Information Retrieval & Social Web

CS 525/DS 595

Worcester Polytechnic Institute

Department of Computer Science

Instructor: Prof. Kyumin Lee

Project Team

- Jesse Gaulin, Brian Zylich, Bryan Nguon, Huyen Nguyen
- You Zhou, Jiani Gao, Zijun Xu, Han Bao
- Quyen Hoang, Khuyen Cao and Anqi Lu
- Cheng Zhu, Xi Liu, Yupeng Su, Ye Wang.
- Xiaoyu Zheng, Di You, Guocheng Yao
- Weiqing Li, Huayi Zhang, Jiaming Di, Yingnan Han
- Sarun Paisarnsrisomsuk, Fangling Zhang, Tes Shizume, Anthony Topper
- Claire Danaher, Janvi Kothari, Kavin Chandrasekaran
- Guanyi Mou, Guohui Huang, Zhenyu Mao, Yun Yue
- YaoChun Hsieh, Hoawen Zhu, Yang Tao

10 teams!

HW2

- https://canvas.wpi.edu/courses/7874/assignments/48829
- Due date is Feb 15

Previous Class...

Wild-card queries
→ Permuterm Index

Permuterm index

- For term ***hello***, index under:
  - ***hello$, ello$h, llohe, lohel, o$hell, $hello***

where $ is a special symbol.

Query = ***hel*o***

X=*hel*, Y=*o*

Lookup ***o$hel****

Previous Class…

TF and IDF

Previous Class…

TF and IDF

tf-idf weighting

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

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)

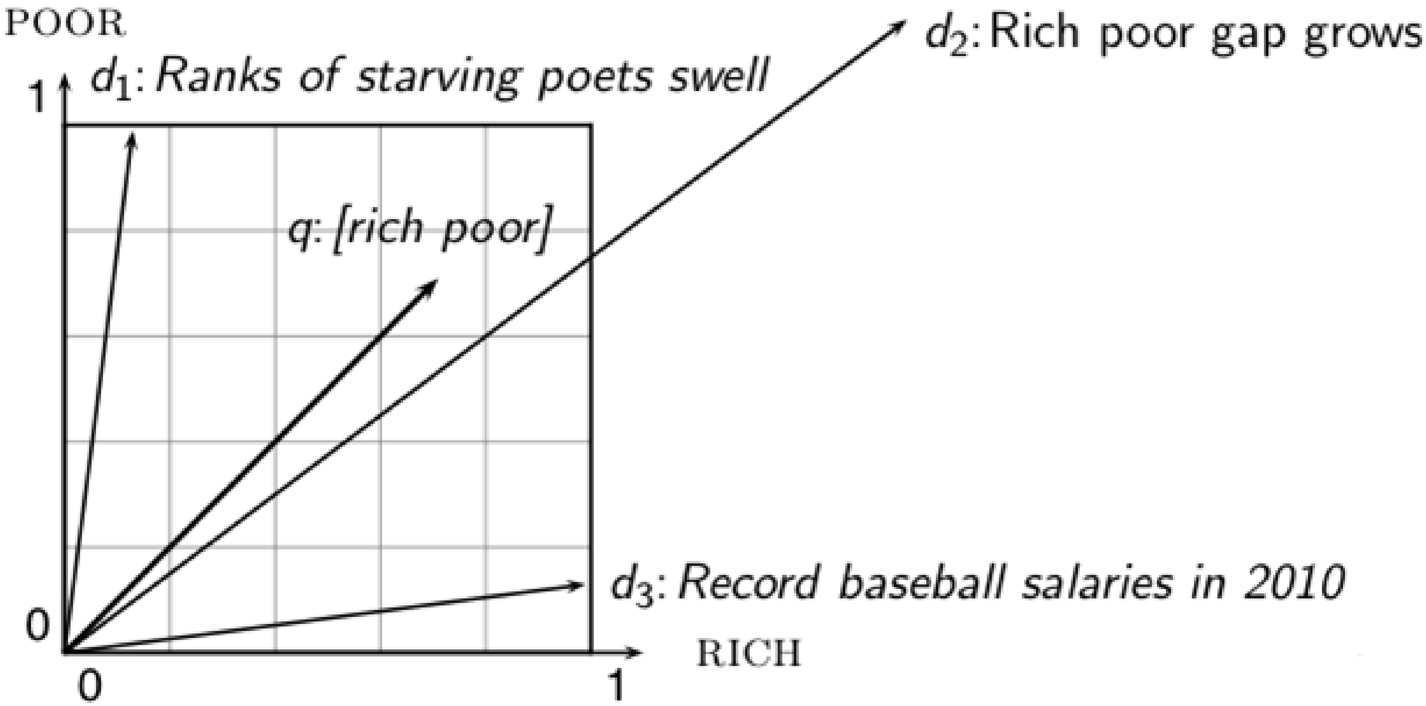
The Vector Space Model

Binary → 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 $\in R^{|V|}$.

Why distance is a bad idea



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

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°]

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

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

- 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 $\vec{d}$.

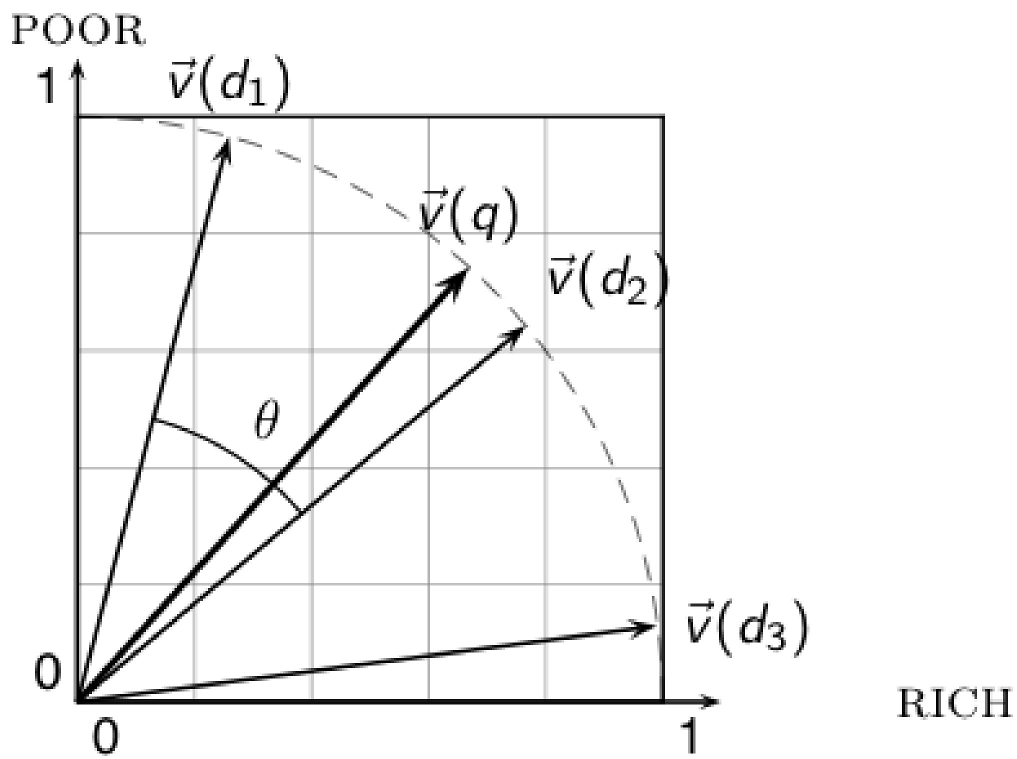
Cosine for normalized vectors

- 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?
SaS: Sense and Sensibility
PaP: Pride and Prejudice
WH: Wuthering Heights

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) ≈

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)	$\frac{1}{\sqrt{w_1^2 + w_2^2 + \ldots + w_M^2}}$
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)	$1/\mathit{CharLength}^{\alpha}$, $\alpha < 1$
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

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

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

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

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

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

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 $w_{t,q}$ and fetch postings list for t
    for each pair$(d, \mathrm{tf}_{t,d})$ 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 $Scores[]$

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*

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 $K < |A| << N$
  - *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 <u>pruning</u> non-contenders
- Will look at several schemes following this approach

Index elimination

- 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 $\rightarrow$ 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

So far…

- Talked about how to speed up computing the relevancy between query and docs

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 or diggs
  - (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

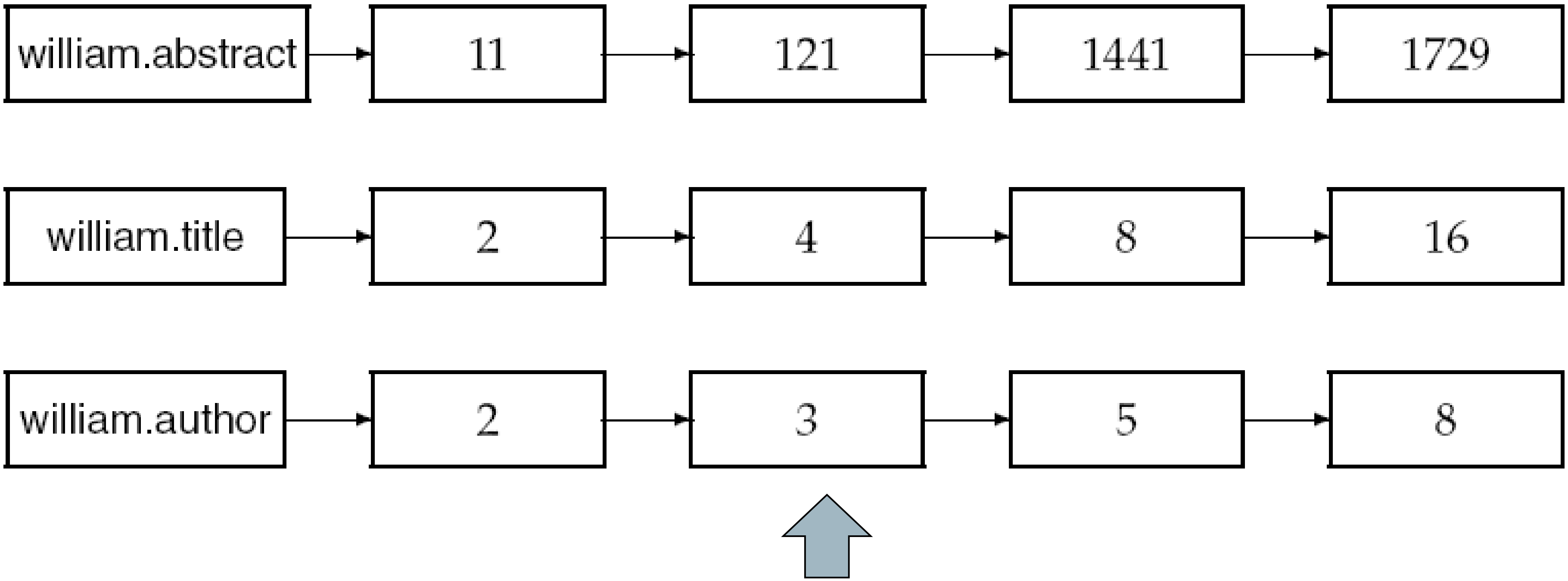
- Consider a simple total score combining cosine relevance and authority
- net-score(q,d) = $g(d) + $ cosine(q,d)
  - Can use some other linear combination
- Now we seek the top K docs by net score

In Addition….

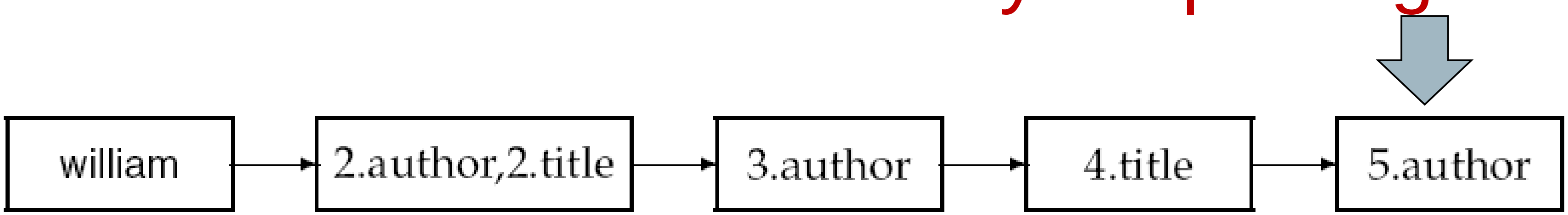
Zone

- 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



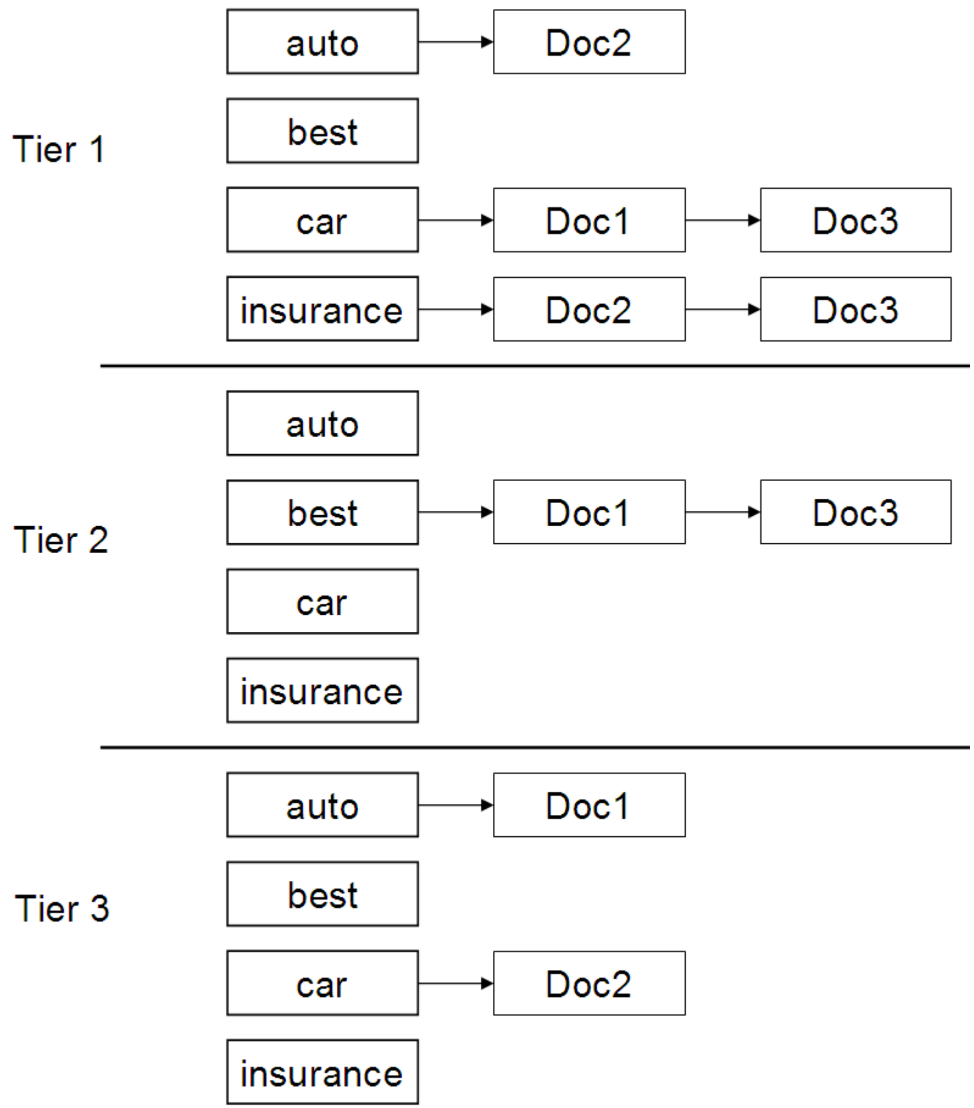
Encode zones in dictionary vs. postings.



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



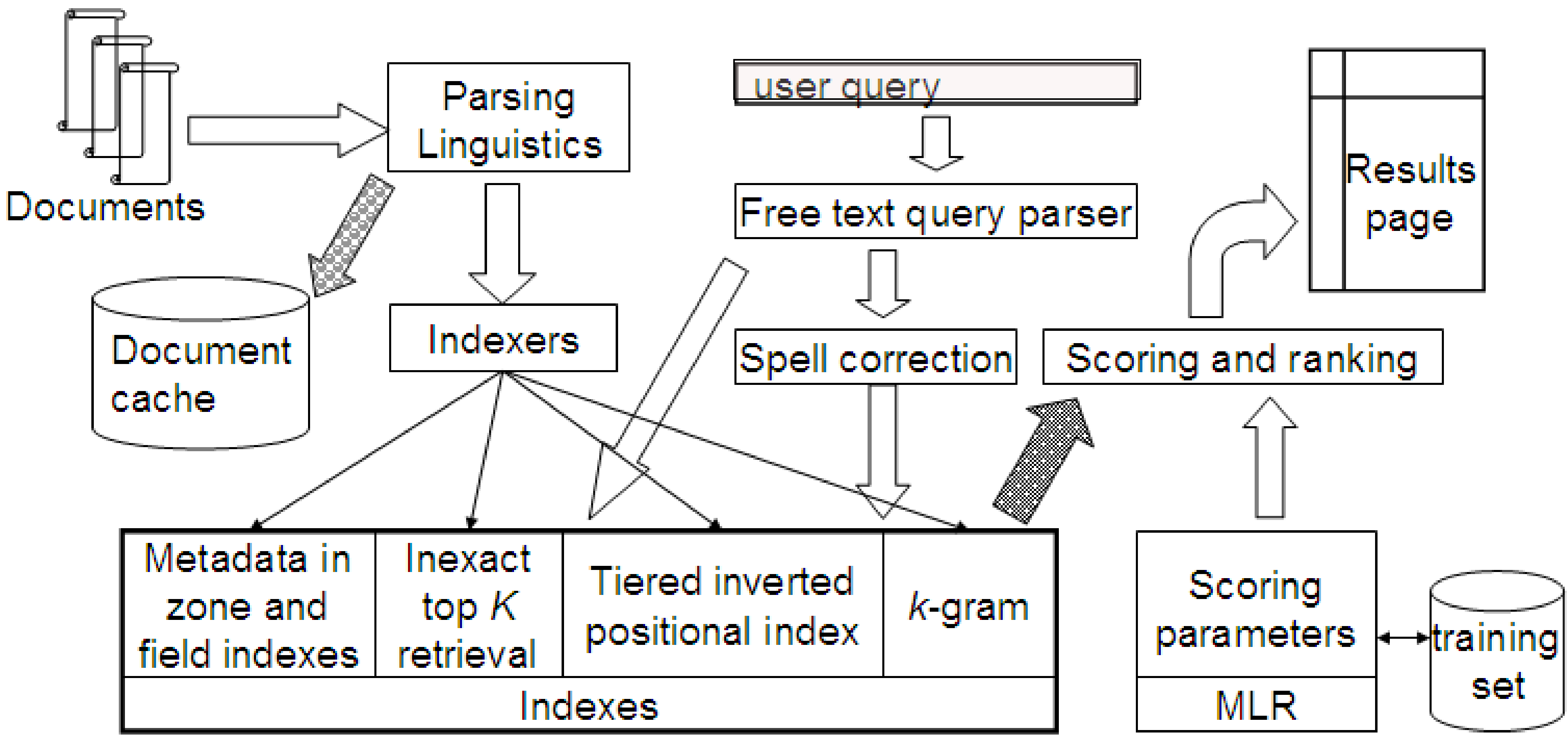
Query parsers

- 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 <u>query parser</u>

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
  - Learning to Rank

Putting it all together



Statistical Language Models

Three “classic” approaches to IR

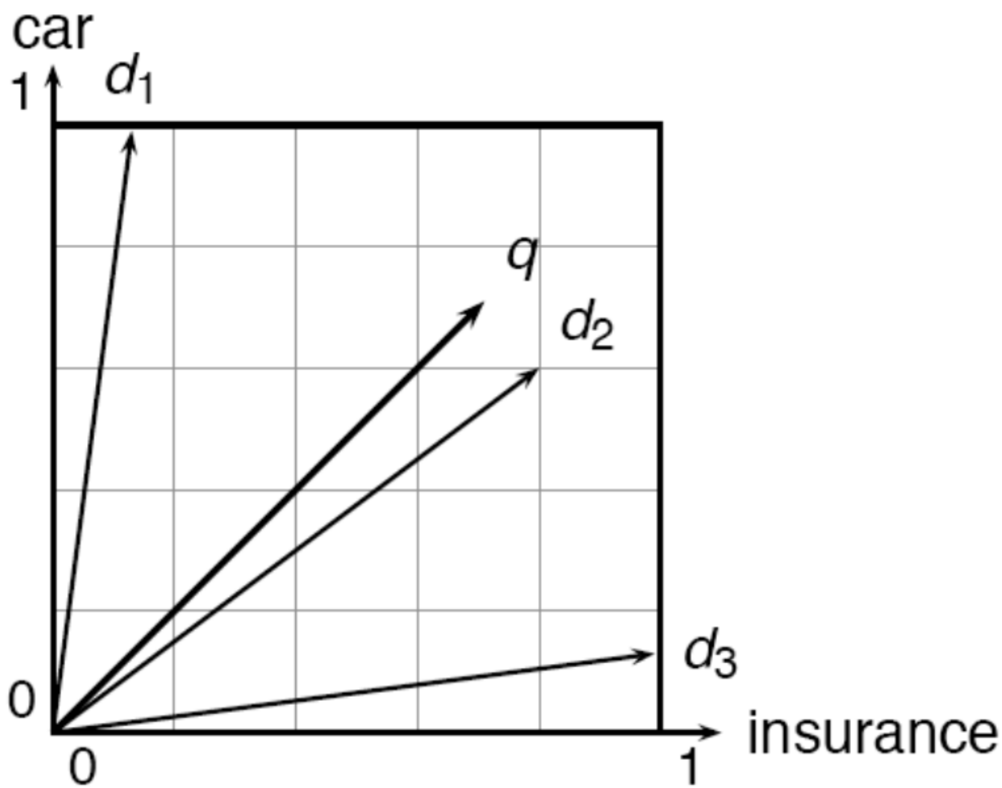
Recall: Boolean Retrieval

	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***

1 if play contains word, 0 otherwise

Recall: Vector Space Retrieval

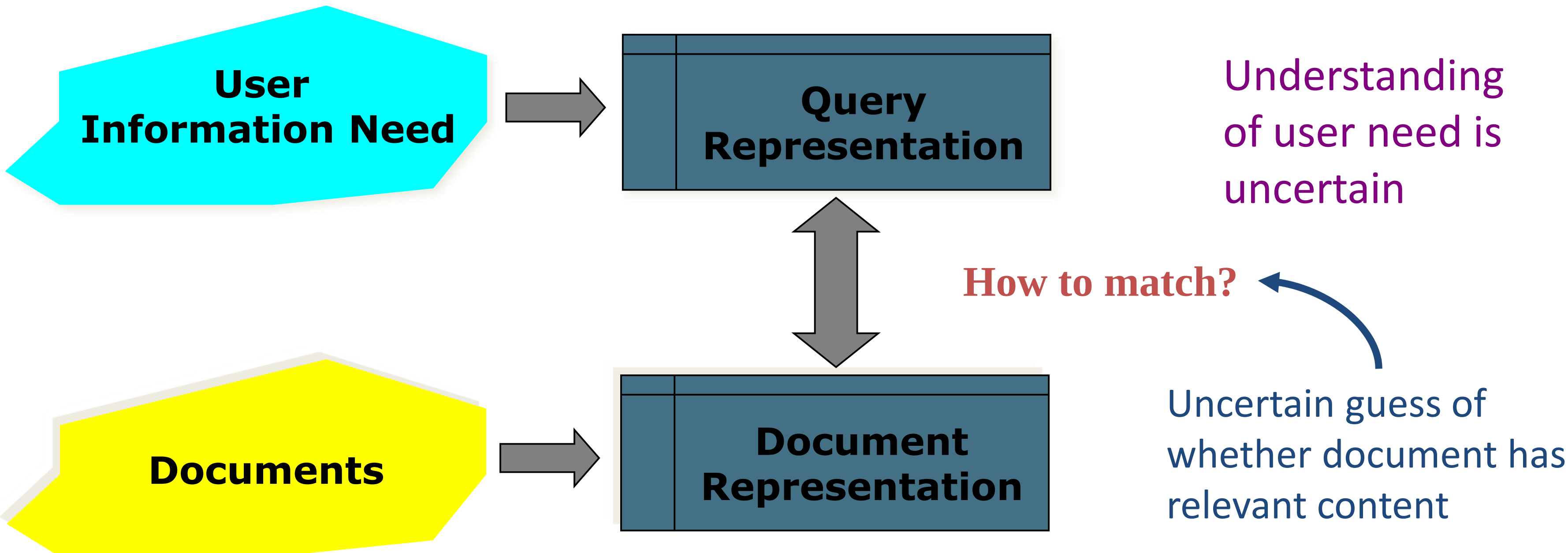


$$sim(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{\left|\vec{d}_j\right|\left|\vec{d}_k\right|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2}\sqrt{\sum_{i=1}^{n} w_{i,k}^2}}$$

Probabilistic IR

- Chapter 12
  - Statistical Language Models

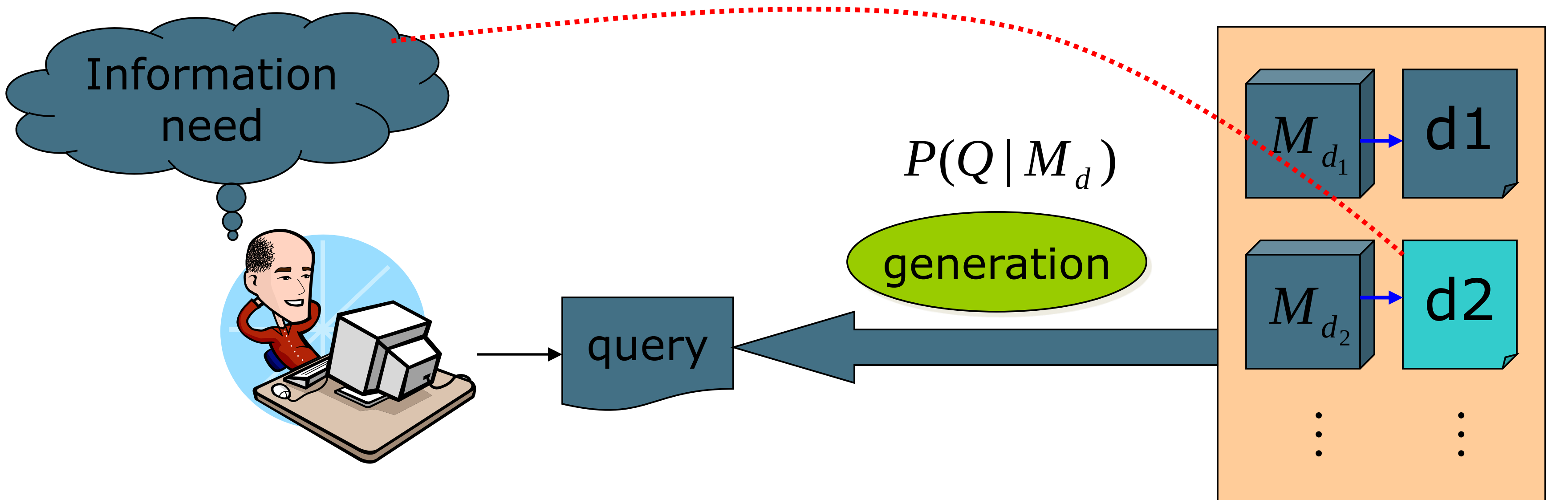
Why probabilities in IR?



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?

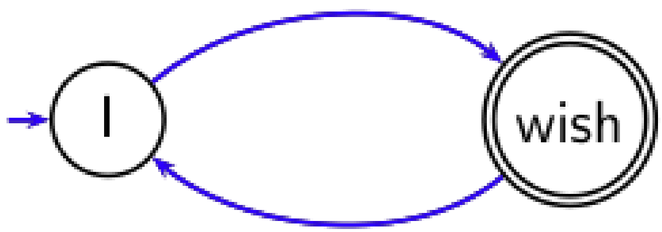
IR based on Language Model (LM)



- 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”

or “I wish I”. 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
0.1	a
0.01	man
0.01	woman
0.03	said
0.02	likes
…	

the	man	likes	the	woman
0.2	0.01	0.02	0.2	0.01

multiply

P(s | M) = 0.00000008