CS60092: Information Retrieval

Ranking and Vector Space Models

Prof. Sourangshu Bhattacharya

CSE, IIT Kharagpur

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 the collection.

- Also good for applications: Applications can easily consume 1000s of results.

Not good for the majority of users.

- Most users incapable 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.

Problem with Boolean search: feast or famine

Boolean queries often result in either too few (=0) or too many (1000s) results.

Query 1: "*standard user dlink 650*" → 200,000 hits

Query 2: "*standard user dlink 650 no card found*": 0 hits

It takes a lot of skill to come up with a query that produces a manageable number of hits.

AND gives too few; OR gives too many

Ranked retrieval models

Rather than a set of documents satisfying a query expression, in ranked retrieval, the system returns an ordering over the (top) documents in the collection for a query

Free text queries: Rather than a query language of operators and expressions, the user's query is just one or more words in a human language

In principle, there are two separate choices here, but in practice, ranked retrieval has normally been associated with free text queries and vice versa

Feast or famine: not a problem in ranked retrieval

When a system produces a ranked result set, large result sets are not an issue

- Indeed, the size of the result set is not an issue
- We just show the top *k* (≈ 10) results
- We don't overwhelm the user

Premise: the ranking algorithm works

Scoring as the basis of ranked retrieval

We wish to return in order the documents most likely to be useful to the searcher

How can we rank-order the documents in the collection with respect to a query?

Assign a score – say in [0, 1] – to each document

This score measures how well document and query "match".

Take 1: Jaccard coefficient

A common measure of overlap of two sets A and B

jaccard$(A,B) = |A \cap B| \,/\, |A \cup B|$

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: Scoring example

What is the query-document match score that the Jaccard coefficient computes for each of the two documents below?

Query: *ides of march*

Document 1: *caesar died in march*

Document 2: *the long march*

Issues with Jaccard for scoring

It doesn't consider *term frequency* (how many times a term occurs in a document)

Rare terms in a collection are more informative than frequent terms. Jaccard doesn't consider this information

We need a more sophisticated way of normalizing for length

Query-document matching scores

We need a way of assigning a score to 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 (should be)

We will look at a number of alternatives for this.

Recall (Lecture 2): Binary term-document incidence matrix

	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

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

Term-document count matrices

Consider the number of occurrences of a term in a document:

Each document is a count vector in $\mathbb{N}^{v}$: a column below

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Bag of words model

Vector representation doesn't consider the ordering of words in a document

John is quicker than Mary and *Mary is quicker than John* have the same vectors

This is called the bag of words model.

In a sense, this is a step back: The positional index was able to distinguish these two documents.

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*.

Note: Frequency means count in IR

We want to use tf when computing query-document match scores. But how?

Raw term frequency is not what we want:

A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.

But not 10 times more relevant.

Relevance does not increase proportionally with term frequency.

Log-frequency weighting

The log frequency weight of term t in d is

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

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*:

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

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

Rare terms are more informative

Rare terms are more informative than frequent terms

Recall stop words

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 to the query *arachnocentric*

→ We want a high weight for rare terms like *arachnocentric*.

Collection vs. Document frequency

Collection frequency of *t* is the number of occurrences of *t* in the collection
Document frequency of t is the number of documents in which t occurs
Example:

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

Which word is for better search (gets higher weight)

idf weight

df_t is the document frequency of *t*: the number of documents that contain *t*

- df_t is an inverse measure of the informativeness of *t*
- $df_t \leq N$

We define the idf (inverse document frequency) of *t* by

We use log (N/df_t) instead of N/df_t to "dampen" the effect of idf.

$$idf_t = \log_{10}(N/df_t)$$

idf example, suppose *N* = 1 million

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

$$\text{idf}_t = \log_{10}(N/\text{df}_t)$$

There is one idf value for each term *t* in a collection.

Effect of idf on ranking

Does idf have an effect on ranking for one-term queries, like
- iPhone

idf has no effect on ranking one term queries
- idf affects the ranking of documents for queries with at least two terms

For the query capricious person, idf weighting makes occurrences of capricious count for much more in the final document ranking than occurrences of person.

tf-idf weighting

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

$$w_{t,d} = \log(1 + \mathrm{tf}_{t,d}) \times \log_{10}(N / \mathrm{df}_t)$$

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

Increases with the number of occurrences within a document

Increases with the rarity of the term in the collection

Score for a document given a query

$$\text{Score}(q,d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$$

There are many variants

- How “tf” is computed (with/without logs)
- Whether the terms in the query are also weighted
- …

Binary → count → weight matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

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

Documents as vectors

So we have a |V|-dimensional 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 a web search engine

These are very sparse vectors - most entries are zero.

Queries as vectors

Key idea 1: Do the same for queries: represent them as vectors in the space

Key idea 2: Rank documents according to their proximity to the query in this space

proximity = similarity of vectors

proximity ≈ inverse of distance

Formalizing vector space proximity

First cut: distance between two points

(= distance between the end points of the two vectors)

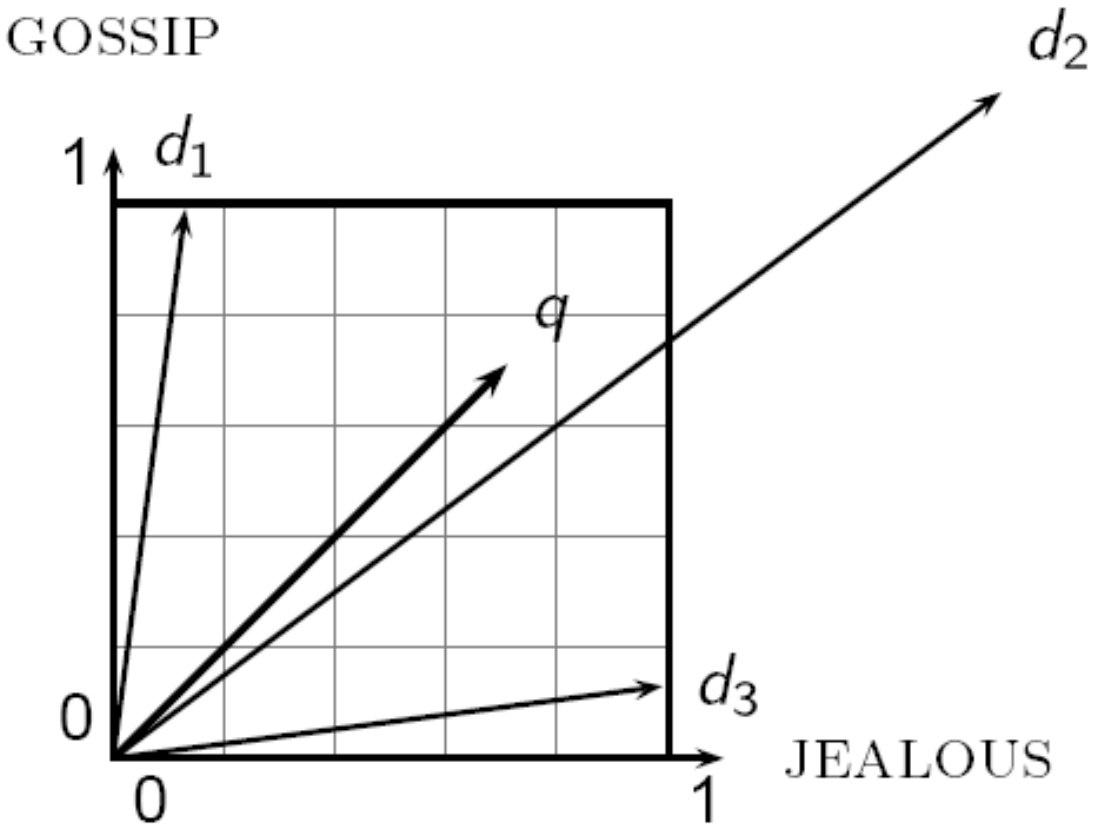
Euclidean distance?

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 between $\vec{q}$ and $\vec{d_2}$ is large even though the distribution of terms in the query $\vec{q}$ and the distribution of terms in the document $\vec{d_2}$ are very similar.



Use angle instead of distance

Thought experiment: take a document *d* and append it to itself. Call this document *d*′.

"Semantically" d and d′ have the same content

The Euclidean distance between the two documents can be quite large

The angle between the two documents is 0, corresponding to maximal similarity.

Key idea: Rank documents according to angle with query.

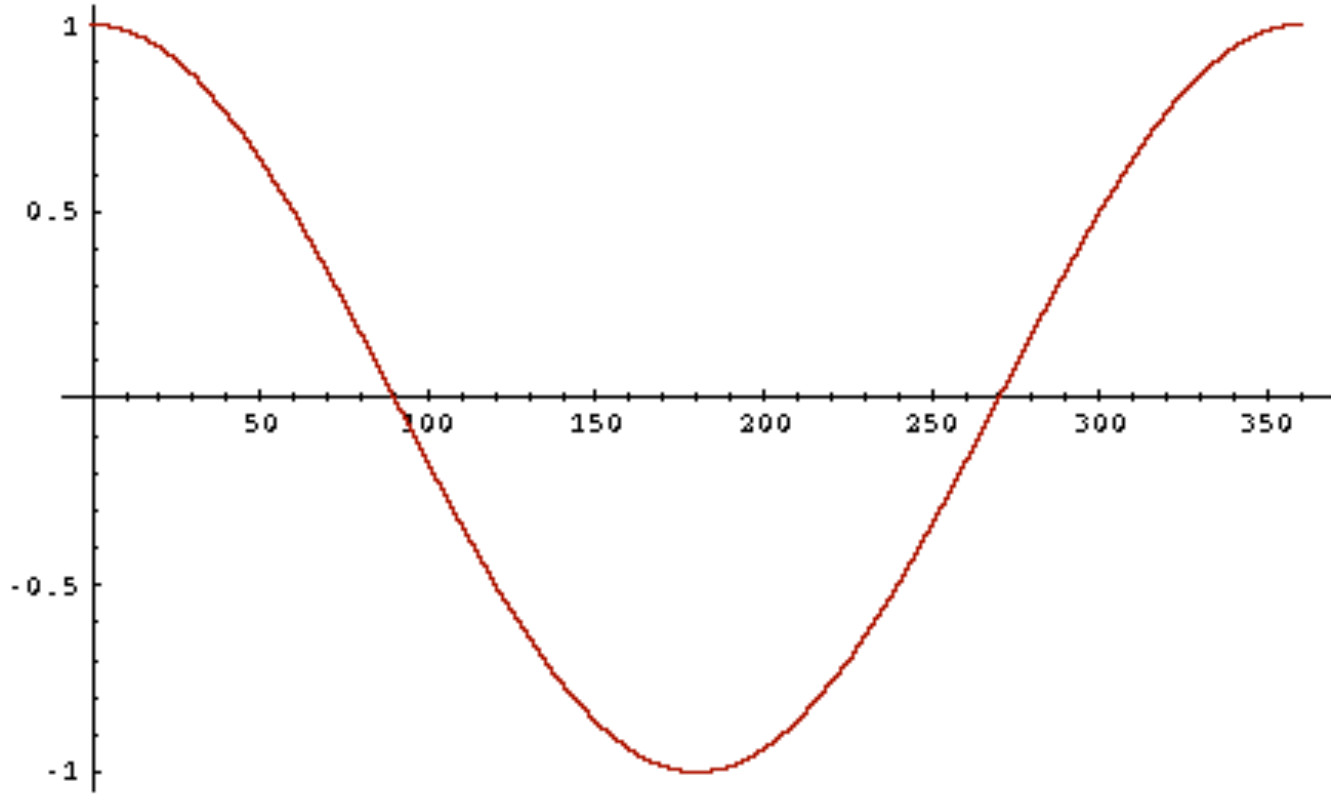
From angles to cosines

The following two notions are equivalent.

- Rank documents in decreasing order of the angle between query and document
- Rank documents in increasing order of cosine(query,document)

Cosine is a monotonically decreasing function for the interval [0^o, 180^o]

From angles to cosines



But how should we be computing cosines?

Length normalization

A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm:

$$\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$$

Dividing a vector by its L_2 norm makes it a unit (length) vector (on surface of unit hypersphere)

Effect on the two documents d and d′ (d appended to itself) from earlier slide: they have identical vectors after length-normalization.

Long and short documents now have comparable weights

cosine(query,document)

Dot product

Unit vectors

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

q_i is the weight of term i in the query
d_i is the weight of term i in the document

$\cos(\vec{q},\vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$ … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

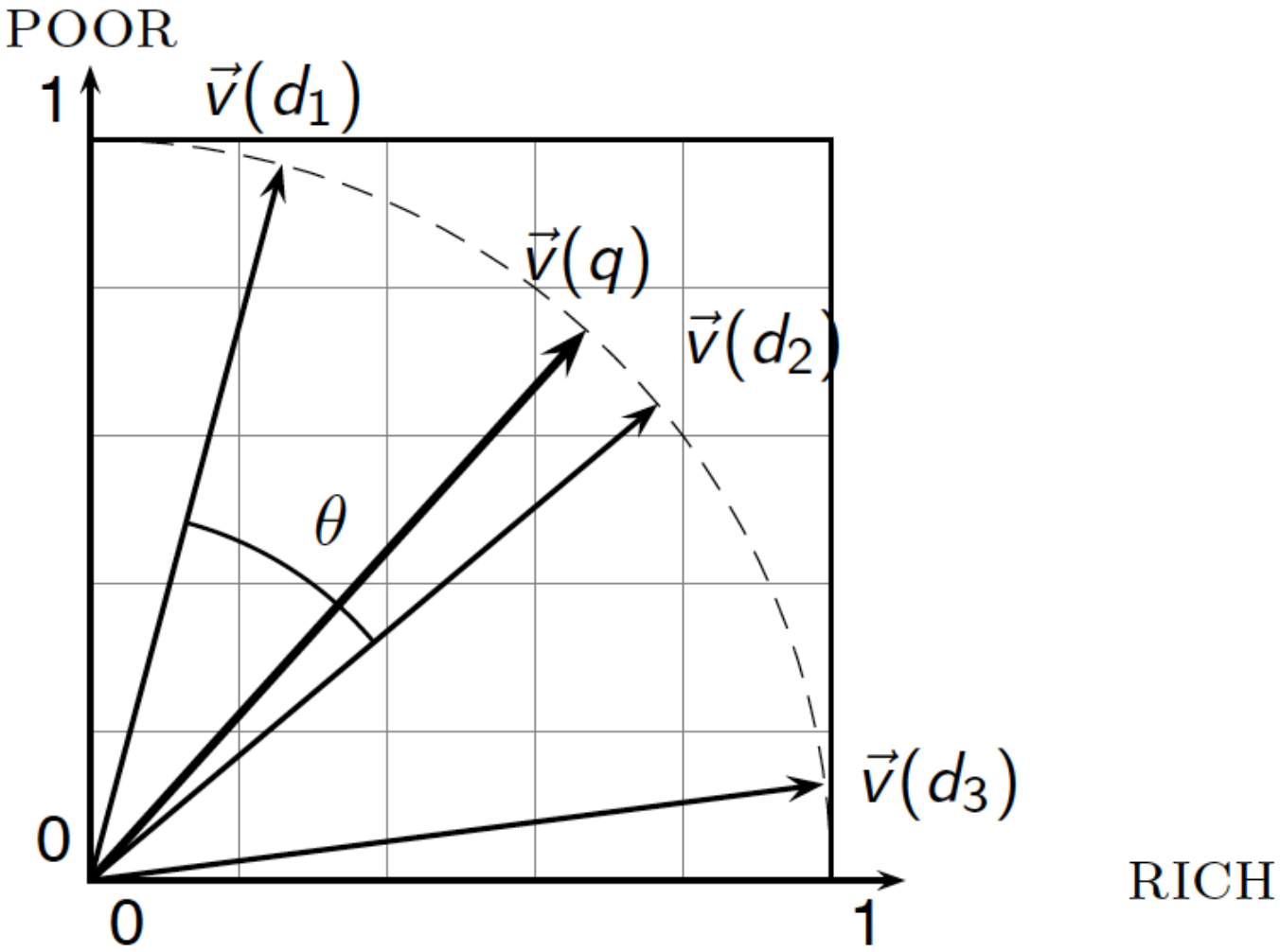
Cosine for length-normalized vectors

For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):

$$\cos(\vec{q},\vec{d}) = \vec{q} \bullet \vec{d} = \sum_{i=1}^{|V|} q_i d_i$$

for q, d length-normalized.

Cosine similarity illustrated



Cosine similarity amongst 3 documents

How similar are the novels
SaS: *Sense and Sensibility*
PaP: *Pride and Prejudice*, and
WH: *Wuthering Heights*?

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Term frequencies (counts)

Note: To simplify this example, we don't do idf weighting.

3 documents example contd.

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

dot(SaS,PaP) ≈ 12.1
dot(SaS,WH) ≈ 13.4
dot(PaP,WH) ≈ 10.1

cos(SaS,PaP) ≈ 0.94
cos(SaS,WH) ≈ 0.79
cos(PaP,WH) ≈ 0.69

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} × 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[]
```

Computing cosine scores

Previous algorithm scores term-at-a-time (TAAT)

Algorithm can be adapted to scoring document-at-a-time (DAAT)

Storing $w_{t,d}$ in each posting could be expensive

- …because we'd have to store a floating point number
- For tf-idf scoring, it suffices to store $tf_{t,d}$ in the posting and idf_t in the head of the postings list

Extracting the top K items can be done with a priority queue (e.g., a heap)

tf-idf weighting has many variants

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)	$1/CharLength^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

Weighting may differ in queries vs documents

Many search engines allow for different weightings for queries vs. documents

SMART Notation: denotes the combination in use in an engine, with the notation *ddd.qqq,* using the acronyms from the previous table

A very standard weighting scheme is: lnc.ltc

Document: logarithmic tf (l as first character), no idf and cosine normalization

Query: logarithmic tf (l in leftmost column), idf (t in second column), cosine normalization …

tf-idf example: lnc.ltc

Document: *car insurance auto insurance*
Query: *best car insurance*

Term	Query						Document				Prod
	tf-raw	tf-wt	df	idf	wt	n'lize	tf-raw	tf-wt	wt	n'lize	
auto	0	0	5000	2.3	0	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0.34	0	0	0	0	0
car	1	1	10000	2.0	2.0	0.52	1	1	1	0.52	0.27
insurance	1	1	1000	3.0	3.0	0.78	2	1.3	1.3	0.68	0.53

Doc length $= \sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92$

Score = 0+0+0.27+0.53 = 0.8

Summary – vector space ranking

Represent the query as a weighted tf-idf vector

Represent each document as a weighted tf-idf vector

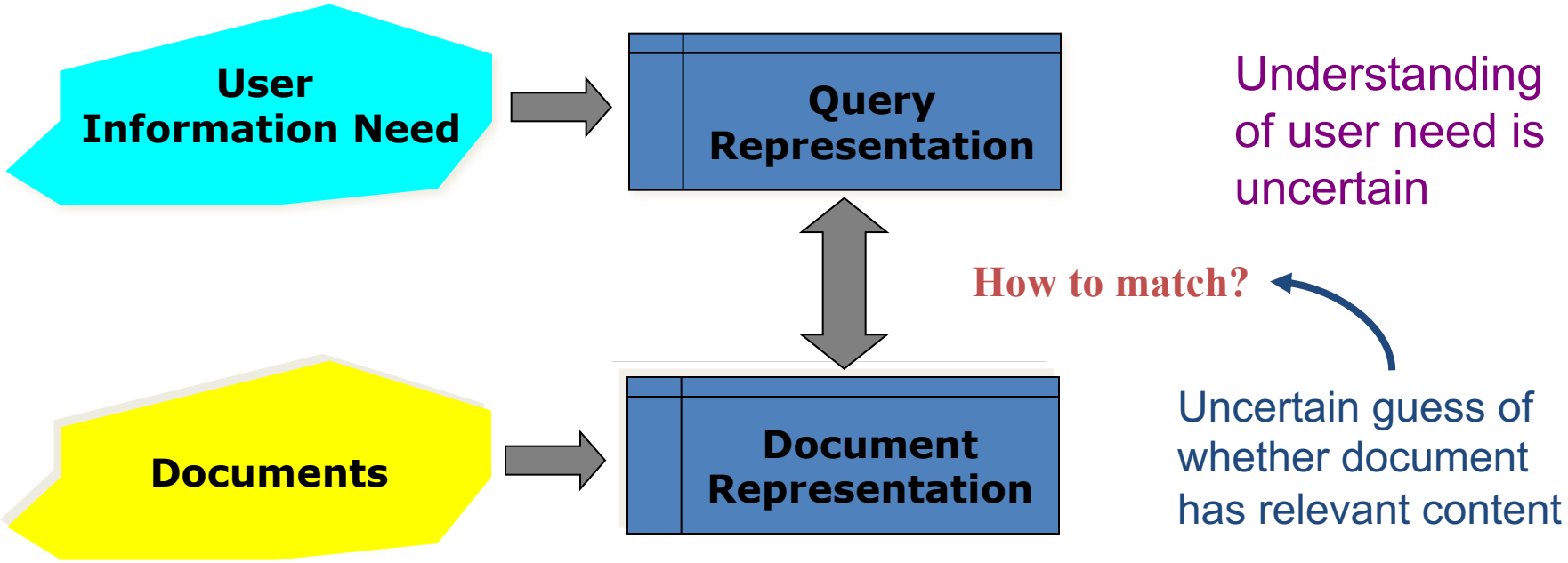
Compute the cosine similarity score for the query vector and each document vector

Rank documents with respect to the query by score

Return the top *K* (e.g., *K* = 10) to the user

Probabilistic IR

2. 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 search uncertainties?

The document ranking problem

We have a collection of documents

User issues a query

A list of documents needs to be returned

Ranking method is the core of modern IR systems:

In what order do we present documents to the user?

We want the "best" document to be first, second best second, etc.

Idea: Rank by probability of relevance of the document w.r.t. information need

$P(R=1|\text{document}_i, \text{query})$

The Probability Ranking Principle (PRP)

"If a reference retrieval system's response to each request is a ranking of the documents in the collection in order of decreasing probability of relevance to the user who submitted the request, where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose, the overall effectiveness of the system to its user will be the best that is obtainable on the basis of those data."

[1960s/1970s] S. Robertson, W.S. Cooper, M.E. Maron; van Rijsbergen (1979:113); Manning & Schütze (1999:538)

Recall a few probability basics

For events *A* and *B:*

$$p(A,B) = p(A \cap B) = p(A \mid B)p(B) = p(B \mid A)p(A)$$

Bayes' Rule

$$p(A \mid B) = \frac{p(B \mid A)p(A)}{p(B)} = \frac{p(B \mid A)p(A)}{\sum_{X=A,\overline{A}} p(B \mid X)p(X)}$$

Prior

Odds:

Posterior

$$O(A) = \frac{p(A)}{p(\overline{A})} = \frac{p(A)}{1 - p(A)}$$

The Probability Ranking Principle (PRP)

Let *x* represent a document in the collection.
Let *R* represent **relevance** of a document w.r.t. given (fixed) query and let **R=1** represent relevant and **R=0** not relevant**.**

Need to find p(*R*=1|*x*) – probability that a document *x* is **relevant.**

$$p(R=1|x)=\frac{p(x|R=1)p(R=1)}{p(x)}$$

$$p(R=0|x)=\frac{p(x|R=0)p(R=0)}{p(x)}$$

$$p(R=0|x)+p(R=1|x)=1$$

p(*R=1*),p(*R=0*) - prior probability of retrieving a relevant or non-relevant document at random

p(*x*|*R*=1), p(*x*|*R*=0) - probability that if a relevant (not relevant) document is retrieved, it is *x*.

Probabilistic Retrieval Strategy

First, estimate how each term contributes to relevance

- How do other things like term frequency and document length influence your judgments about document relevance?
  - Not at all in BIM
  - A more nuanced answer is given by BM25

Combine to find document relevance probability

Order documents by decreasing probability

Theorem: Using the PRP is optimal, in that it minimizes the loss (Bayes risk) under 1/0 loss

- Provable if all probabilities correct, etc. [e.g., Ripley 1996]

3. Binary Independence Model

Traditionally used in conjunction with PRP

"Binary" = Boolean: documents are represented as binary incidence vectors of terms (cf. IIR Chapter 1):

$$\vec{x} = (x_1, \ldots, x_n)$$

$x_i = 1$ iff term *i* is present in document *x*.

"Independence": terms occur in documents independently

Different documents can be modeled as the same vector

Binary Independence Model

Queries: binary term incidence vectors

Given query ***q***,

- for each document ***d*** need to compute ***p(R|q,d)***
- replace with computing ***p(R|q,x)*** where ***x*** is binary term incidence vector representing ***d***
- Interested only in ranking

Will use odds and Bayes' Rule:

$$O(R \mid q,\vec{x}) = \frac{p(R=1 \mid q,\vec{x})}{p(R=0 \mid q,\vec{x})} = \frac{\dfrac{p(R=1 \mid q)p(\vec{x} \mid R=1,q)}{p(\vec{x} \mid q)}}{\dfrac{p(R=0 \mid q)p(\vec{x} \mid R=0,q)}{p(\vec{x} \mid q)}}$$

Binary Independence Model

$$O(R \mid q,\vec{x}) = \frac{p(R=1 \mid q,\vec{x})}{p(R=0 \mid q,\vec{x})} = \frac{p(R=1 \mid q)}{p(R=0 \mid q)} \cdot \frac{p(\vec{x} \mid R=1,q)}{p(\vec{x} \mid R=0,q)}$$

Constant for a given query

Needs estimation

- Using **Independence** Assumption:

$$\frac{p(\vec{x} \mid R=1,q)}{p(\vec{x} \mid R=0,q)} = \prod_{i=1}^{n} \frac{p(x_i \mid R=1,q)}{p(x_i \mid R=0,q)}$$

$$O(R \mid q,\vec{x}) = O(R \mid q) \cdot \prod_{i=1}^{n} \frac{p(x_i \mid R=1,q)}{p(x_i \mid R=0,q)}$$

Binary Independence Model

$$O(R \mid q, \vec{x}) = O(R \mid q) \cdot \prod_{i=1}^{n} \frac{p(x_i \mid R=1, q)}{p(x_i \mid R=0, q)}$$

- Since x_i is either 0 or 1:

$$O(R \mid q, \vec{x}) = O(R \mid q) \cdot \prod_{x_i=1} \frac{p(x_i=1 \mid R=1, q)}{p(x_i=1 \mid R=0, q)} \cdot \prod_{x_i=0} \frac{p(x_i=0 \mid R=1, q)}{p(x_i=0 \mid R=0, q)}$$

- Let $p_i = p(x_i=1 \mid R=1, q);\ \ r_i = p(x_i=1 \mid R=0, q);$

- Assume, for all terms not occurring in the query ($q_i=0$) $p_i = r_i$

$$O(R \mid q, \vec{x}) = O(R \mid q) \cdot \prod_{\substack{x_i=1 \\ q_i=1}} \frac{p_i}{r_i} \cdot \prod_{\substack{x_i=0 \\ q_i=1}} \frac{(1-p_i)}{(1-r_i)}$$

	document	relevant (R=1)	not relevant (R=0)
term present	$x_i = 1$	p_i	r_i
term absent	$x_i = 0$	$(1 - p_i)$	$(1 - r_i)$

Binary Independence Model

$$O(R \mid q, \vec{x}) = O(R \mid q) \cdot \prod_{x_i = q_i = 1} \frac{p_i}{r_i} \cdot \prod_{\substack{x_i = 0 \\ q_i = 1}} \frac{1 - p_i}{1 - r_i}$$

All matching terms

Non-matching query terms

$$O(R \mid q, \vec{x}) = O(R \mid q) \cdot \prod_{\substack{x_i = 1 \\ q_i = 1}} \frac{p_i}{r_i} \cdot \prod_{\substack{x_i = 1 \\ q_i = 1}} \left(\frac{1 - r_i}{1 - p_i} \cdot \frac{1 - p_i}{1 - r_i} \right) \prod_{\substack{x_i = 0 \\ q_i = 1}} \frac{1 - p_i}{1 - r_i}$$

$$O(R \mid q, \vec{x}) = O(R \mid q) \cdot \prod_{x_i = q_i = 1} \frac{p_i (1 - r_i)}{r_i (1 - p_i)} \cdot \prod_{q_i = 1} \frac{1 - p_i}{1 - r_i}$$

All matching terms

All query terms

Binary Independence Model

$$O(R \mid q, \vec{x}) = O(R \mid q) \cdot \prod_{x_i = q_i = 1} \frac{p_i (1 - r_i)}{r_i (1 - p_i)} \cdot \prod_{q_i = 1} \frac{1 - p_i}{1 - r_i}$$

Constant for each query

Only quantity to be estimated for rankings

Retrieval Status Value:

$$RSV = \log \prod_{x_i = q_i = 1} \frac{p_i (1 - r_i)}{r_i (1 - p_i)} = \sum_{x_i = q_i = 1} \log \frac{p_i (1 - r_i)}{r_i (1 - p_i)}$$

Binary Independence Model

[Robertson & Spärck-Jones 1976]

All boils down to computing RSV.

$$RSV = \log \prod_{x_i = q_i = 1} \frac{p_i(1-r_i)}{r_i(1-p_i)} = \sum_{x_i = q_i = 1} \log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

$$RSV = \sum_{x_i = q_i = 1} c_i; \qquad c_i = \log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

The c_i are **log odds ratios** (of contingency table a few slides back)
They function as the term weights in this model

So, how do we compute c_i's from our data?

Binary Independence Model

- Estimating RSV coefficients in theory
- For each term *i* look at this table of document counts:

$$c_i = \log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

Documents	Relevant	Non-Relevant	Total
$x_i=1$	s	$n-s$	n
$x_i=0$	$S-s$	$N-n-S+s$	$N-n$
Total	S	$N-S$	N

- Estimates:

$$p_i \approx \frac{s}{S} \qquad r_i \approx \frac{(n-s)}{(N-S)}$$

$$c_i \approx K(N,n,S,s) = \log \frac{s/(S-s)}{(n-s)/(N-n-S+s)}$$

For now, assume no zero terms. Remember smoothing.

Estimation – key challenge

If non-relevant documents are approximated by the whole collection, then r_i (prob. of occurrence in non-relevant documents for query) *is n/N* and

$$\log \frac{1-r_i}{r_i} = \log \frac{N-n-S+s}{n-s} \approx \log \frac{N-n}{n} \approx \log \frac{N}{n} = IDF!$$

Inverse Document Frequency (IDF)

Spärck-Jones (1972)

A key, still-important term weighting concept

Estimation – key challenge

p_i (probability of occurrence in relevant documents) cannot be approximated as easily

p_i can be estimated in various ways:

from relevant documents if you know some

Relevance weighting can be used in a feedback loop

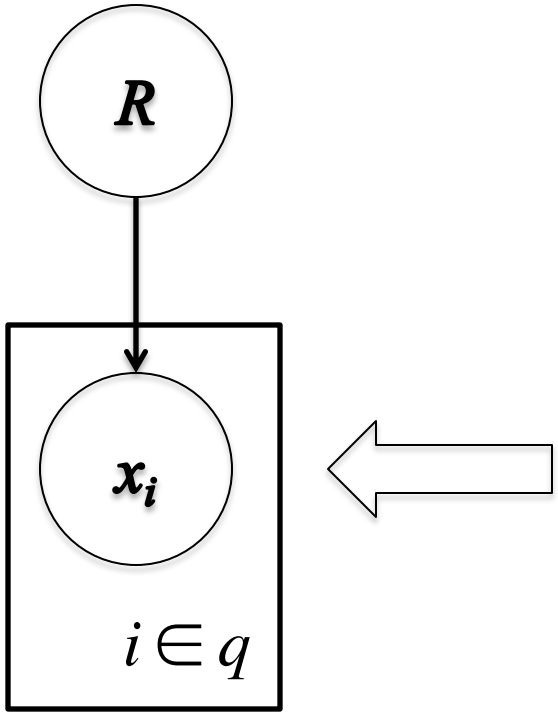
constant (Croft and Harper combination match) – then just get idf weighting of terms (with $p_i=0.5$)

$$RSV = \sum_{x_i=q_i=1} \log \frac{N}{n_i}$$

proportional to prob. of occurrence in collection

Greiff (SIGIR 1998) argues for 1/3 + 2/3 df_i/N

Graphical model for BIM – Bernoulli NB



Binary variables

$$x_i = (tf_i \neq 0)$$

4. Probabilistic Relevance Feedback

1. Guess a preliminary probabilistic description of R=1 documents; use it to retrieve a set of documents
2. Interact with the user to refine the description: learn some definite members with $R = 1$ and $R = 0$
3. Re-estimate p_i and r_i on the basis of these

   If i appears in V_i within set of documents V: $p_i = |V_i|/|V|$

   Or can combine new information with original guess (use Bayesian prior):

4. Repeat, thus generating a succession of approximations to relevant documents

$$p_i^{(2)} = \frac{|V_i| + \kappa p_i^{(1)}}{|V| + \kappa}$$

κ is prior weight

Pseudo-relevance feedback (iteratively auto-estimate p_i and r_i)

1. Assume that p_i *is* constant over all x_i in query and r_i as before
   p_i = 0.5 (even odds) for any given doc

2. Determine guess of relevant document set:
   V is fixed size set of highest ranked documents on this model

3. We need to improve our guesses for p_i and r_i, so
   Use distribution of x_i in docs in V. Let V_i be set of documents containing x_i
   $p_i = |V_i| / |V|$
   Assume if not retrieved then not relevant
   $r_i = (n_i - |V_i|) / (N - |V|)$

4. Go to 2. until converges then return ranking

PRP and BIM

It is possible to reasonably approximate probabilities

- But either require partial relevance information or need to make do with somewhat inferior term weights

Requires restrictive assumptions:

- "Relevance" of each document is independent of others
  - Really, it's bad to keep on returning **duplicates**
- Term independence
- Terms not in query don't affect the outcome
- Boolean representation of documents/queries
- Boolean notion of relevance

Some of these assumptions can be removed

Removing term independence

In general, index terms aren't independent

"Hong Kong"

Dependencies can be complex

van Rijsbergen (1979) proposed simple model of dependencies as a tree

Each term dependent on one other

Exactly Friedman and Goldszmidt's Tree Augmented Naive Bayes (AAAI 13, 1996)

In 1970s, estimation problems held back success of this model

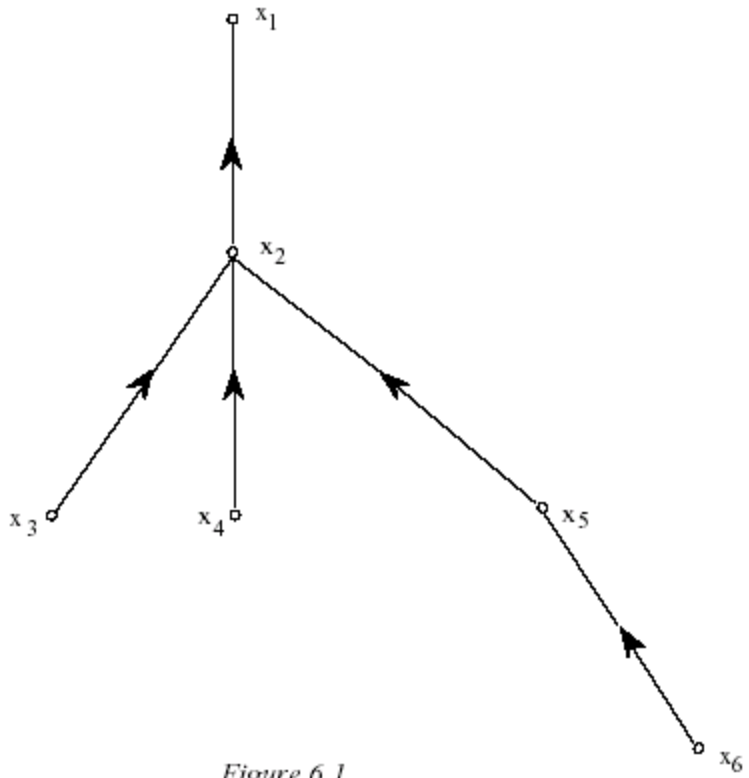


Figure 6.1.

5. Term frequency and the VSM

Right in the first lecture, we said that a page should rank higher if it mentions a word more

- Perhaps modulated by things like page length

Why not in BIM? Much of early IR was designed for titles or abstracts, and not for modern full text search

We now want a model with term frequency in it

We'll mainly look at a probabilistic model (BM25)

First, a quick summary of vector space model

Summary – vector space ranking (ch. 6)

Represent the query as a weighted term frequency/inverse document frequency (tf-idf) vector

(0, 0, 0, 0, 2.3, 0, 0, 0, 1.78, 0, 0, 0, …, 0, 8.17, 0, 0)

Represent each document as a weighted tf-idf vector

(1.2, 0, 3.7, 1.5, 2.0, 0, 1.3, 0, 3.7, 1.4, 0, 0, …, 3.5, 5.1, 0, 0)

Compute the cosine similarity score for the query vector and each document vector

Rank documents with respect to the query by score

Return the top *K* (e.g., *K* = 10) to the user

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

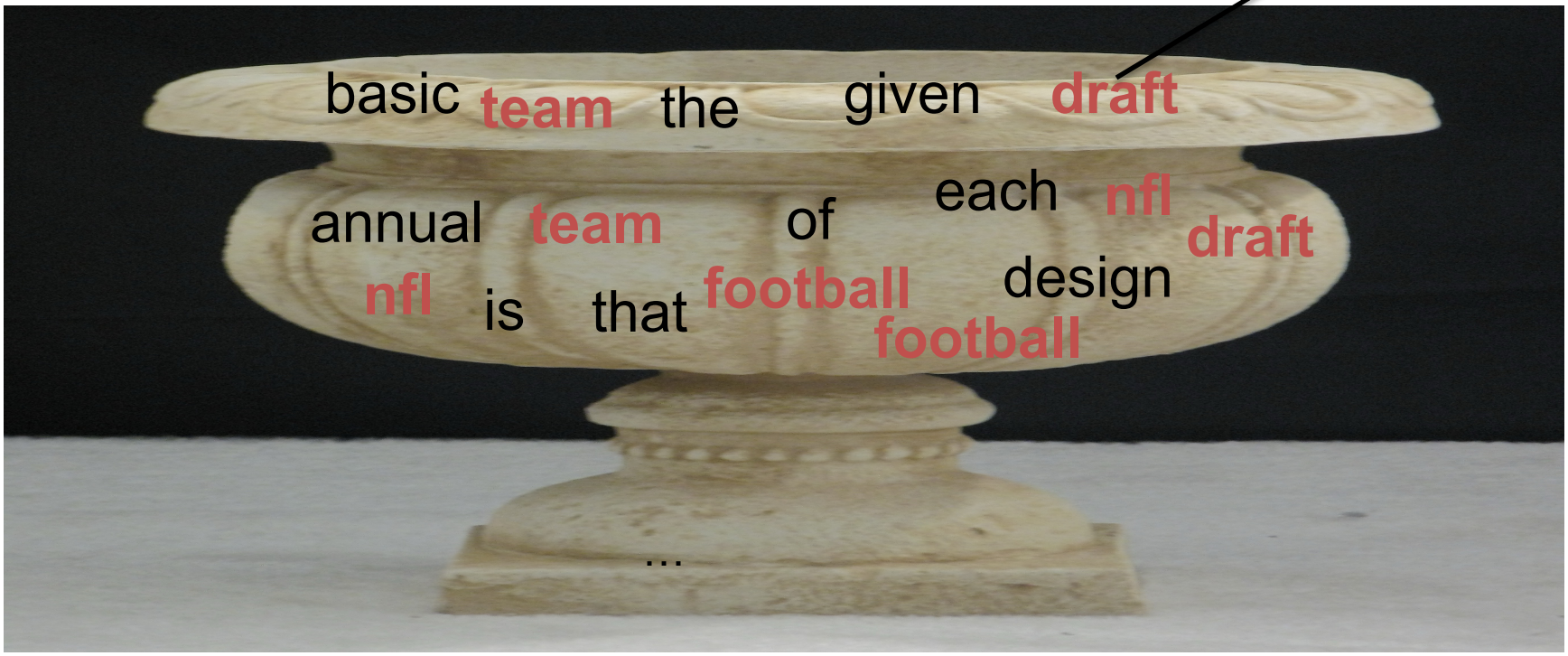
Goal: be sensitive to term frequency and document length while not adding too many parameters

- (Robertson and Zaragoza 2009; Spärck Jones et al. 2000)

Generative model for documents

- Words are drawn independently from the vocabulary using a multinomial distribution

... the **draft** is that each **team** is given a position in the **draft** …



Generative model for documents

- Distribution of term frequencies (*tf*) follows a binomial distribution – approximated by a Poisson

Poisson distribution

The Poisson distribution models the probability of *k,* the number of events occurring in a fixed interval of time/space, with known average rate λ (= cf/*T*), independent of the last event

$$p(k) = \frac{\lambda^k}{k!} e^{-\lambda}$$

Examples

Number of cars arriving at a toll booth per minute
Number of typos on a page

Poisson distribution

If *T* is large and *p* is small, we can approximate a binomial distribution with a Poisson where λ = *Tp*

$$p(k) = \frac{\lambda^k}{k!} e^{-\lambda}$$

Mean = Variance = λ = *Tp.*

Example *p* = 0.08, *T* = 20. Chance of 1 occurrence is:

Binomial

Poisson

$$P(1) = \binom{20}{1} (.08)^1 (.92)^{19} = .3282$$ … already close

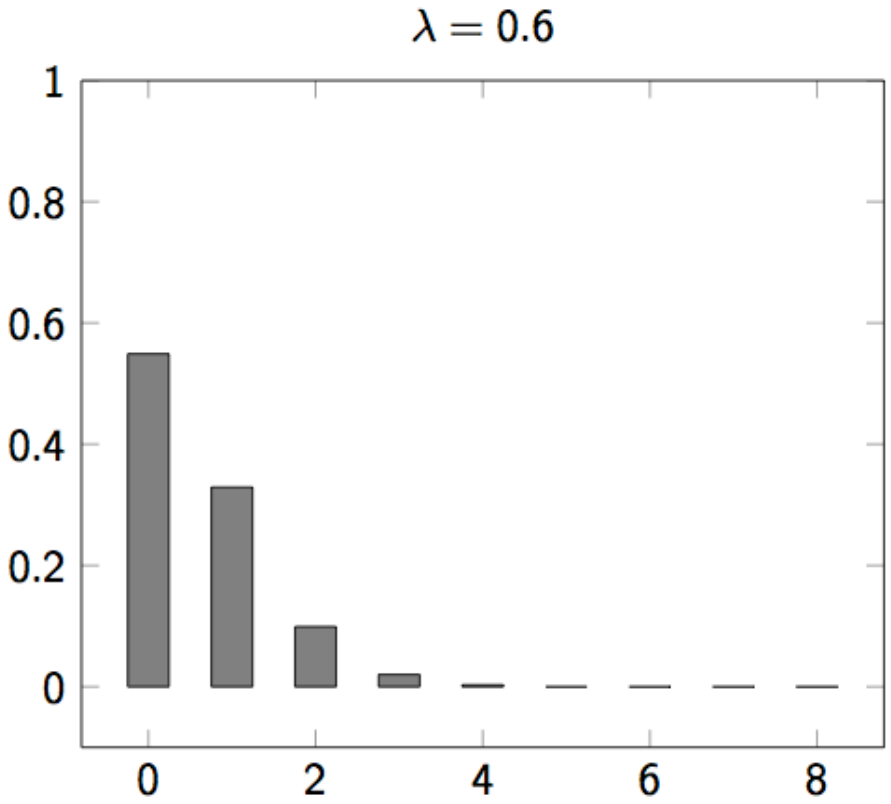
$$P(1) = \frac{[(20)(.08)]^1}{1!} e^{-(20)(.08)} = \frac{1.6}{1} e^{-1.6} = 0.3230$$

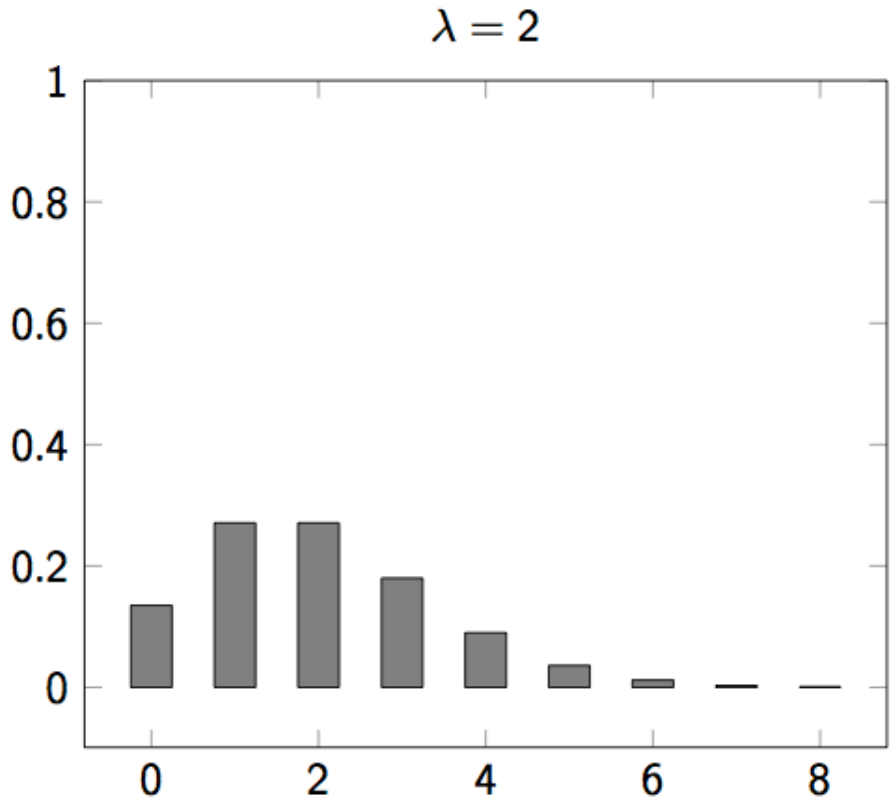
Poisson model

Assume that term frequencies in a document (tf_i) follow a Poisson distribution

- "Fixed interval" implies fixed document length … think roughly constant-sized document abstracts
  - … will fix later

Poisson distributions





(One) Poisson Model flaw

Is a reasonable fit for "general" words

Is a poor fit for topic-specific words

get higher *p(k)* than predicted too often

		Documents containing *k* occurrences of word (λ = 53/650)												
Freq	**Word**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
53	**expected**	**599**	**49**	**2**										
52	*based*	600	48	2										
53	*conditions*	604	39	7										
55	*cathexis*	619	22	3	2	1	2	0	1					
51	*comic*	642	3	0	1	0	0	0	0	0	0	1	1	2

Harter, "A Probabilistic Approach to Automatic Keyword Indexing", JASIST, 1975

Eliteness ("aboutness")

Model term frequencies using *eliteness*

What is eliteness?

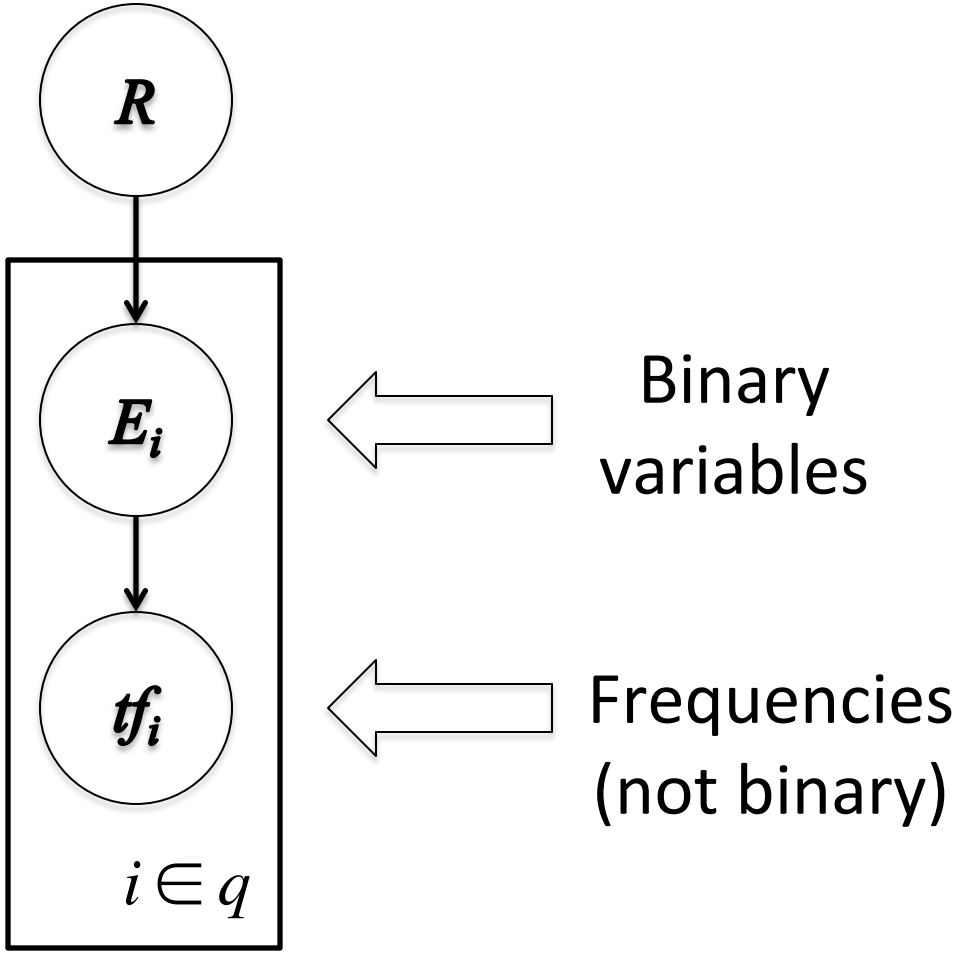
- Hidden variable for each document-term pair, denoted as E_i for term i
- Represents *aboutness*: a term is elite in a document if, in some sense, the document is about the concept denoted by the term
- Eliteness is binary
- Term occurrences depend only on eliteness…
- … but eliteness depends on relevance

Elite terms

Text from the Wikipedia page on the NFL draft showing **elite terms**

The **National Football League Draft** is an annual event in which the **National Football League** (**NFL**) **teams select eligible college football players**. It serves as the **league's** most common source of **player recruitment**. The basic design of the **draft** is that each **team** is given a **position** in the **draft order** in **reverse order** relative to its **record** …

Graphical model with eliteness



Retrieval Status Value

Similar to the BIM derivation, we have

$$RSV^{elite} = \sum_{i \in q, tf_i > 0} c_i^{elite}(tf_i);$$

where

$$c_i^{elite}(tf_i) = \log \frac{p(TF_i = tf_i \,|\, R = 1)\, p(TF_i = 0 \,|\, R = 0)}{p(TF_i = 0 \,|\, R = 1)\, p(TF_i = tf_i \,|\, R = 0)}$$

and using eliteness, we have:

$$p(TF_i = tf_i \,|\, R) = p(TF_i = tf_i \,|\, E_i = elite)\, p(E_i = elite \,|\, R) + p(TF_i = tf_i \,|\, E_i = \overline{elite})(1 - p(E_i = elite \,|\, R))$$

2-Poisson model

The problems with the 1-Poisson model suggests fitting two Poisson distributions

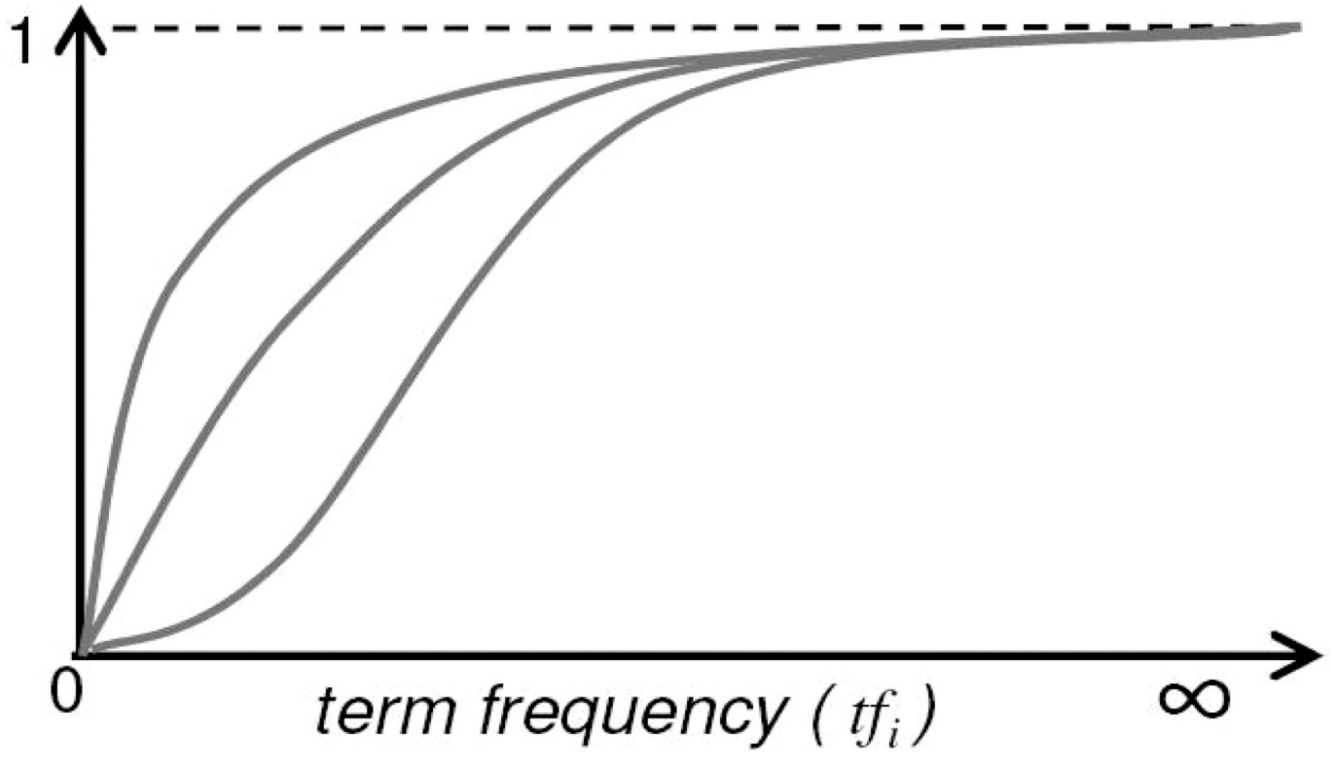
In the "2-Poisson model", the distribution is different depending on whether the term is elite or not

$$p(TF_i = k_i \mid R) = \pi \frac{\lambda^k}{k!} e^{-\lambda} + (1 - \pi) \frac{\mu^k}{k!} e^{-\mu}$$

where π is probability that document is elite for term

but, unfortunately, we don't know π, λ, μ

Let's get an idea: Graphing $c_i^{elite}(tf_i)$ for different parameter values of the 2-Poisson



Qualitative properties

$$c_i^{elite}(0) = 0$$

increases monotonically with tf_i

$$c_i^{elite}(tf_i)$$

… but asymptotically approaches a maximum value as

[not true for simple scaling of tf]

$$tf_i \rightarrow \infty$$

… with the asymptotic limit being

c_i^{BIM} ⟸ Weight of eliteness feature

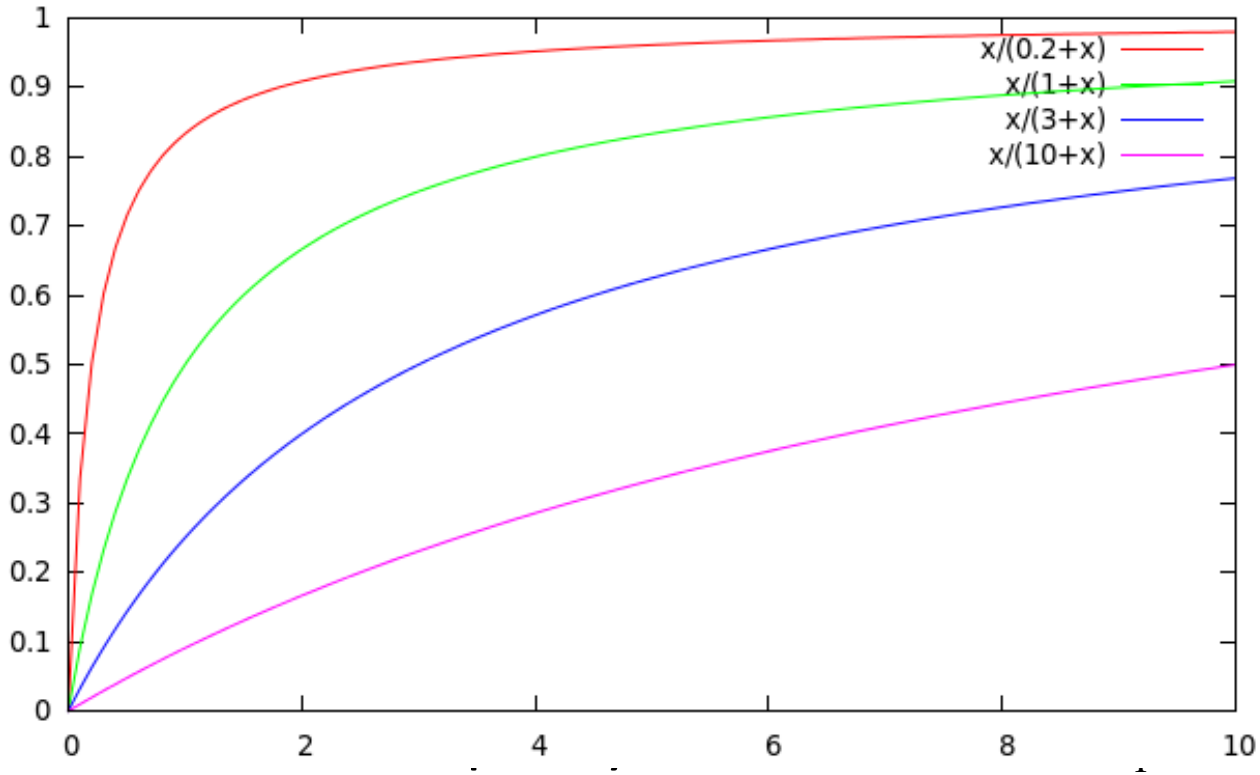
Approximating the saturation function

Estimating parameters for the 2-Poisson model is not easy

… So approximate it with a simple parametric curve that has the same qualitative properties

$$\frac{tf}{k_1 + tf}$$

Saturation function



For

sigr

Cor

ntribute

“Early” versions of BM25

Version 1: using the saturation function

$$c_i^{BM25v1}(tf_i) = c_i^{BIM}\frac{tf_i}{k_1 + tf_i}$$

Version 2: BIM simplification to IDF

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

(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

Document length normalization

Longer documents are likely to have larger tf_i values

Why might documents be longer?

Verbosity: suggests observed tf_i too high

Larger scope: suggests observed tf_i may be right

A real document collection probably has both effects
… so should apply some kind of partial normalization

Document length normalization

Document length:

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

avdl: Average document length over collection

Length normalization component

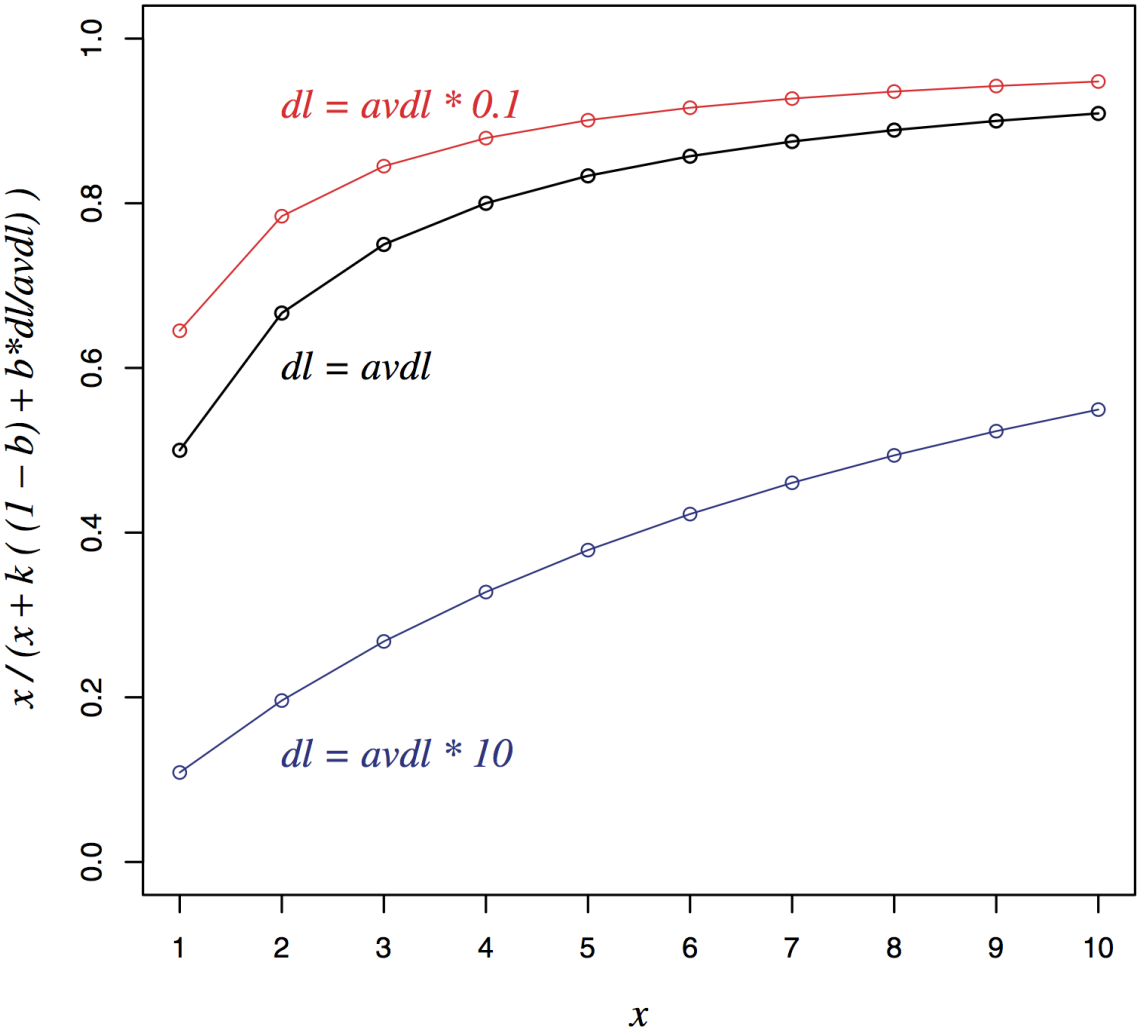
$$B = \left((1 - b) + b \frac{dl}{avdl} \right),$$

$b = 1$ full document length normalization

$b = 0$ no document length normalization

$$0 \leq b \leq 1$$

Document length normalization



Okapi BM25

Normalize *tf* using document length

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

$$c_i^{BM25}(tf_i) = \log\frac{N}{df_i} \times \frac{(k_1+1)tf_i'}{k_1 + tf_i'}$$

BM25 ranking function $= \log\frac{N}{df_i} \times \frac{(k_1+1)tf_i}{k_1((1-b)+b\frac{dl}{avdl}) + tf_i}$

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

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

IIR sec. 11.4.3 discusses incorporating query term weighting and (pseudo) relevance feedback

Why is BM25 better than VSM tf-idf?

Suppose your query is [machine learning]

Suppose you have 2 documents with term counts:

doc1: learning 1024; machine 1

doc2: learning 16; machine 8

tf-idf: $\log_2 tf * \log_2 (N/df)$

doc1: 11 * 7 + 1 * 10 = **87**

doc2: 5 * 7 + 4 * 10 = **75**

BM25: $k_1 = 2$

doc1: 7 * 3 + 10 * 1 = **31**

doc2: 7 * 2.67 + 10 * 2.4 = **42.7**

7. Ranking with features

Textual features

- Zones: Title, author, abstract, body, anchors, …
- Proximity
- …

Non-textual features

- File type
- File age
- Page rank
- …

Ranking with zones

Straightforward idea:

- Apply your favorite ranking function (BM25) to each zone separately
- Combine zone scores using a weighted linear combination

But that seems to imply that the eliteness properties of different zones are different and independent of each other

- …which seems unreasonable

Ranking with zones

Alternate idea

- Assume eliteness is a term/document property shared across zones
- … but the relationship between eliteness and term frequencies are zone-dependent
  - e.g., denser use of elite topic words in title

Consequence

- First combine evidence across zones for each term
- Then combine evidence across terms

BM25F with zones

Calculate a weighted variant of total term frequency
… and a weighted variant of document length

$$\tilde{tf}_i = \sum_{z=1}^{Z} v_z tf_{zi} \qquad \tilde{dl} = \sum_{z=1}^{Z} v_z len_z \qquad avd\tilde{l} = \text{Average } \tilde{dl} \text{ across all documents}$$

where

v_z is zone weight
tf_{zi} is term frequency in zone z
len_z is length of zone z
Z is the number of zones

Simple BM25F with zones

$$RSV^{SimpleBM25F} = \sum_{i \in q} \log \frac{N}{df_i} \cdot \frac{(k_1 + 1)\tilde{tf}_i}{k_1((1-b) + b\frac{\tilde{dl}}{avdl}) + \tilde{tf}_i}$$

Simple interpretation: zone z is "replicated" v_z times

But we may want zone-specific parameters (k_1, b, IDF)

BM25F

Empirically, zone-specific length normalization (i.e., zone-specific b) has been found to be useful

$$\tilde{tf}_i = \sum_{z=1}^{Z} v_z \frac{tf_{zi}}{B_z}$$

$$B_z = \left((1 - b_z) + b_z \frac{len_z}{avlen_z} \right), \qquad 0 \leq b_z \leq 1$$

$$RSV^{BM25F} = \sum_{i \in q} \log \frac{N}{df_i} \cdot \frac{(k_1 + 1)\tilde{tf}_i}{k_1 + \tilde{tf}_i}$$

See Robertson and Zaragoza (2009: 364)

Resources

S. E. Robertson and K. Spärck Jones. 1976. Relevance Weighting of Search Terms. *Journal of the American Society for Information Sciences* 27(3): 129–146.

C. J. van Rijsbergen. 1979. *Information Retrieval.* 2nd ed. London: Butterworths, chapter 6. http://www.dcs.gla.ac.uk/Keith/Preface.html

K. Spärck Jones, S. Walker, and S. E. Robertson. 2000. A probabilistic model of information retrieval: Development and comparative experiments. Part 1. *Information Processing and Management* 779–808.

S. E. Robertson and H. Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval* 3(4): 333-389.

End of Slides