Introduction to

Information Retrieval

- Scoring and Term Weighting
    - The Vector Space Model

Contents

- Parametric and zone indexes
- Ranked retrieval
- Term Weighting schemes
- Vector space scoring

Parametric and zone indexes

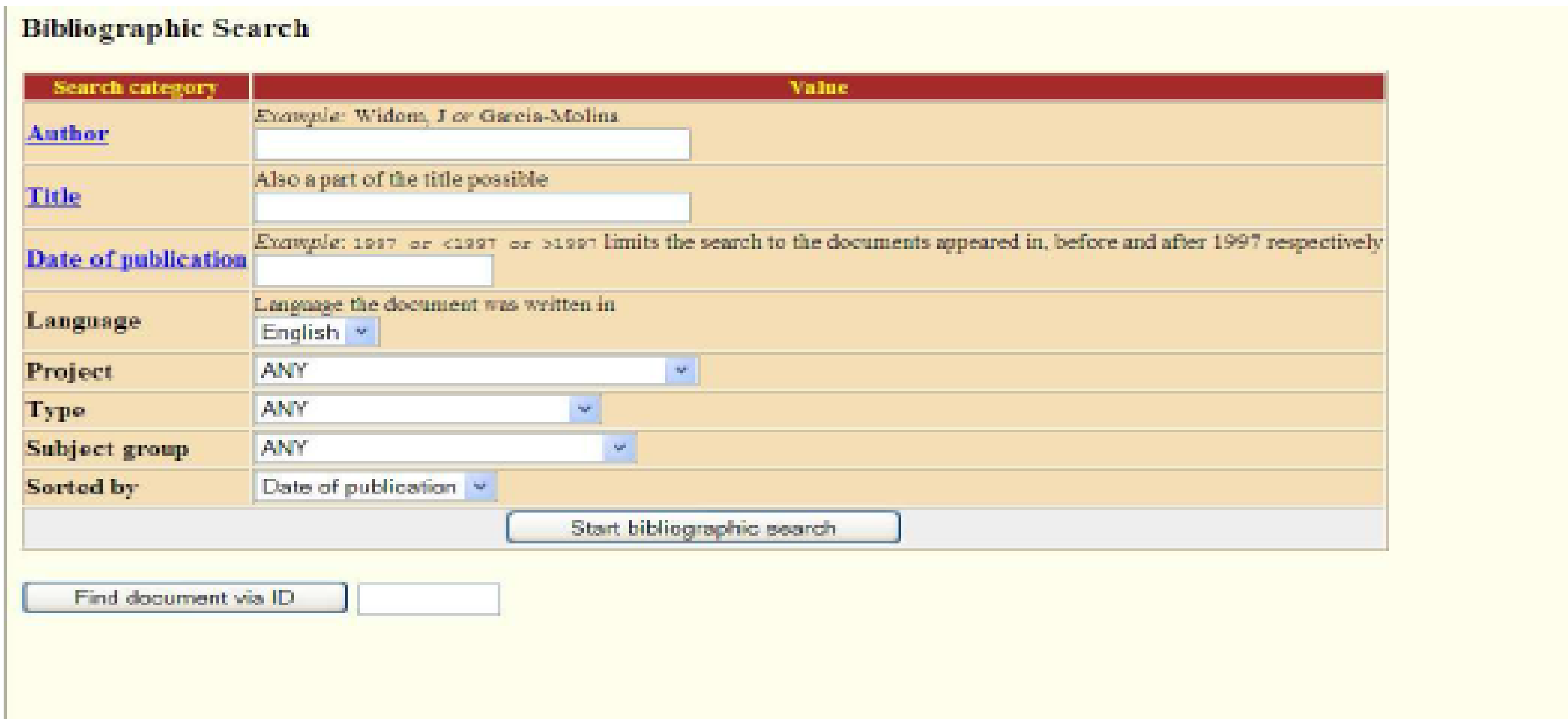
- Each document has, in addition to text, some “meta-data” in fields e.g.,
  - Language = French
  - Format = pdf
  - Subject = Physics etc.
  - Date = Feb 2000

Fields

Values

- A parametric search interface allows the user to combine a full-text query with selections on these field values e.g.,
  - language, date range, etc.

User view



Bibliographic Search

Search category	Value
Author	*Example:* Widom, J *or* Garcia-Molina
Title	Also a part of the title possible
Date of publication	*Example:* 1997 or <1997 or >1997 limits the search to the documents appeared in, before and after 1997 respectively
Language	Language the document was written in English
Project	ANY
Type	ANY
Subject group	ANY
Sorted by	Date of publication

Start bibliographic search

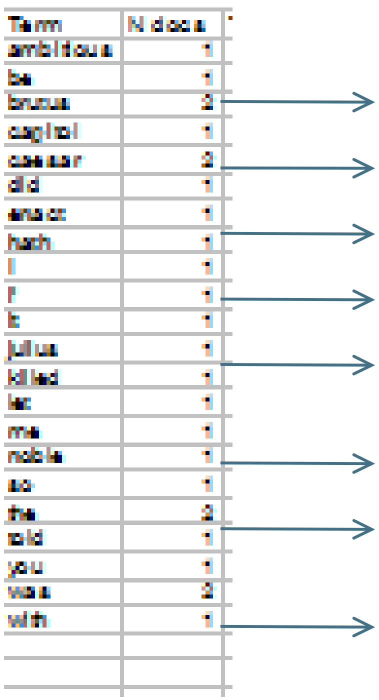
Find document via ID

► **Figure 6.1** Parametric search. In this example we have a collection with fields allowing us to select publications by zones such as Author and fields such as Language.

Zones

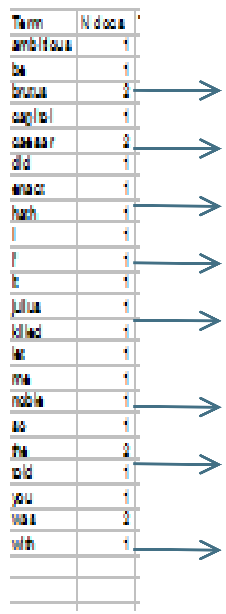
- A zone is an identified region within a doc
  - E.g., Title, Abstract, Bibliography
- Contents of a zone are free text
  - Not a “finite” vocabulary
- Indexes for each zone - allow queries like
  - find documents with merchant in the title and william in the author list and the phrase gentle rain in the body

Zone indexes – simple view



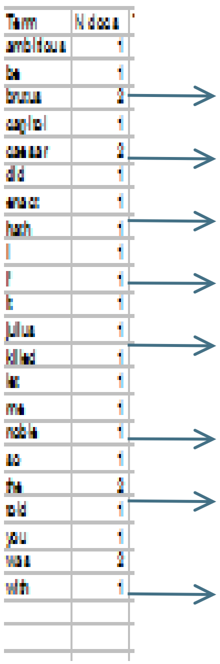
Term	N docs
ambitious	1
be	1
brutus	2
capitol	1
caesar	2
did	1
enact	1
hath	1
I	1
i'	1
it	1
julius	1
killed	1
let	1
me	1
noble	1
so	1
the	2
told	1
you	1
was	2
with	1

Title



Term	N docs
ambitious	1
be	1
brutus	2
capitol	1
caesar	2
did	1
enact	1
hath	1
I	1
i'	1
it	1
julius	1
killed	1
let	1
me	1
noble	1
so	1
the	2
told	1
you	1
was	2
with	1

Author

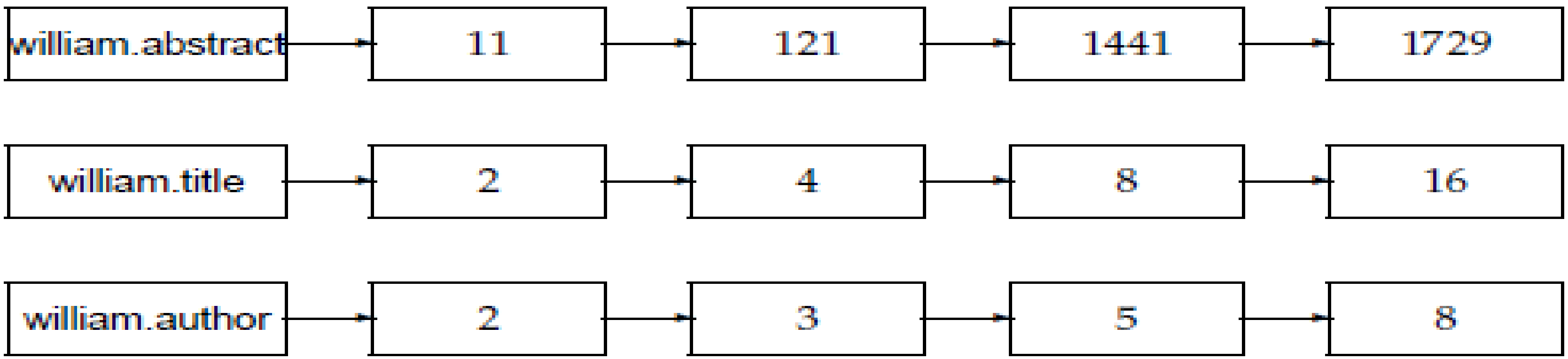


Term	N docs
ambitious	1
be	1
brutus	2
capitol	1
caesar	2
did	1
enact	1
hath	1
I	1
i'	1
it	1
julius	1
killed	1
let	1
me	1
noble	1
so	1
the	2
told	1
you	1
was	2
with	1

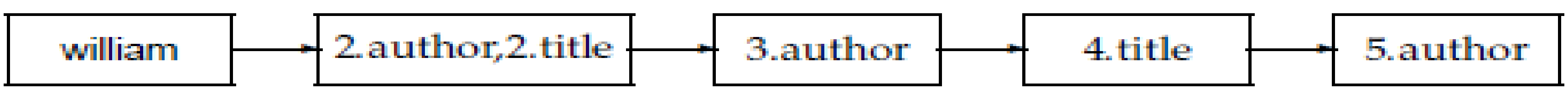
Body

etc.

Zone indexes – common view



► **Figure 6.2** Basic zone index ; zones are encoded as extensions of dictionary entries.



► **Figure 6.3** Zone index in which the zone is encoded in the postings rather than the dictionary.

Weighted zone scoring

- Given a Boolean query *q and a document d, weighted zone scoring assigns* to the pair (*q, d) a score in the interval [0, 1],*
  - *by computing a linear combination* of *zone scores*
  - *where each zone of the document contributes a Boolean* value.
- Specifically,
  - let there are L zones. Let $g_1, \ldots, g_L \in [0, 1]$ *such that* $\sum_{i=1}^{L} g_i = 1$
  - let s_i *be the* Boolean score denoting a match (or absence thereof) between *q and the ith* zone.
  - *Then,* the weighted zone score is defined to be $\sum_{i=1}^{L} g_i * s_i$

Example: Weighted zone scoring

- Query Q= shakespeare
- consider a collection in which each document has three zones: ***author, title and body***
- Suppose we set *g1 = 0.2, g2 = 0.3 and g3 = 0.5 where g1, g2 and g3 represents the **author, title and body** zone weights.*
- If the term shakespeare appear in the **title** and **body** zones but not the *author* zone of a document, the score of this document would be
- 0.8.

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

Recall (Lecture 1): 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 : 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.
- We will look at "recovering" positional information later in this course.
- For now: bag of words model

Term frequency tf

- The term frequency $\mathrm{tf}_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.
- We want to use tf when computing query-document match scores. But how?
- Raw term frequency is not what we want:
  - 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 incr[...] term frequency.

NB: frequency = count in IR

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*:
- 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.

Document frequency

- Rare terms are more informative than frequent terms
  - Recall stop words
- → We want a high weight for rare terms.

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
  - $\text{df}_t \leq N$
- We define the idf (inverse document frequency) of t by $\text{idf}_t = \log_{10}(N/\text{df}_t)$
  - We use log (N/df_t) instead of N/df_t to "dampen" the effect of idf.

Will turn out the base of the log is immaterial.

idf example, suppose N = 806791

term	df_t	idf_t
car	18,165	1.65
auto	6723	2.08
insurance	19,241	1.62
best	25,235	1.5

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

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

tf-idf weighting

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

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

- 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

Binary → count → tf-idf 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 $\in R^{|V|}$

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
  - …

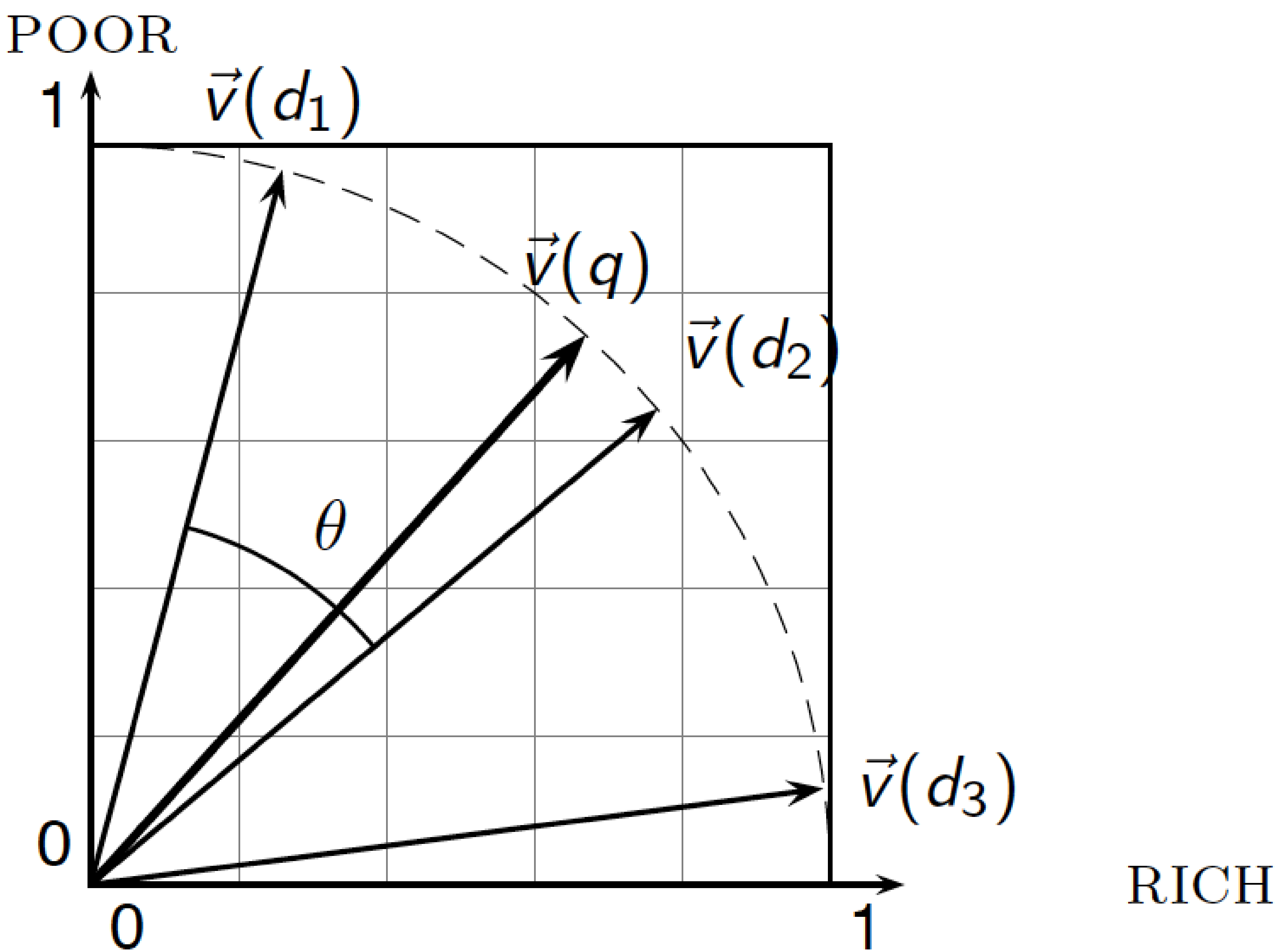
The Vector Space Model for Scoring

- The set of documents in a collection may be viewed as a set of vectors in a 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.

Vector Similarity

- How do we quantify the similarity between two documents in this vector space?
- A first attempt : the magnitude of the vector difference between two document vectors.
  - Drawback: two documents with very similar content can have a significant vector difference simply because one is much longer than the other.
- **Solution** to compensate for the effect of document length is to compute the *cosine similarity*.

Cosine similarity illustrated



From angles to cosines

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

cosine(document,document)

DOT PRODUCT

$$\mathrm{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}, \qquad (2)$$

EUCLIDEAN LENGTH

The dot product $\vec{x} \cdot \vec{y}$ of two vectors is defined as $\sum_{i=1}^{M} x_i y_i$

The Euclidean length of d is defined to be $\sqrt{\sum_{i=1}^{M} \vec{V}_i^2(d)}$.

Cosine for length-normalized vectors

The effect of the denominator of Equation (2) is thus to *length-normalize* the vectors $\vec{V}(d_1)$ and $\vec{V}(d_2)$ to unit vectors $\vec{v}(d_1) = \vec{V}(d_1)/|\vec{V}(d_1)|$ and $\vec{v}(d_2) = \vec{V}(d_2)/|\vec{V}(d_2)|$.

Then we can rewrite the previous equation as:

$$\text{sim}(d_1, d_2) = \vec{v}(d_1) \cdot \vec{v}(d_2). \quad (3)$$

Example: N= 1,000,000

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

DF	Term
5000	auto
50000	best
10000	car
1000	insurance

Example: N= 1,000,000

Document: *car insurance auto insurance*

Query: *best car insurance*

Term	Query						Document				Prod
auto											
best											
car											
insurance											

Example: N= 1,000,000

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	2.3	0.47	0
best	1	1	50000	1.3	1.3	0.34	0	0	0	0	0
car	1	1	10000	2.0	2.0	0.53	1	1	2.0	0.41	0.22
insurance	1	1	1000	3.0	3.0	0.79	2	1.3	3.9	0.80	0.63

Query length = $\sqrt{0^2 + 1.3^2 + 2^2 + 3^2} \approx 3.8$

Doc length = $\sqrt{2.3^2 + 0^2 + 2^2 + 3.9^2} \approx 4.9$

Score = 0+0+0.22+0.63 = 0.85