
    - Can we do this without computing all *N* cosines?

Special case – unweighted queries

- No weighting on query terms
  - Assume each query term occurs only once
- Then for ranking, don’t need to normalize query vector
  - Slight simplification of algorithm from IIR Chapter 6

Computing the *K* largest cosines: selection vs. sorting

- Typically we want to retrieve the top *K* docs (in the cosine ranking for the query)
  - not to totally order all docs in the collection
- Can we pick off docs with *K* highest cosines?
- Let *J* = number of docs with nonzero cosines
  - We seek the *K* best of these *J*

Bottlenecks

- Primary computational bottleneck in scoring: cosine computation
- Can we avoid all this computation?
- Yes, but may sometimes get it wrong
  - a doc *not* in the top *K* may creep into the list of *K* output docs
  - Is this such a bad thing?

Cosine similarity is only a proxy

- User has a task and a query formulation
- Cosine matches docs to query
- Thus cosine is anyway a proxy for user happiness
- If we get a list of *K* docs “close” to the top *K* by cosine measure, should be ok

Generic approach

- Find a set A of *contenders*, with $K < |A| \ll N$
  - A does not necessarily contain the top K, but has many docs from among the top K
  - Return the top K docs in A
- Think of A as pruning non-contenders
- Will look at several schemes following this approach

Index elimination

- Basic algorithm: cosine computation algorithm only considers docs containing at least one query term
- Take this further:
  - Only consider high-idf query terms
  - Only consider docs containing many query terms

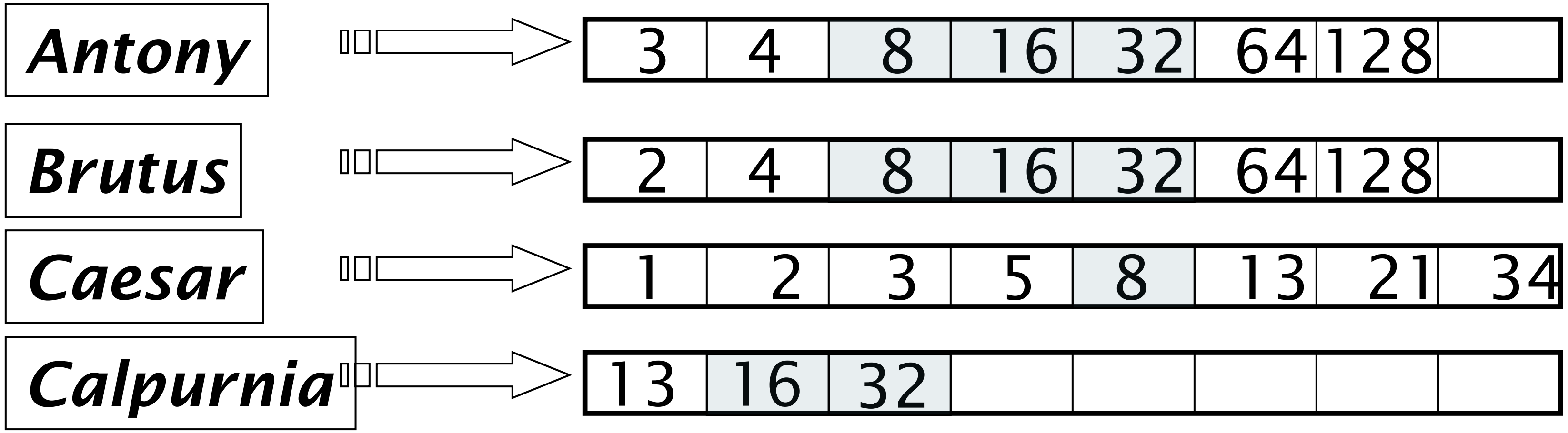
High-idf query terms only

- For a query such as *catcher in the rye*
- Only accumulate scores from *catcher* and *rye*
- Intuition: ***in*** and ***the*** contribute little to the scores and so don’t alter rank-ordering much
- Benefit:
  - Postings of low-idf terms have many docs → these (many) docs get eliminated from set *A* of contenders

Docs containing many query terms

- Any doc with at least one query term is a candidate for the top *K* output list
- For multi-term queries, only compute scores for docs containing several of the query terms
  - Say, at least 3 out of 4
- Easy to implement in postings traversal

3 of 4 query terms



Antony → 3 4 8 16 32 64 128

Brutus → 2 4 8 16 32 64 128

Caesar → 1 2 3 5 8 13 21 34

Calpurnia → 13 16 32

Scores only computed for docs 8, 16 and 32.

Champion lists

- Precompute for each dictionary term *t,* the *r* docs of highest weight in *t*'s postings
  - Call this the champion list for *t*
  - (aka fancy list or top docs for *t*)
- Note that *r* has to be chosen at index build time
  - Thus, it's possible that *r* < *K*
- At query time, only compute scores for docs in the champion list of some query term
  - Pick the *K* top-scoring docs from amongst these

Static quality scores

- We want top-ranking documents to be both *relevant* and *authoritative*
- *Relevance* is being modeled by cosine scores
- *Authority* is typically a query-independent property of a document
- Examples of authority signals
  - Wikipedia among websites
  - Articles in certain newspapers
  - A paper with many citations
  - Many bitly's marks
  - (Pagerank)

Quantitative

Modeling authority

- Assign to each document a *query-independent* quality score in [0,1] to each document *d*
  - Denote this by *g(d)*

- Thus, a quantity like the number of citations is scaled into [0,1]

Net score

- Consider a simple total score combining cosine relevance and authority
- net-score(*q*,*d*) = *g*(*d*) + cosine(*q*,*d*)
  - Can use some other linear combination

- Now we seek the top *K* docs by net score

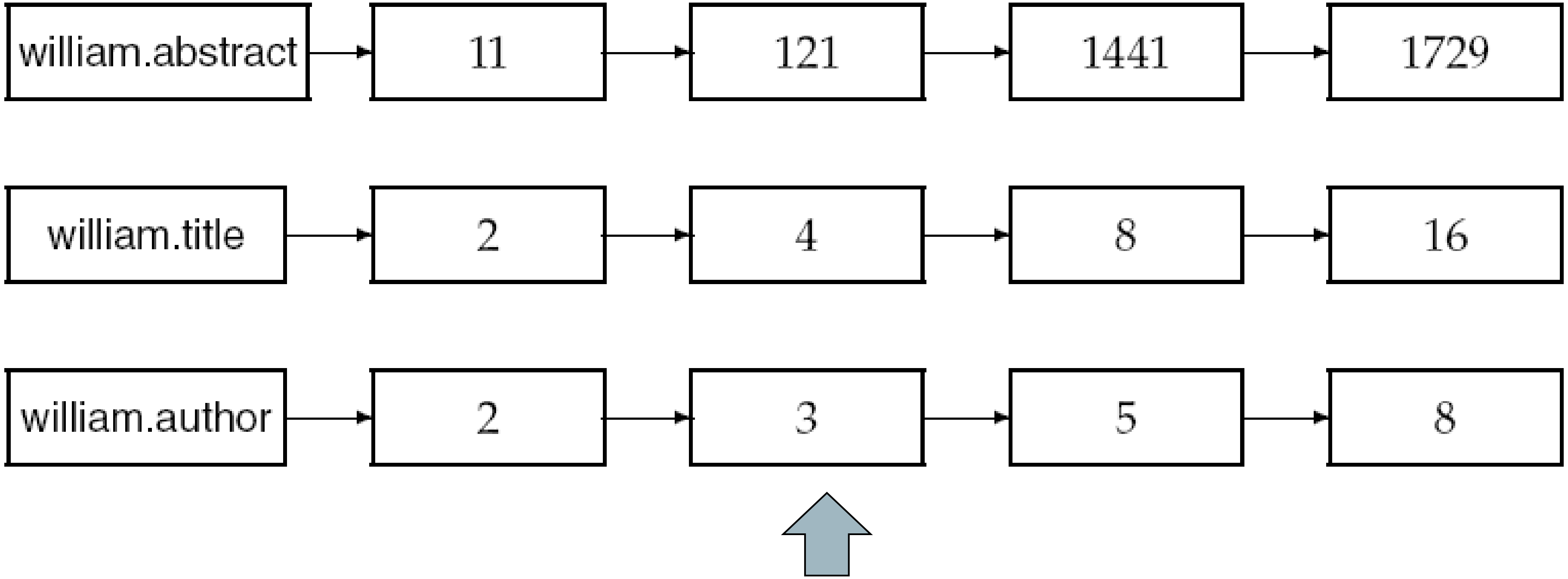
High and low lists

- For each term, we maintain two postings lists called *high* and *low*
  - Think of *high* as the champion list
- When traversing postings on a query, only traverse *high* lists first
  - If we get more than *K* docs, select the top *K* and stop
  - Else proceed to get docs from the *low* lists
- Can be used even for simple cosine scores, without global quality *g(d)*
- A means for segmenting index into two tiers

Zone

- A zone is a region of the doc that can contain an arbitrary amount of text, e.g.,
  - Title
  - Abstract
  - References …
- Build inverted indexes on zones as well to permit querying
- E.g., “find docs with *merchant* in the title zone and matching the query *gentle rain”*

Example zone indexes

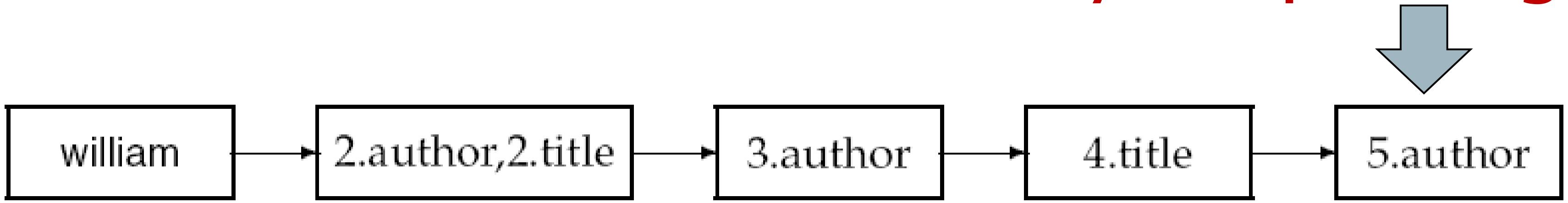


william.abstract → 11 → 121 → 1441 → 1729

william.title → 2 → 4 → 8 → 16

william.author → 2 → 3 → 5 → 8

Encode zones in dictionary vs. postings.

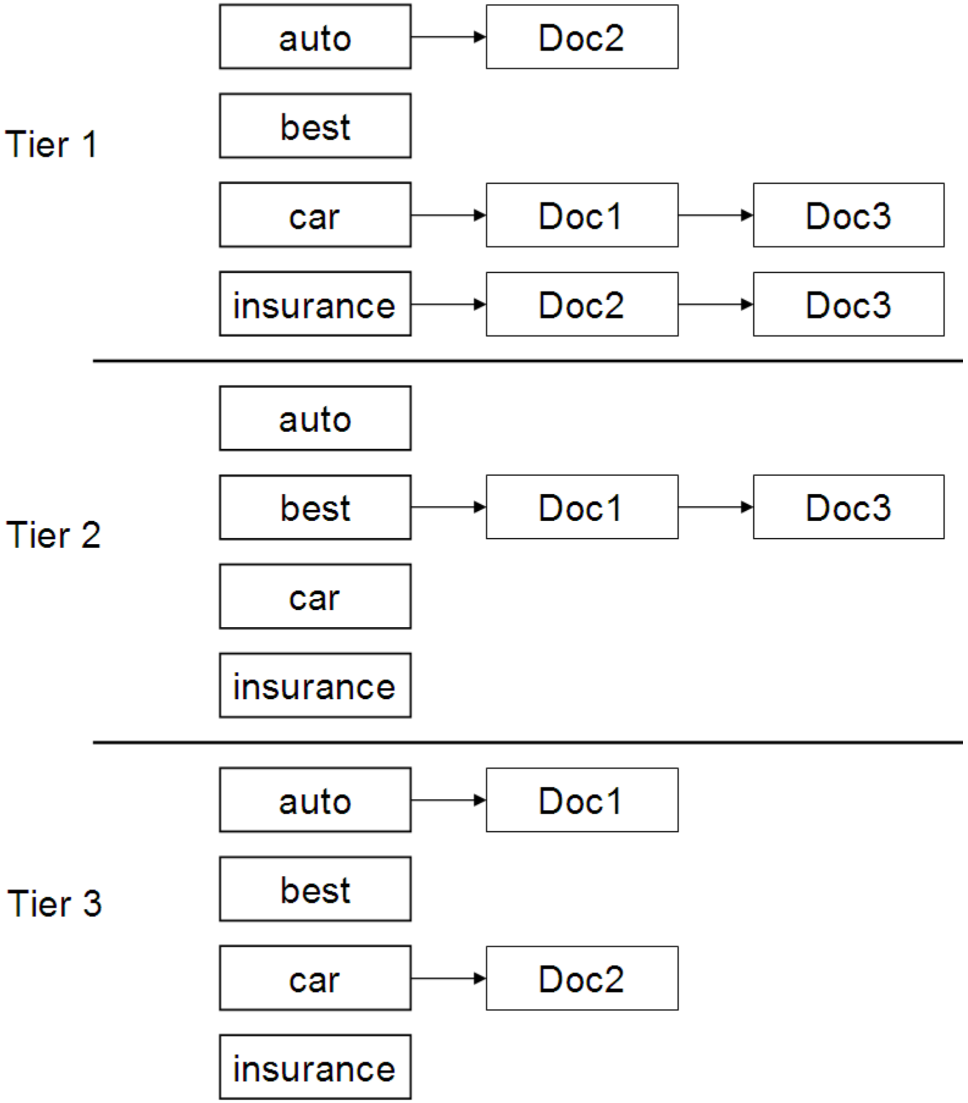


william → 2.author,2.title → 3.author → 4.title → 5.author

Tiered indexes

- Break postings up into a hierarchy of lists
  - Most important
  - …
  - Least important
- Can be done by *g(d)* or another measure
- Inverted index thus broken up into <u>tiers</u> of decreasing importance
- At query time use top tier unless it fails to yield *K* docs
  - If so drop to lower tiers

Example tiered index



Tier 1

auto → Doc2

best

car → Doc1 → Doc3

insurance → Doc2 → Doc3

Tier 2

auto

best → Doc1 → Doc3

car

insurance

Tier 3

auto → Doc1

best

car → Doc2

insurance

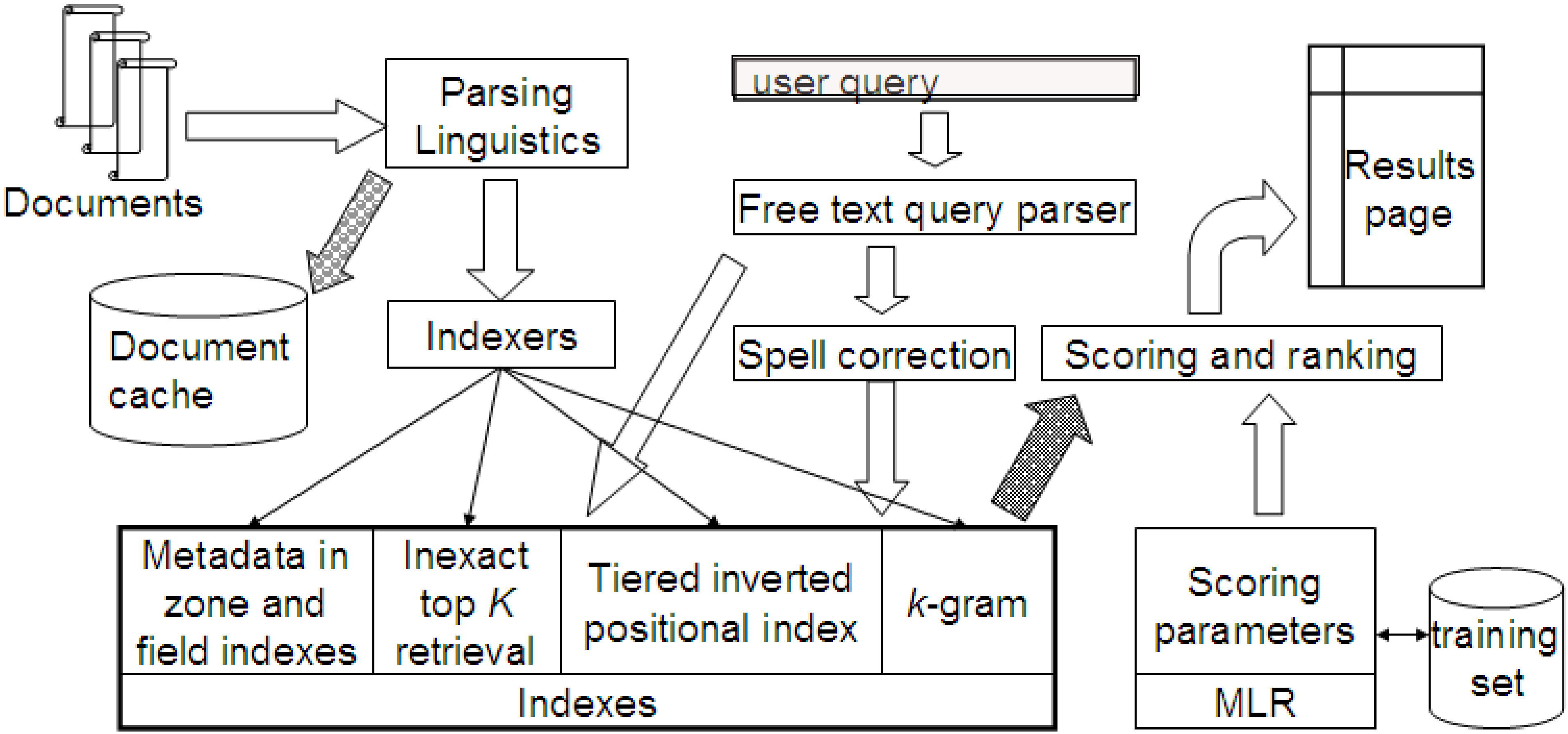
Query parsers

- Free text query from user may in fact spawn one or more queries to the indexes, e.g., query *rising interest rates*
  - Run the query as a phrase query
  - If <*K* docs contain the phrase *rising interest rates*, run the two phrase queries *rising interest* and *interest rates*
  - If we still have <*K* docs, run the vector space query *rising interest rates*
  - Rank matching docs by vector space scoring
- This sequence is issued by a query parser

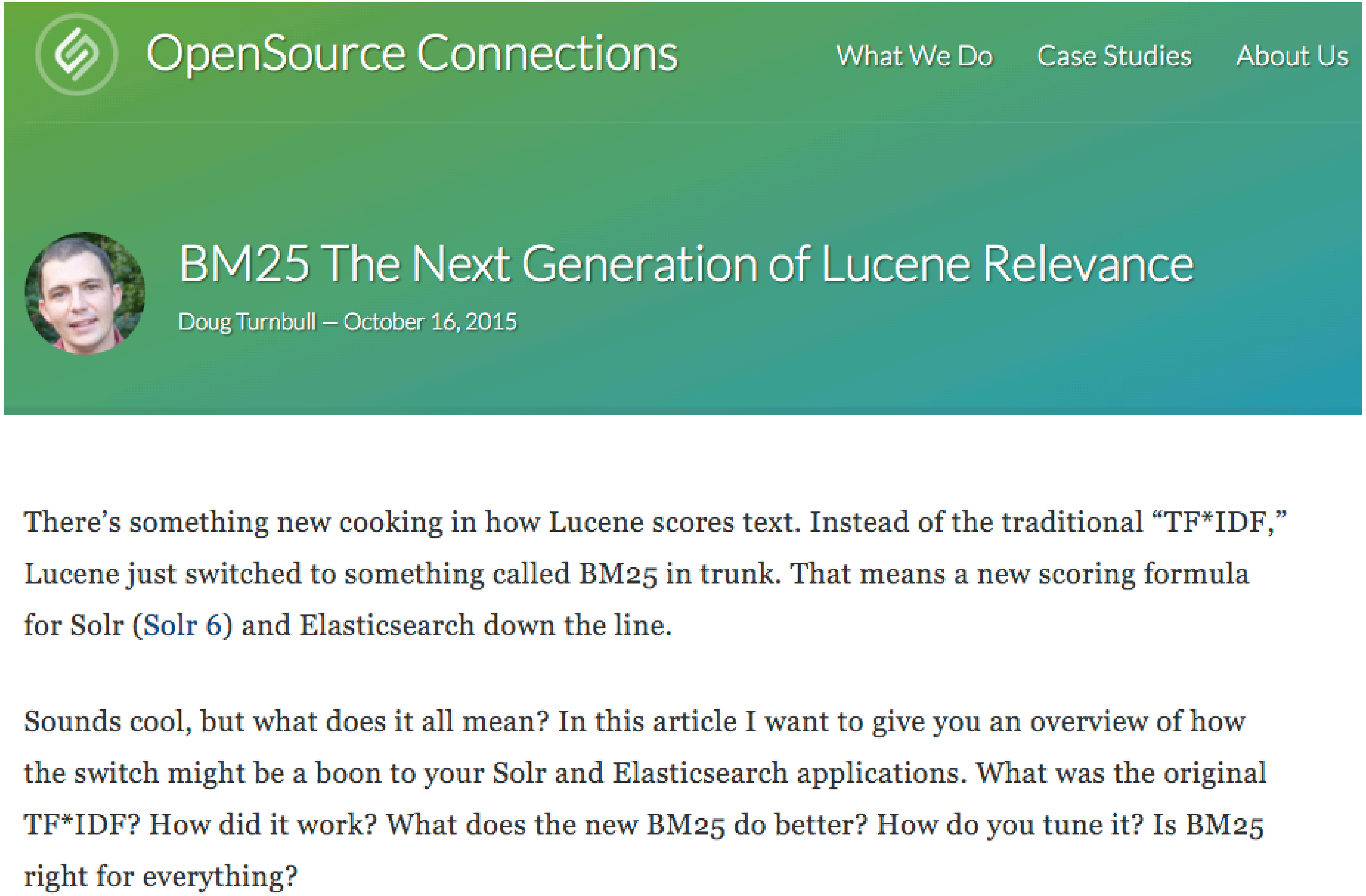
Aggregate scores

- We’ve seen that score functions can combine cosine, static quality, etc.
- How do we know the best combination?
- Some applications – expert-tuned
- Increasingly common: machine-learned

Putting it all together



BM25



BM25 The Next Generation of Lucene Relevance

Doug Turnbull — October 16, 2015

There's something new cooking in how Lucene scores text. Instead of the traditional "TF*IDF," Lucene just switched to something called BM25 in trunk. That means a new scoring formula for Solr (Solr 6) and Elasticsearch down the line.

Sounds cool, but what does it all mean? In this article I want to give you an overview of how the switch might be a boon to your Solr and Elasticsearch applications. What was the original TF*IDF? How did it work? What does the new BM25 do better? How do you tune it? Is BM25 right for everything?

https://opensourceconnections.com/blog/2015/10/16/bm25-the-next-generation-of-lucene-relevation/

Okapi BM25 [Robertson et al. 1994, TREC City U.]

- BM25 “Best Match 25” (they had a bunch of tries!)
  - Developed in the context of the Okapi system
  - Started to be increasingly adopted by other teams during the TREC competitions
  - It works well

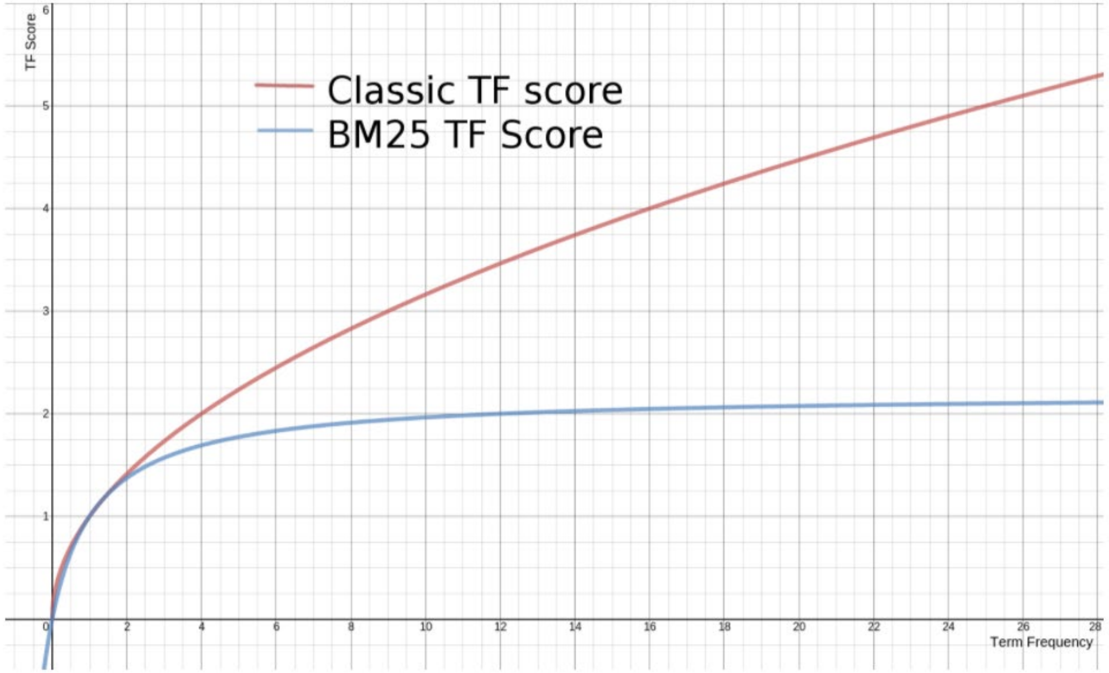
- Goal: be sensitive to term frequency and document length while not adding too many parameters
  - (Robertson and Zaragoza 2009; Spärck Jones et al. 2000)

“Early” version of BM25

- Version 2:

$$c_i^{BM25v2}(tf_i) = \log \frac{N}{df_i} \times \frac{(k_1+1)tf_i}{k_1+tf_i}$$

  - (k_1+1) factor doesn’t change ranking, but makes term score 1 when $tf_i = 1$
  - Similar to *tf-idf*, but term scores are bounded



But it still don’t model document length

➔ Longer documents are likely to have larger tf_i values

Document length normalization

- Document length:

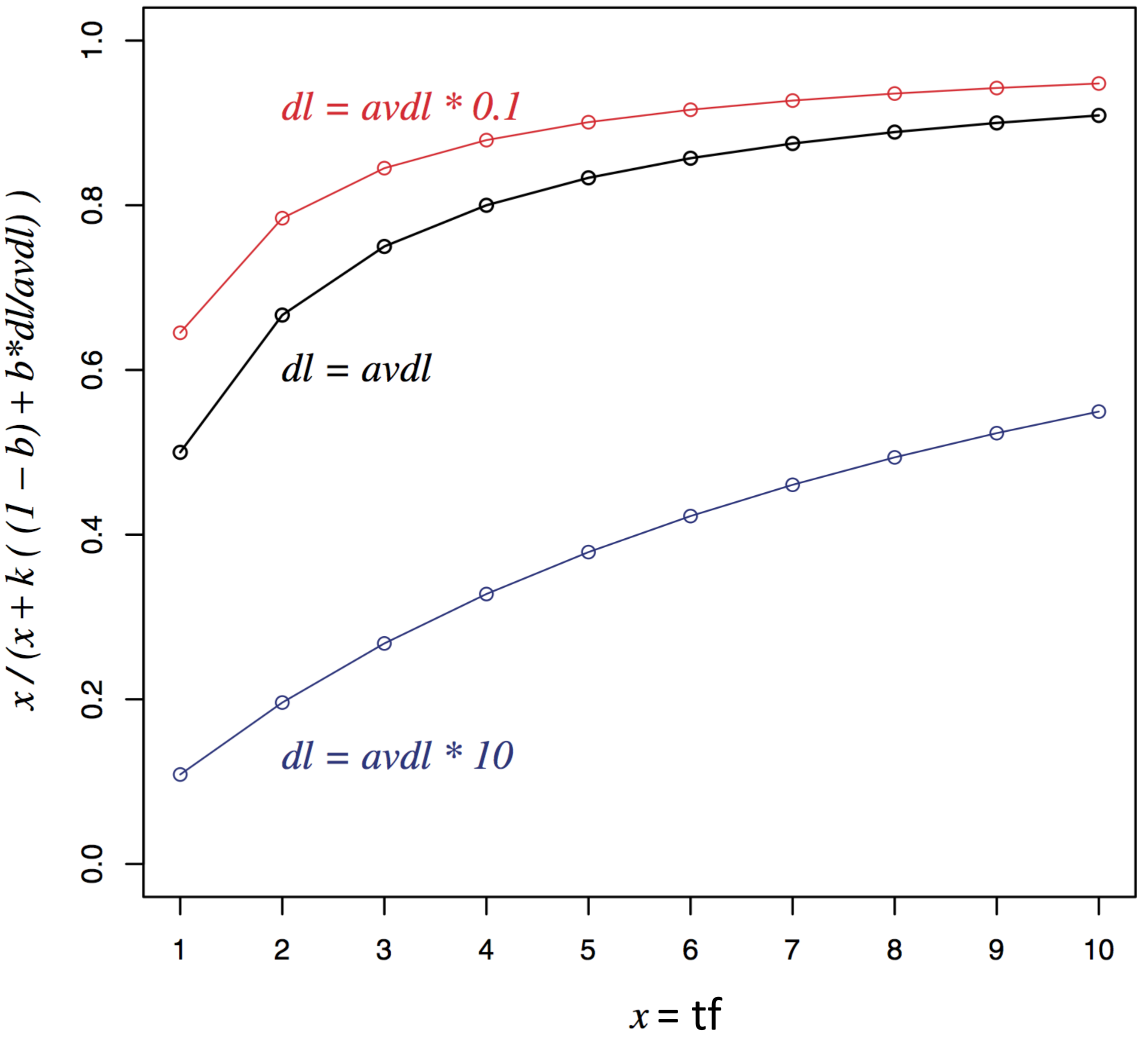
$$dl = \sum_{i \in V} tf_i$$

- $avdl$: Average document length over collection
- Length normalization component

$$B = \left((1-b) + b \frac{dl}{avdl} \right), \qquad 0 \le b \le 1$$

  - $b = 1$ full document length normalization
  - $b = 0$ no document length normalization

Document length normalization



Okapi BM25

- Normalize *tf* using document length

$$tf_i' = \frac{tf_i}{B}$$

$$\begin{aligned} c_i^{BM25}(tf_i) &= \log\frac{N}{df_i} \times \frac{(k_1+1)tf_i'}{k_1+tf_i'} \\ &= \log\frac{N}{df_i} \times \frac{(k_1+1)tf_i}{k_1((1-b)+b\frac{dl}{avdl})+tf_i} \end{aligned}$$

- BM25 ranking function

$$RSV^{BM25} = \sum_{i\in q} c_i^{BM25}(tf_i);$$

RSV = Retrieval Status Value

Okapi BM25

$$RSV^{BM25} = \sum_{i \in q} \log \frac{N}{df_i} \cdot \frac{(k_1+1)tf_i}{k_1((1-b)+b\frac{dl}{avdl})+tf_i}$$

- k_1 controls term frequency scaling
  - $k_1 = 0$ is binary model; k_1 large is raw term frequency
- b controls document length normalization
  - $b = 0$ is no length normalization; $b = 1$ is relative frequency (fully scale by document length)
- Typically, k_1 is set around 1.2–2 and b around 0.75