

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

So far, 12 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

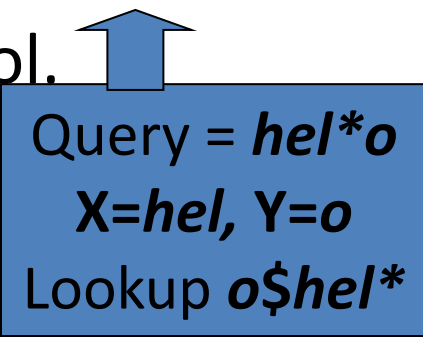
Skip pointers &
Positional index

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.



Interview video

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.

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.

The screenshot shows a Google search for "children's unicycle". The results are numbered 1 through 6, with red arrows indicating a scanning pattern that starts at the top, moves down to result 1, then up to result 2, then down to result 3, then up to result 4, then down to result 5, and finally up to result 6. The results are as follows:

- 1. **Unicycle.UK.com - F.A.Q. - What size?**
17" 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=53-23& - Cached - Similar pages
- 2. **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
- 3. **100 Miles for Kids - The Goal**
"The Afghan Mobile Mini Circus for Children is established ... attempt to break the GUINNESS WORLD RECORD for the ORL HIGHER UNICYCLE DISTANCE RECORD. ...
www.unicycle4kids.org/ - 9k - Cached - Similar pages
- 4. **Unicycles page at Juggling World**
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
- 5. **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
- 6. **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.oranregister.com/oranregister/news/homepage/article_1293785.php - 31k - Cached - Similar pages

Kinds of behaviors we see in the data

Short / Nav



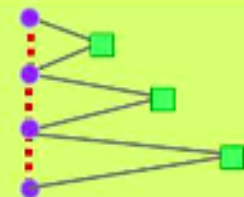
Topic exploration



Topic switch



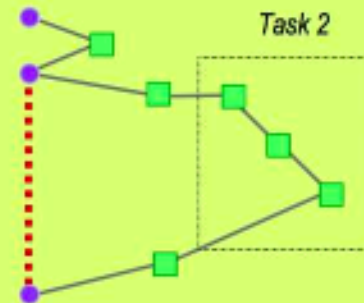
Methodical results exploration



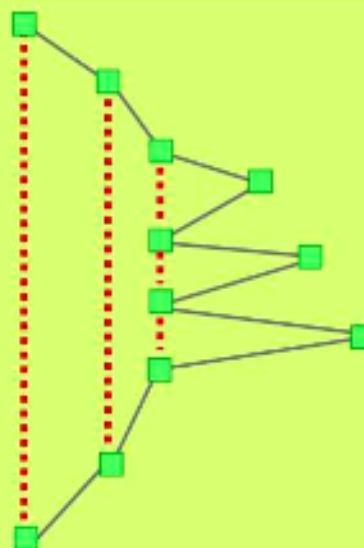
Query reform



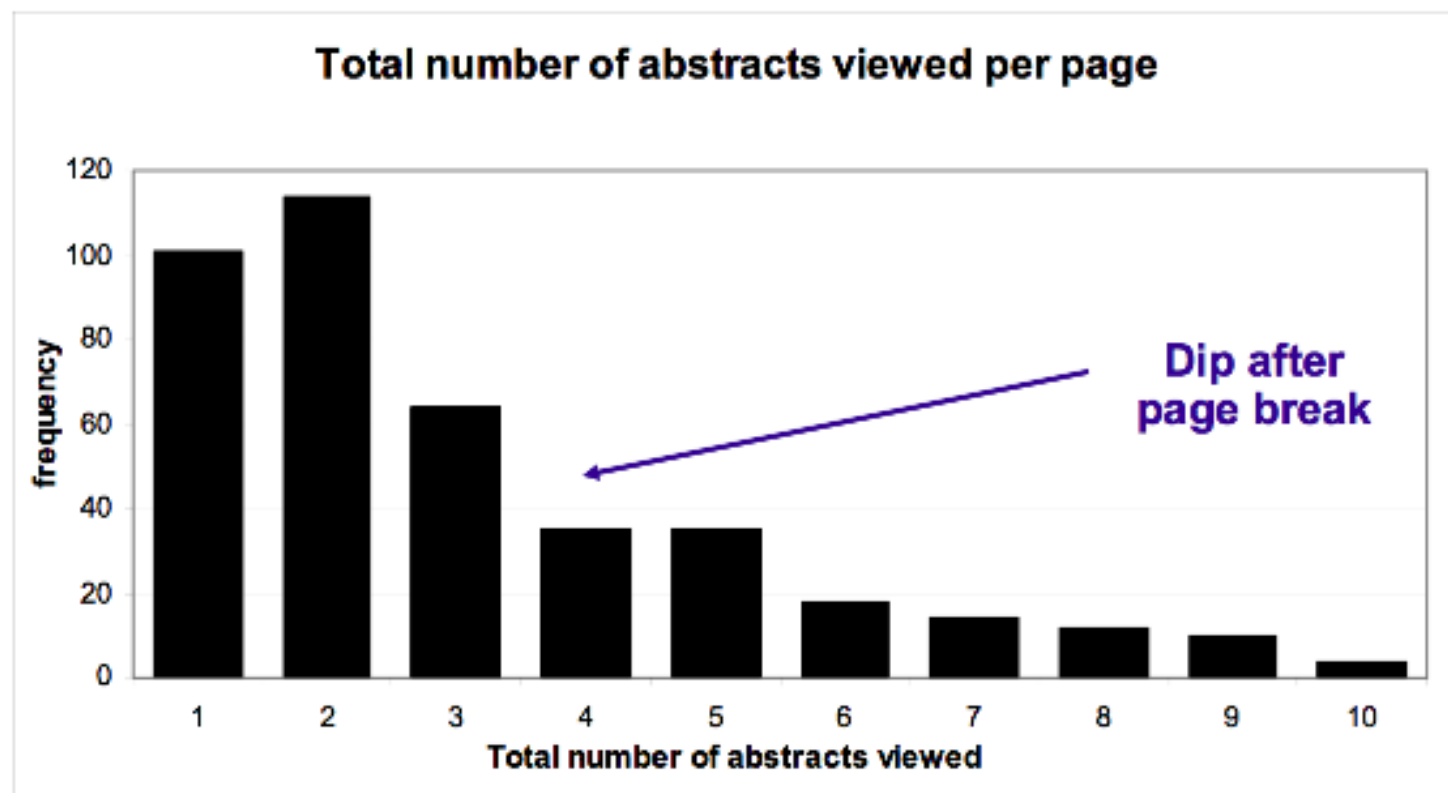
Multitasking



Stacking behavior

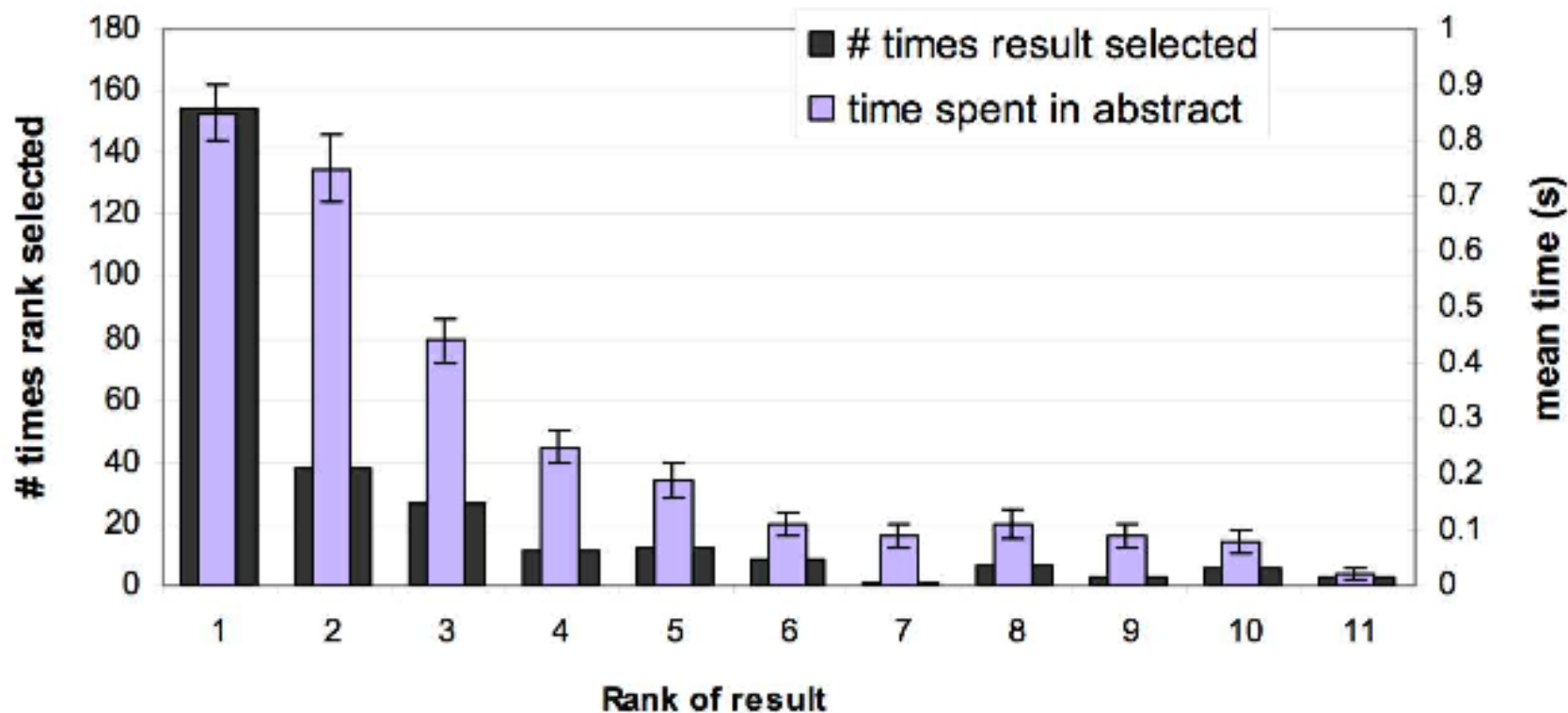


How many links do users view?



Mean: 3.07 Median/Mode: 2.00

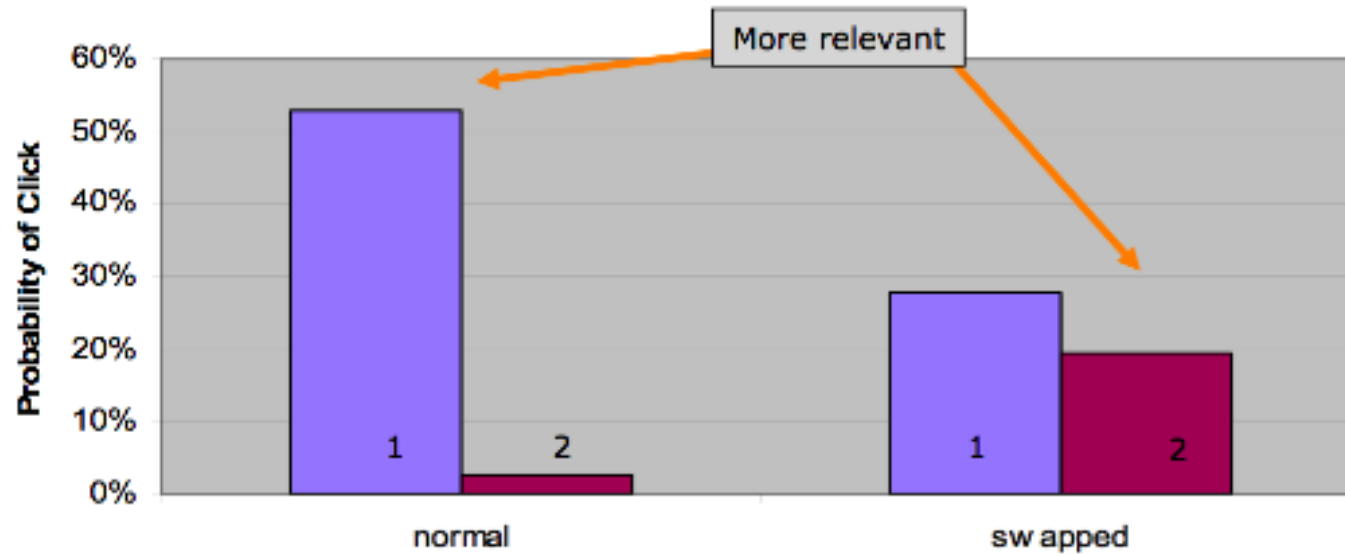
Looking vs. Clicking



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

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:

$$\text{JACCARD}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

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

- $\text{JACCARD}(A, A) = 1$
- $\text{JACCARD}(A, B) = 0$ if $A \cap B = \emptyset$
- 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 $\in \mathbb{N}^{|V|}$.

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 $tf_{t,d}$ 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

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

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

$$\text{JACCARD}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

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

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

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:

$$idf_t = \log_{10} \frac{N}{df_t}$$

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

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 “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**.

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

- 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 :

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

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

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 tfidf weights $\in \mathbb{R}^{|V|}$.

Documents as vectors

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

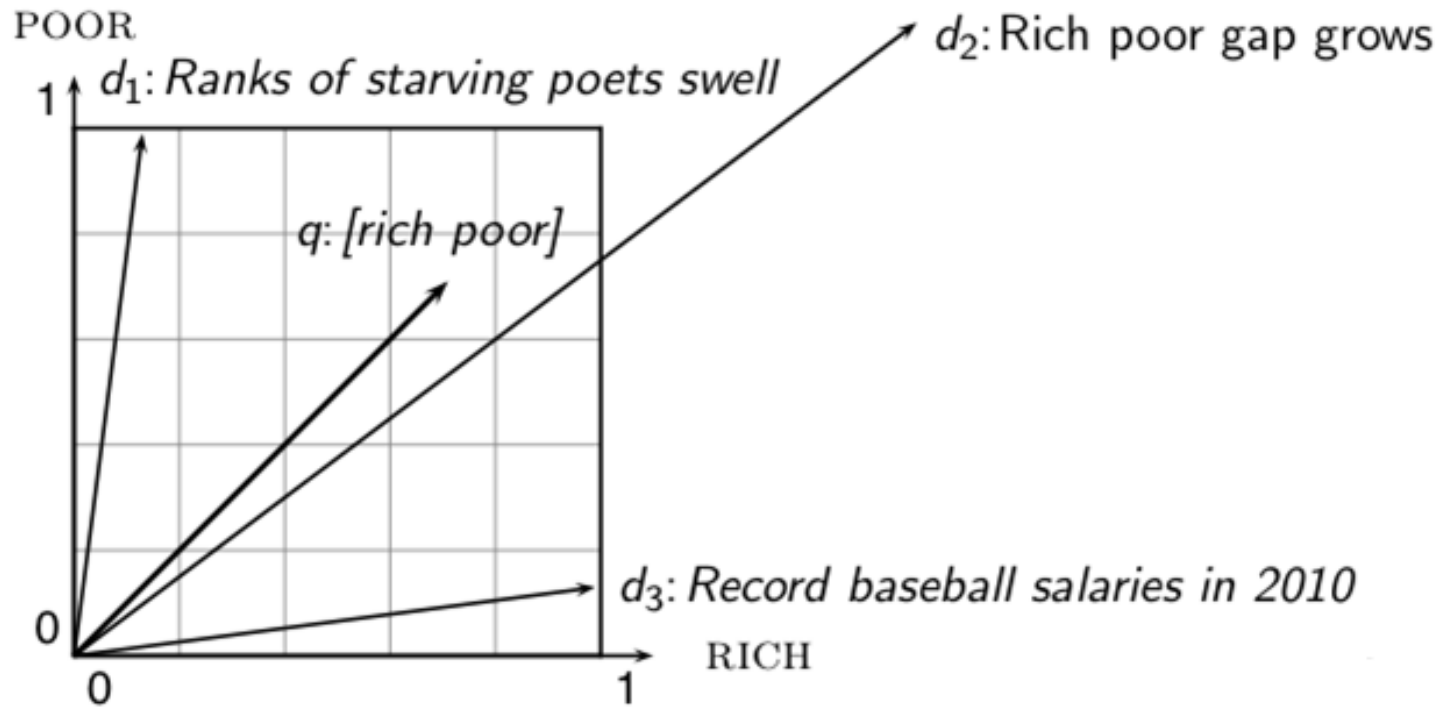
Queries as vectors

- 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 \approx 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



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.

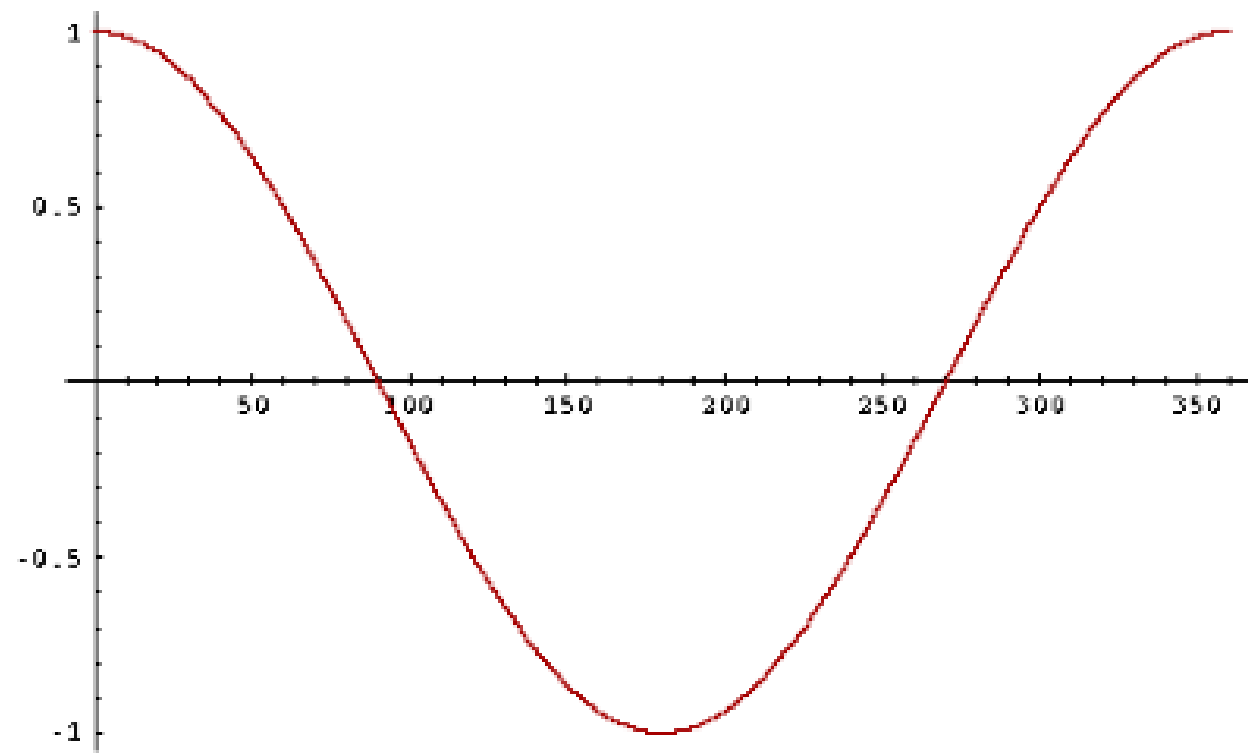
Use angle instead of distance

- 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 $[0^\circ, 180^\circ]$

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

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

- 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} .

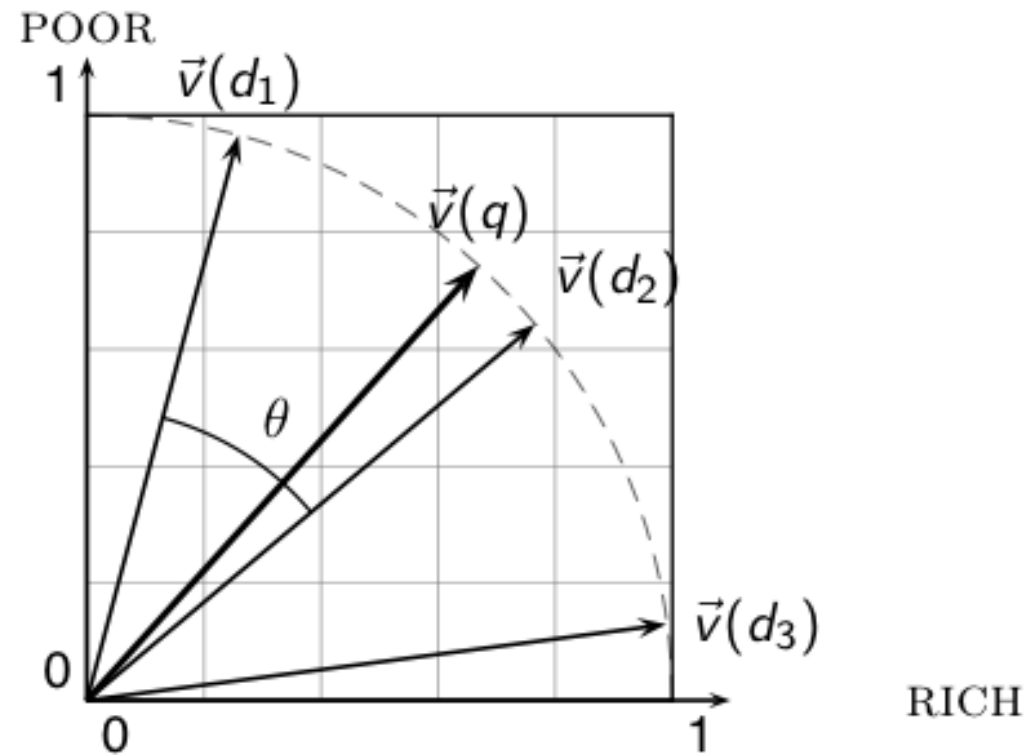
Cosine for normalized vectors

- 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

term frequencies (counts)

How similar are these novels?	term	SaS	PaP	WH
SaS: Sense and Sensibility	AFFECTION	115	58	20
PaP: Pride and Prejudice	JEALOUS	10	7	11
	GOSSIP	2	0	6
WH: Wuthering Heights	WUTHERING	0	0	38

Cosine: Example

term frequencies (counts)				log frequency weighting			
term	SaS	PaP	WH	term	SaS	PaP	WH
AFFECTION	115	58	20	AFFECTION	3.06	2.76	2.30
JEALOUS	10	7	11	JEALOUS	2.0	1.85	2.04
GOSSIP	2	0	6	GOSSIP	1.30	0	1.78
WUTHERING	0	0	38	WUTHERING	0	0	2.58

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

Cosine: Example

log frequency weighting				log frequency weighting & cosine normalization			
term	SaS	PaP	WH	term	SaS	PaP	WH
AFFECTION	3.06	2.76	2.30	AFFECTION	0.789	0.832	0.524
JEALOUS	2.0	1.85	2.04	JEALOUS	0.515	0.555	0.465
GOSSIP	1.30	0	1.78	GOSSIP	0.335	0.0	0.405
WUTHERING	0	0	2.58	WUTHERING	0.0	0.0	0.588

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

Computing the cosine score

COSINESCORE(q)

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

Components of tf-idf weighting

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

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. Reevo, 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 $\{w_1, \dots, w_F\}$ 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)

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

Efficient cosine ranking

- Find the K docs in the collection “nearest” to the query $\Rightarrow K$ 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
- $\text{net-score}(q, d) = g(d) + \text{cosine}(q, d)$
 - Can use some other linear combination
- Now we seek the top K docs by net score

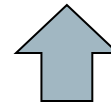
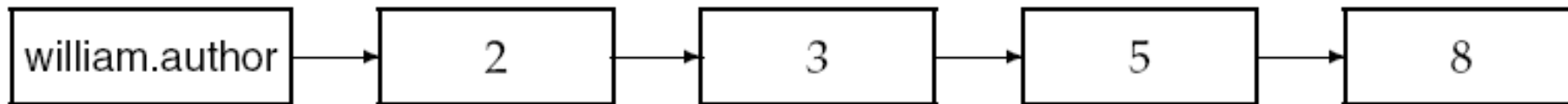
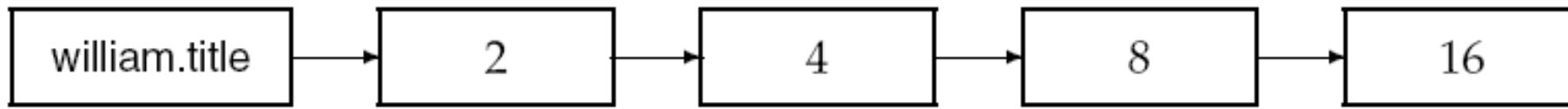
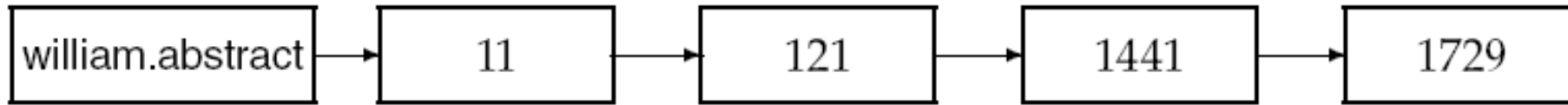
High and low lists

- 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

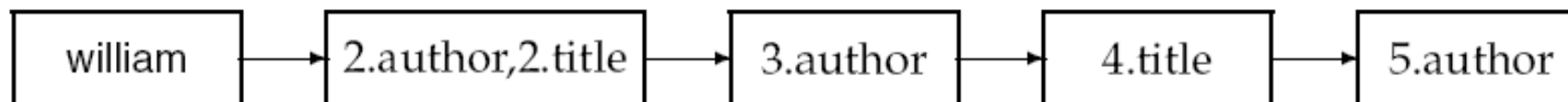
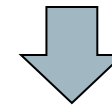
Zone

- 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



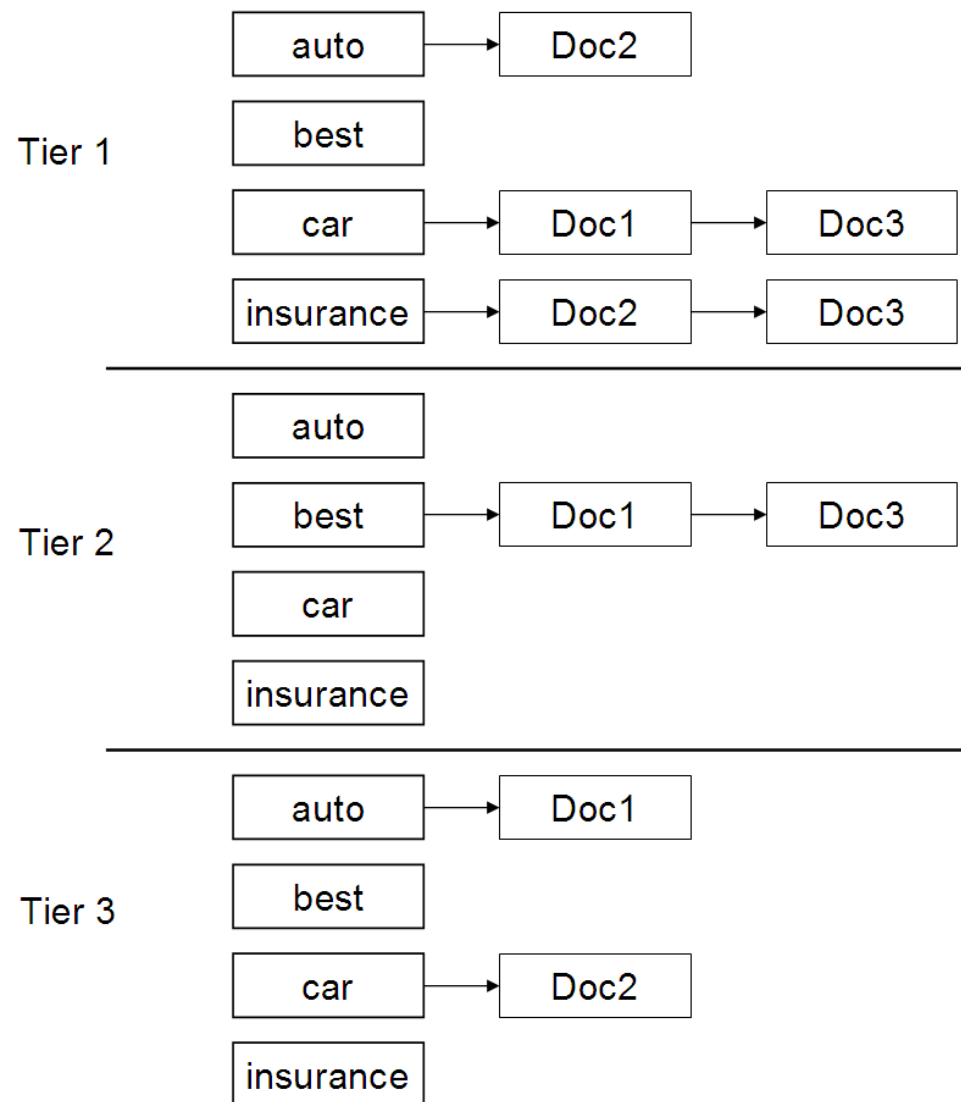
Encode zones in dictionary vs. postings.



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 tiers 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



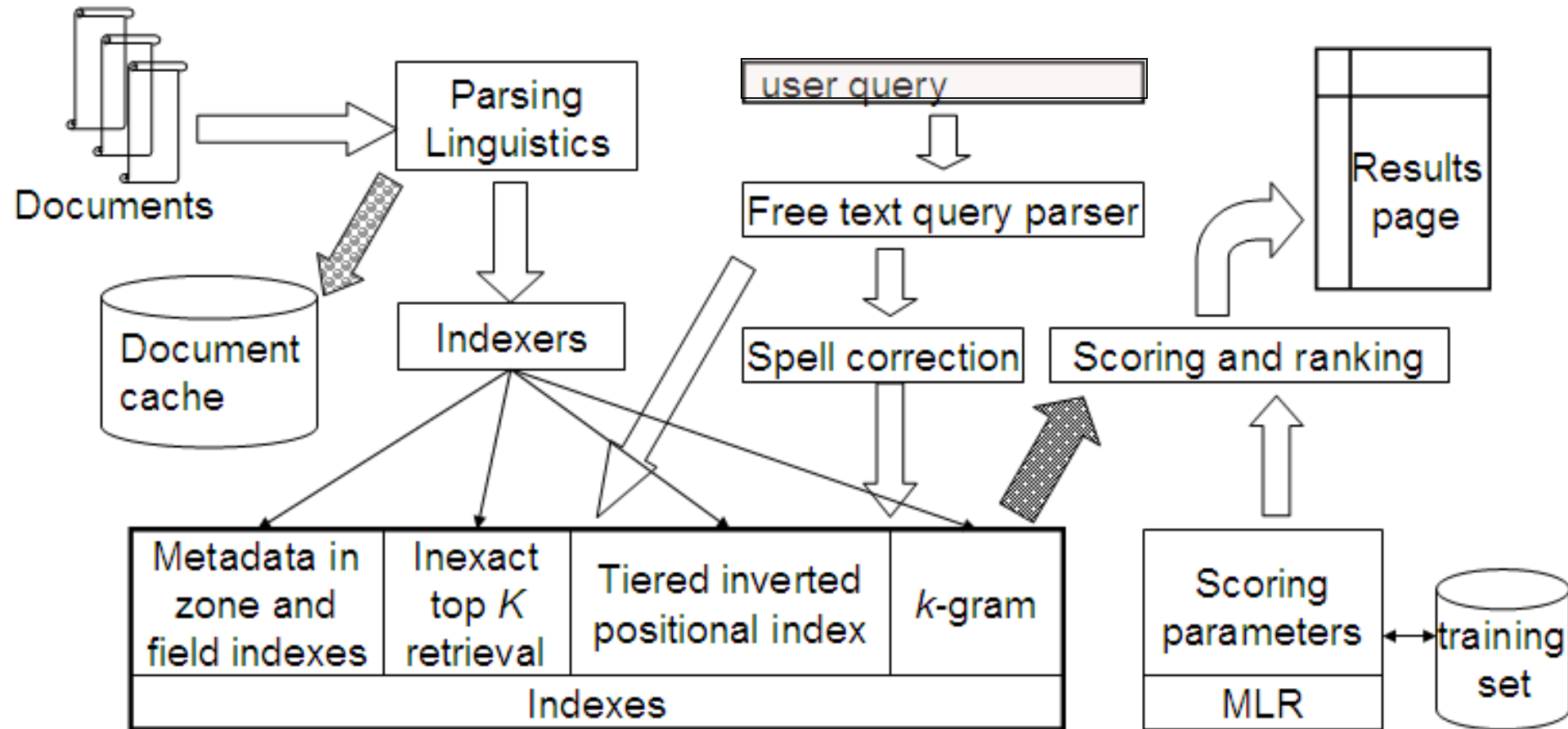
Query parsers

- 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

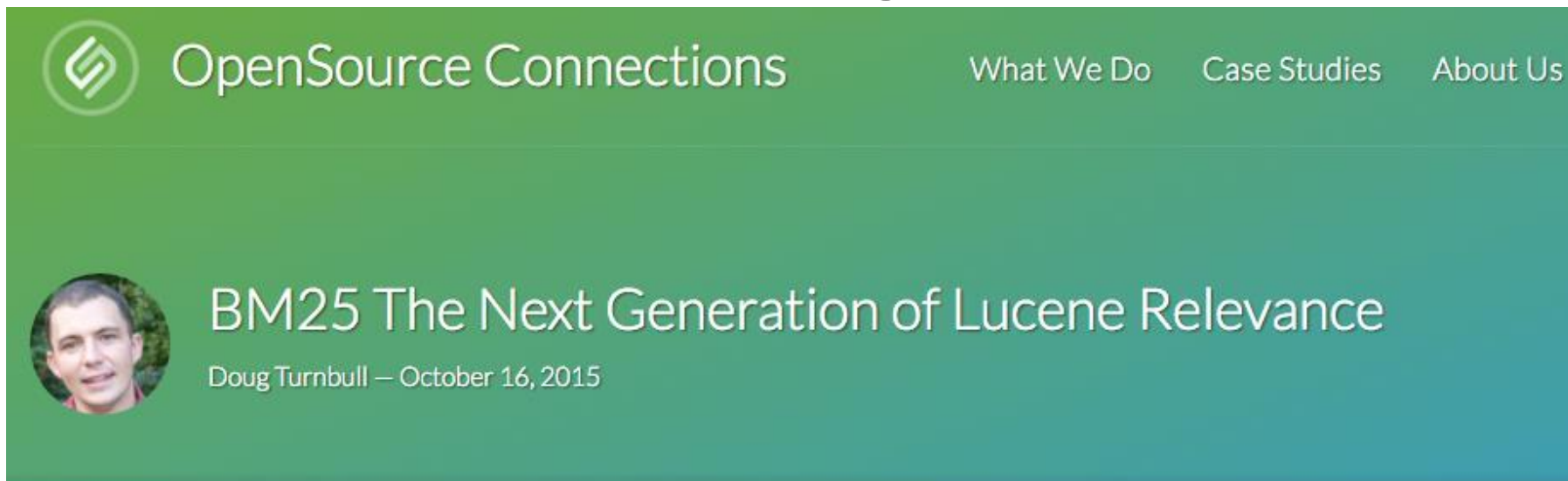
Aggregate scores

- 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



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.]

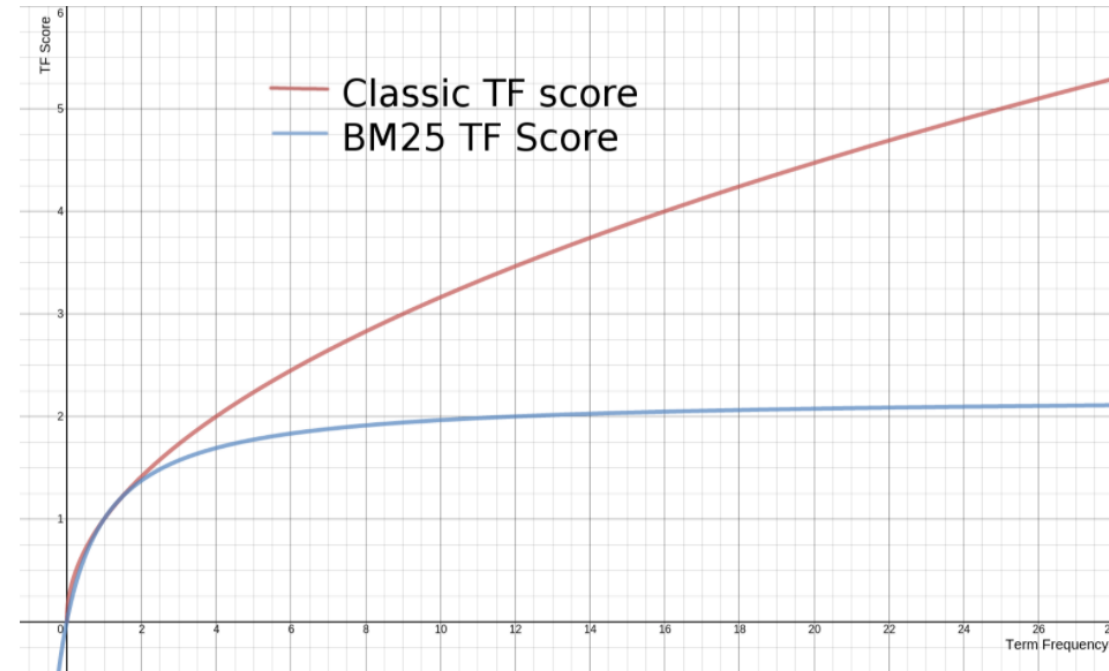
- BM25 “Best Match 25” (they had a bunch of tries!)
 - Developed in the context of the Okapi system
 - Started to be increasingly adopted by other teams during the TREC competitions
 - It works well
- Goal: be sensitive to term frequency and document length while not adding too many parameters
 - (Robertson and Zaragoza 2009; Spärck Jones et al. 2000)

“Early” version of BM25

Version 2:

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

- $(k_1 + 1)$ 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:

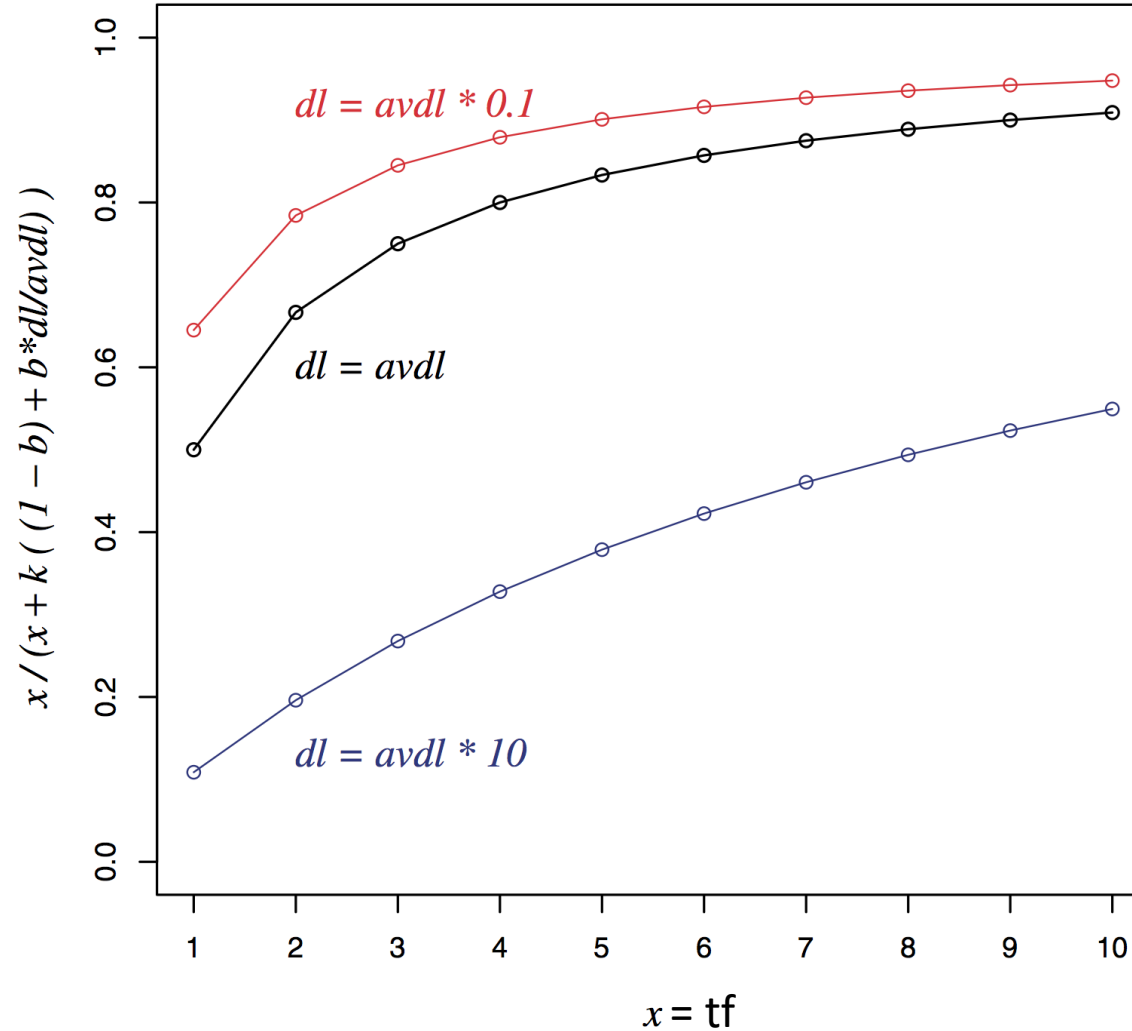
$$dl = \sum_{i \in V} tf_i$$

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

$$B = \frac{dl}{avdl} (1 - b) + b, \quad 0 \leq b \leq 1$$

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

Document length normalization



Okapi BM25

- Normalize tf using document length

$$tf_i^{\text{c}} = \frac{tf_i}{B}$$

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

- BM25 ranking function

$$RSV^{BM25} = \sum_{i=1}^q c_i^{BM25}(tf_i);$$

RSV = Retrieval Status Value

Okapi BM25

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

- 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

Statistical Language Models

Three “classic” approaches to IR

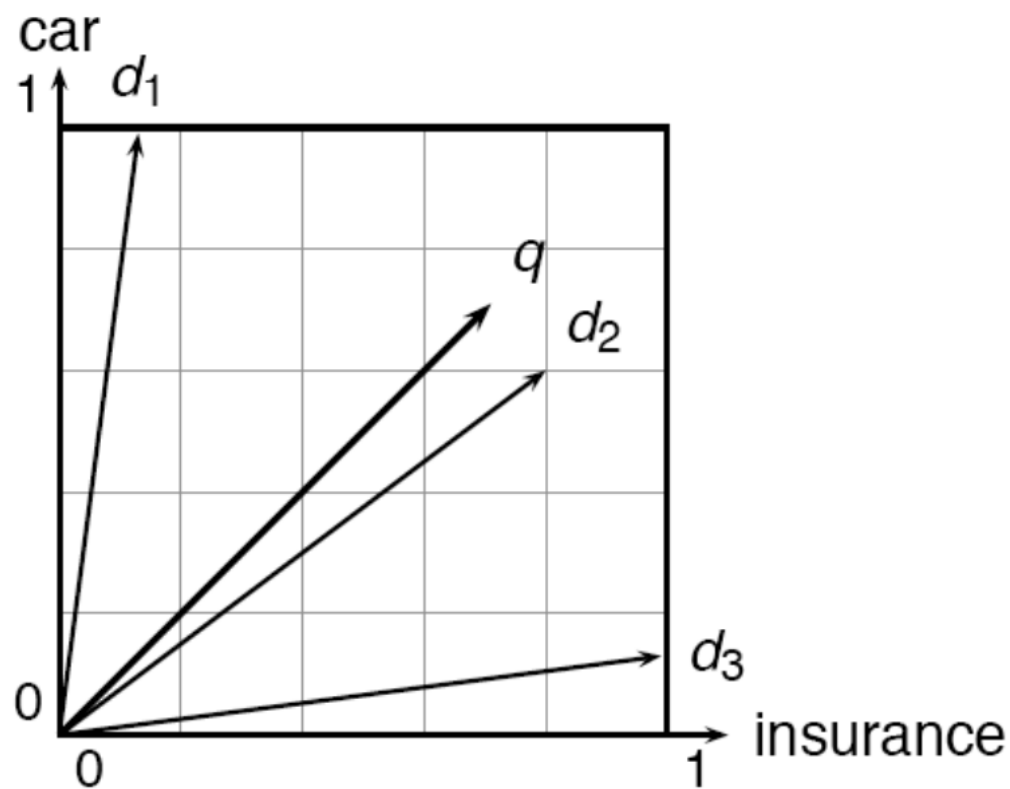
Recall: Boolean Retrieval

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Brutus AND Caesar but NOT Calpurnia

I if play contains
word, 0 otherwise

Recall: Vector Space Retrieval

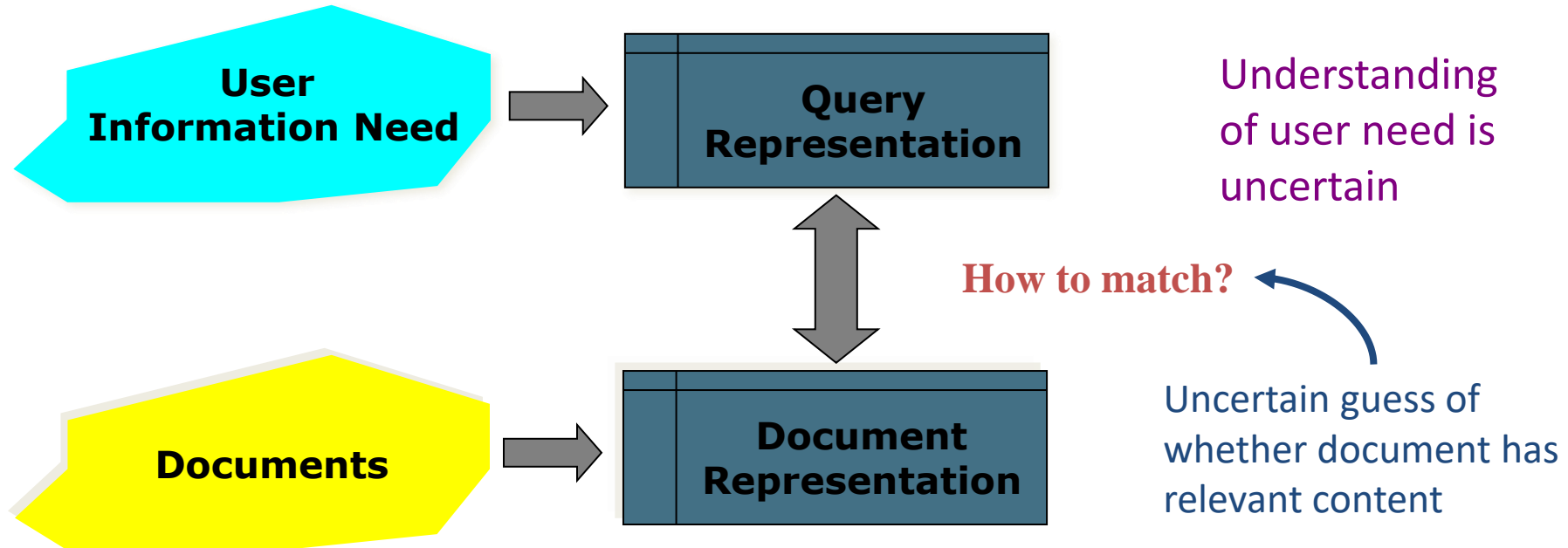


$$\text{sim}(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{\|\vec{d}_j\| \|\vec{d}_k\|} = \frac{\sum_{i=1}^n w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^n w_{i,j}^2} \sqrt{\sum_{i=1}^n w_{i,k}^2}}$$

Probabilistic IR

- Chapter 12
 - Statistical Language Models

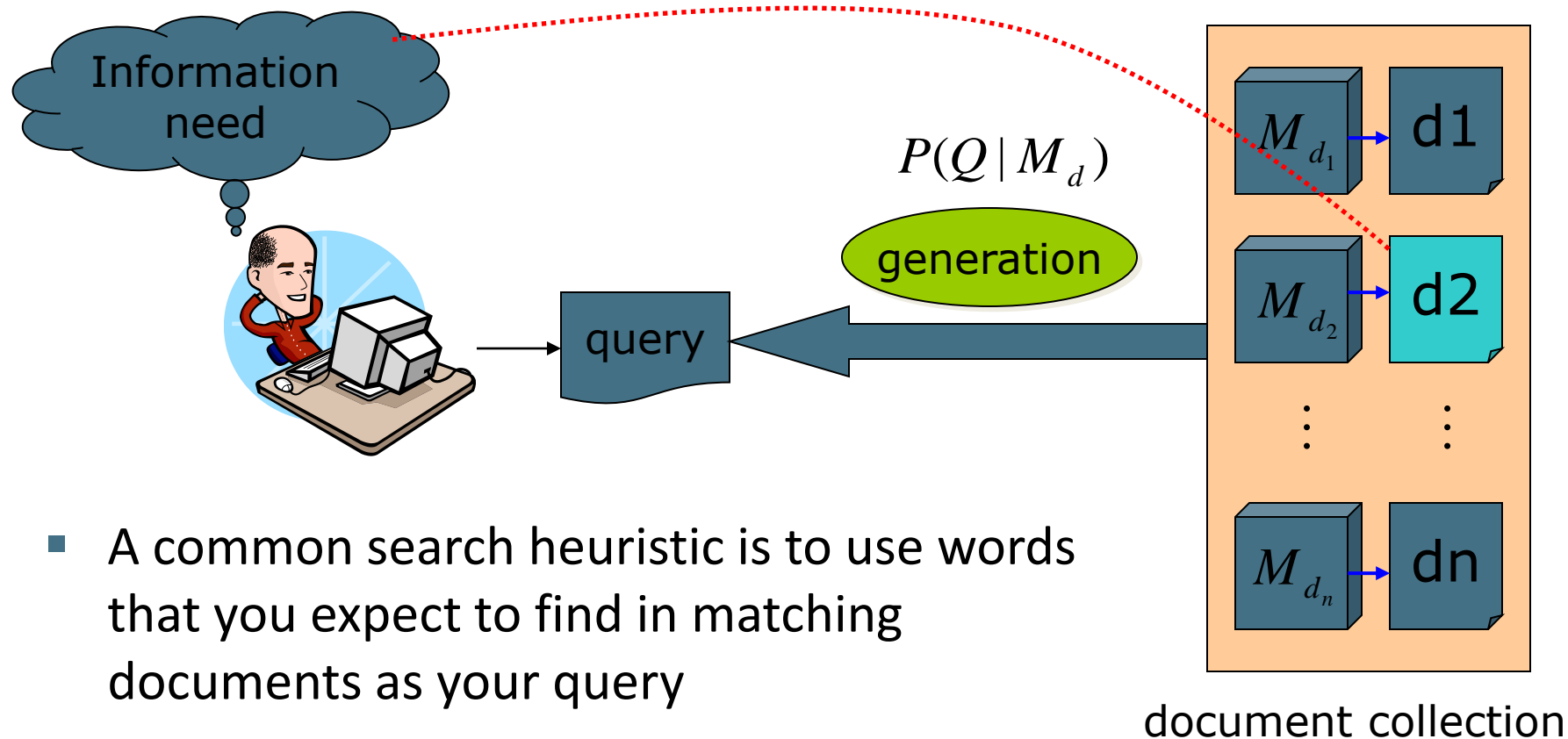
Why probabilities in IR?



In traditional IR systems, matching between each document and query is attempted in a semantically imprecise space of index terms.

Probabilities provide a principled foundation for uncertain reasoning.
Can we use probabilities to quantify our uncertainties?

IR based on Language Model (LM)



- A common search heuristic is to use words that you expect to find in matching documents as your query
- The LM approach directly exploits that idea!

What is a language model?


We can view a **finite state automaton** as a **deterministic** language model.



I wish I wish I wish I wish . . . Cannot generate: “wish I wish”
Our basic model: each document was generated by a different automaton like this except that these automata are **probabilistic**.

Stochastic Language Models

- Models *probability* of generating strings in the language

Model M						
0.2	the	the	man	likes	the	woman
0.1	a	—	—	—	—	—
0.01	man	0.2	0.01	0.02	0.2	0.01
0.01	woman					
0.03	said					
0.02	likes					
...						

multiply

$P(s \mid M) = 0.00000008$

Stochastic Language Models

- Model *probability* of generating any string

Model M1		Model M2						
0.2	the	0.2	the	the	class	pleaseth	yon	maiden
0.01	class	0.0001	class	_____	_____	_____	_____	_____
0.0001	sayst	0.03	sayst	0.2	0.01	0.0001	0.0001	0.0005
0.0001	pleaseth	0.02	pleaseth	0.2	0.0001	0.02	0.1	0.01
0.0001	yon	0.1	yon					
0.0005	maiden	0.01	maiden					
0.01	woman	0.0001	woman					

$$P(s|M2) > P(s|M1)$$

Using language models in IR

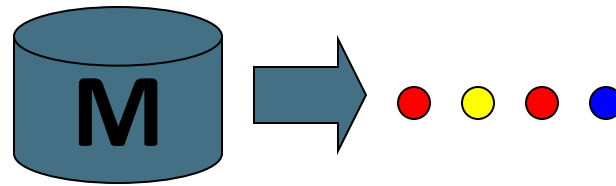
- Each document is treated as (the basis for) a language model.
- Given a query q
- Rank documents based on $P(d|q)$
- Bayes' rule

$$P(d|q) = P(q|d)P(d) / P(q)$$

- $P(q)$ is the same for all documents, so ignore
- $P(d)$ is the prior – often treated as the same for all d
 - But we can give a prior to “high-quality” documents, e.g., those with high PageRank.
- $P(q|d)$ is the probability of q given d .
- So to rank documents according to relevance to q , ranking according to $P(q|d)$ and $P(d|q)$ is equivalent.

Stochastic Language Models

- A statistical model for generating text
 - Probability distribution over a string/query in a given language



$$\begin{aligned} P(\text{red } \text{yellow } \text{red } \text{blue}) \\ = P(\text{red}) P(\text{yellow} \mid \text{red}) P(\text{red} \mid \text{red } \text{yellow}) P(\text{blue} \mid \text{red } \text{yellow } \text{red}) \end{aligned}$$

Unigram and higher-order models

$$P(\text{red} \text{ yellow} \text{ red} \text{ blue})$$

$$= P(\text{red}) P(\text{yellow} | \text{red}) P(\text{red} | \text{red} \text{ yellow}) P(\text{blue} | \text{red} \text{ yellow} \text{ red})$$

- Unigram Language Models

$$P(\text{red}) P(\text{yellow}) P(\text{red}) P(\text{blue})$$

- Bigram (generally, n -gram) Language Models

$$P(\text{red}) P(\text{yellow} | \text{red}) P(\text{red} | \text{yellow}) P(\text{blue} | \text{red})$$

- We use the unigram Language Models

Where we are

- In the LM approach to IR, we attempt to model the query generation process.
- Then we rank documents by the probability that a query would be observed as a random sample from the respective document model.
- That is, we rank according to $P(q | d)$.
- Next: how do we compute $P(q | d)$?

Retrieval based on probabilistic LM

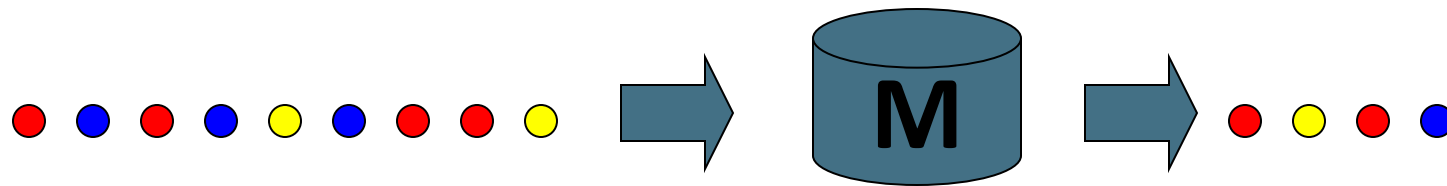
- Intuition
 - Users ...
 - Have a reasonable idea of terms that are likely to occur in documents of interest.
 - They will choose query terms that distinguish these documents from others in the collection.
- Collection statistics ...
 - Are integral parts of the language model.

The fundamental problem of LMs

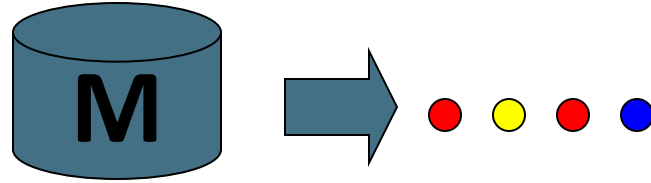
- Usually we don't know the model **M**
 - But have a sample of text representative of that model

$$P(\text{red yellow red blue} \mid M(\text{red blue red blue yellow blue red red yellow}))$$

- Estimate a language model from a sample
- Then compute the observation probability

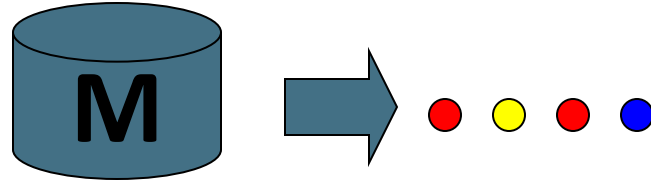


Example



- Doc 1 = “Today is a beautiful day.”
- $p(\text{today} \mid M1) =$
- $p(\text{is} \mid M1) =$
- $p(a \mid M1) =$
- $p(\text{beautiful} \mid M1) =$

Example



- Doc 2 = “Beautiful beautiful beautiful day!”
- $p(\text{today} \mid M2) =$
- $p(\text{is} \mid M2) =$
- $p(a \mid M2) =$
- $p(\text{beautiful} \mid M2) =$

Query generation probability

- Ranking formula

$$\hat{P}(q|M_d)$$

- The probability of producing the query given the language model of document d using *Maximum Likelihood Estimation* (MLE) is:
 - MLE means estimating a probability as the relative frequency. So this value makes the observed data maximally likely

$$\hat{P}(q|M_d) = \prod_{t \in q} \hat{P}_{\text{mle}}(t|M_d) = \prod_{t \in q} \frac{\text{tf}_{t,d}}{L_d}$$

Unigram assumption:
Given a particular language model,
the query terms occur independently

M_d : language model of document d

$\text{tf}_{t,d}$: raw tf of term t in document d

L_d : total number of tokens in document d

Language Models (LMs)

- Unigram LM:
 - Bag-of-words model.
 - Multinomial distributions over words.

$$P(d) = \frac{L_d!}{\text{tf}_{t_1,d}! \text{tf}_{t_2,d}! \cdots \text{tf}_{t_M,d}!} P(t_1)^{\text{tf}_{t_1,d}} P(t_2)^{\text{tf}_{t_2,d}} \cdots P(t_M)^{\text{tf}_{t_M,d}}$$

↓
multinomial coefficient, can leave out in practical calculations.

$$L_d = \sum_{1 \leq i \leq M} \text{tf}_{t_i,d}$$

The length of document d. M is the size of the vocabulary.

Query Likelihood Model

- Multinomial + Unigram:

$$P(q|M_d) = K_q \prod_{t \in V} P(t|M_d)^{\text{tf}_{t,d}}$$

$K_q = L_d! / (\text{tf}_{t_1,d}! \text{tf}_{t_2,d}! \cdots \text{tf}_{t_M,d}!)$ Multinomial coefficient for the query q .
Can be ignored.

- Retrieve based on a language model:
 - Infer a LM for each document.
 - Estimate $P(q|M_{di})$.
 - Rank the documents according to these probabilities.

Example

$$\hat{P}(q|M_d) = \prod_{t \in q} \hat{P}_{\text{mle}}(t|M_d) = \prod_{t \in q} \frac{\text{tf}_{t,d}}{L_d}$$

- Doc 1 = “Today is a beautiful day. “
- Doc 2 = “Beautiful beautiful beautiful day!”
- Query = “today beautiful”

Insufficient data

- Zero probability $\hat{P}(t|M_d) = 0$
 - May not wish to assign a probability of zero to a document that is missing one or more of the query terms
- General approach
 - A non-occurring term is possible, but no more likely than would be expected by chance in the collection.
 - If $tf_{(t,d)} = 0$, $\hat{P}(t|M_d) \leq cf_t/T$

cf_t : raw count of term t in the collection

T : raw collection size (total number of tokens in the collection)

Insufficient data

- We will use $\hat{P}(t|M_c)$ to “smooth” $P(t|d)$ away from zero.
- A simple idea that works well in practice is to use a **mixture** between the document multinomial and the collection multinomial distribution

Mixture model

$$\hat{P}(t|d) = \lambda \hat{P}_{\text{mle}}(t|M_d) + (1 - \lambda) \hat{P}_{\text{mle}}(t|M_c)$$

- Mixes the probability from the document with the general collection frequency of the word.
- High value of λ : “conjunctive-like” search – tends to retrieve documents containing all query words (suitable for short queries)
- Low value of λ : more disjunctive, suitable for long queries
- Correctly setting λ is very important for good performance.

Basic mixture model summary

- General formulation of the LM for IR

$$P(q|d) \propto \prod_{1 \leq k \leq |q|} (\lambda P(t_k|M_d) + (1 - \lambda)P(t_k|M_c))$$

individual-document model



general language model

- The user has a document in mind, and generates the query from this document.
- The equation represents the probability that the document that the user had in mind was in fact this one.

Example

- Document collection (2 documents)
 - d_1 : Xerox reports a profit but revenue is down
 - d_2 : Lucent narrows quarter loss but revenue decreases further
- Model: MLE unigram from documents; $\lambda = \frac{1}{2}$
- Query: *revenue down*
 - $P(Q|d_1) =$
 - $P(Q|d_2) =$

$$P(q|d) \propto \prod_{1 \leq k \leq |q|} (\lambda P(t_k|M_d) + (1 - \lambda)P(t_k|M_c))$$

Example

- Document collection (2 documents)
 - d_1 : Xerox reports a profit but revenue is down
 - d_2 : Lucent narrows quarter loss but revenue decreases further
- Model: MLE unigram from documents; $\lambda = \frac{1}{2}$
- Query: *revenue down*
 - $P(Q|d_1) = [(1/8 + 2/16)/2] \times [(1/8 + 1/16)/2]$
 $= 1/8 \times 3/32 = 3/256$
 - $P(Q|d_2) = [(1/8 + 2/16)/2] \times [(0 + 1/16)/2]$
 $= 1/8 \times 1/32 = 1/256$
- Ranking: $d_1 > d_2$

Exercise

- Suppose, we've got 4 documents

DocID	Document text
1	click go the shears boys click click click
2	click click
3	metal here
4	metal shears click here

- Using the mixture model with $\lambda = 0.5$, work out the per-doc probabilities for the query “click”

-
- collection model for “click” is
 - collection model for “shears” is
 - click in doc1:
 - doc2:
 - doc3:
 - doc4:

-
- collection model for “click” is $7/16$
 - collection model for “shears” is $2/16$
 - click in doc1: $0.5 * 1/2 + 0.5 * 7/16 = 0.4688$
 - doc2: 0.7188
 - doc3: 0.2188
 - doc4: 0.3438

Exercise

- Suppose, we've got 4 documents

DocID	Document text
1	click go the shears boys click click click
2	click click
3	metal here
4	metal shears click here

- For the query “click shears”, what's the ranking of the four documents?

click shears

- **Doc 4: 0.0645**
- Doc 1: 0.0586
- Doc 2: 0.0449
- Doc 3: 0.0137

Summary: LM

- LM approach assumes that documents and expressions of information problems are of the same type
- Computationally tractable, intuitively appealing

LMs vs. vector space model (1)

- LMs have some things in common with vector space models.
 - Term frequency is directed in the model.
 - But it is not scaled in LMs.
 - Probabilities are inherently “length-normalized”.
 - Cosine normalization does something similar for vector space.
 - Mixing document and collection frequencies has an effect similar to idf.
 - Terms rare in the general collection, but common in some documents will have a greater influence on the ranking.

LMs vs. vector space model (2)

- LMs vs. vector space model: differences
 - LMs: based on probability theory
 - Vector space: based on similarity, a geometric/ linear algebra notion
 - Collection frequency vs. document frequency
 - Details of term frequency, length normalization etc.