Chapter 4

Modeling

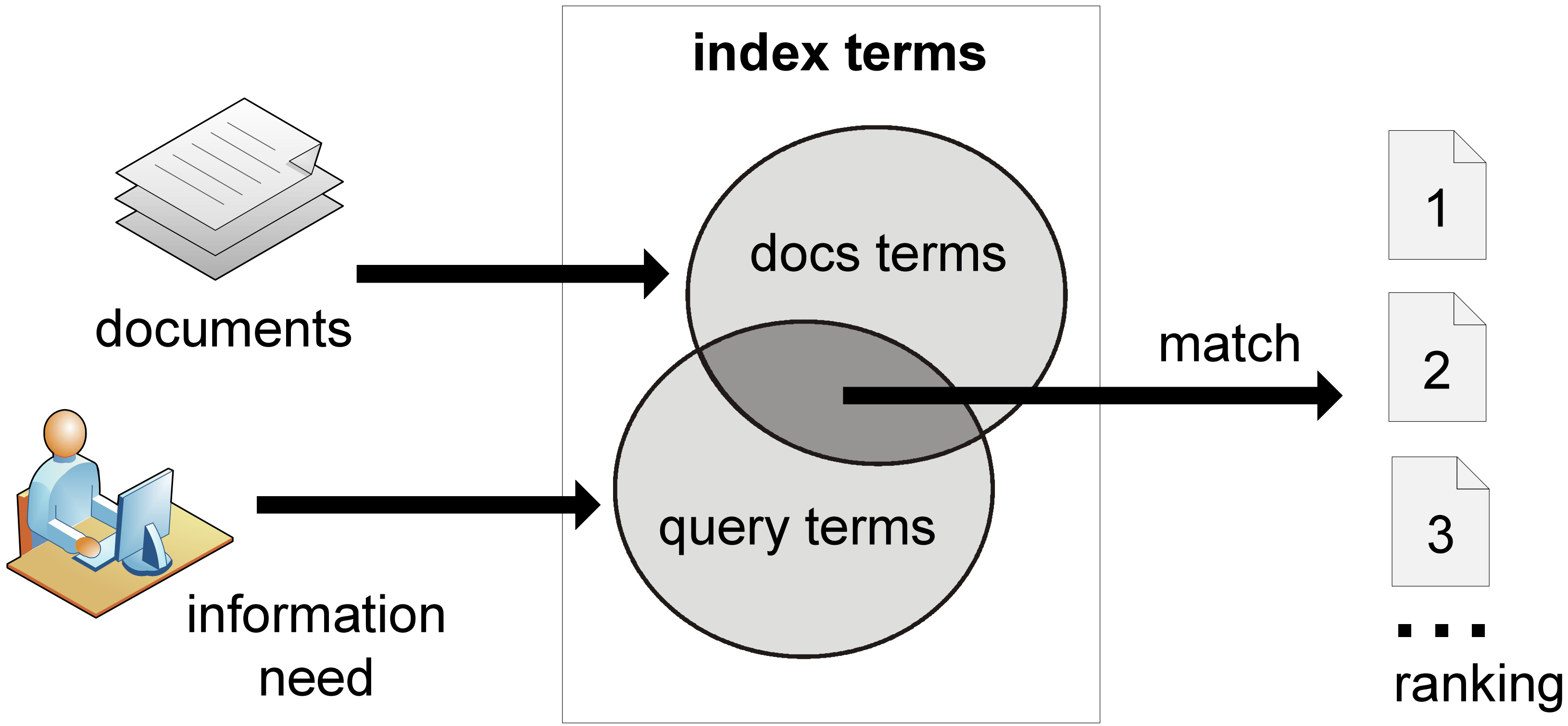
Introduction to IR Models
Retrieval: Ad Hoc x Filtering
Classic IR Models

Introduction

- IR systems usually adopt **index terms** to index and retrieve documents
- Index term:
  - A keyword that has some meaning on its own
  - Usually plays the role of a noun
  - In a more general form, it is any word that appears in a document
- Stemming might be used:
  - connect: connecting, connection, connections

Introduction

- Information retrieval process

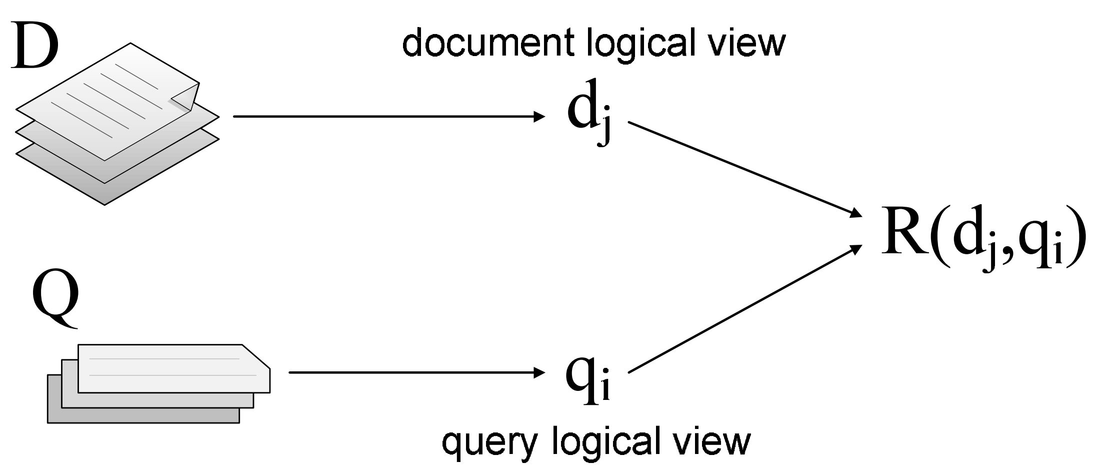


Introduction

- A **ranking** is an ordering of documents that (hopefully) reflects the **relevance** of the documents to a user query
- Thus, any IR system has to deal with the problem of predicting which documents the users will find relevant
- The problem is further compounded by the fact that two users might disagree on what is relevant and what is not

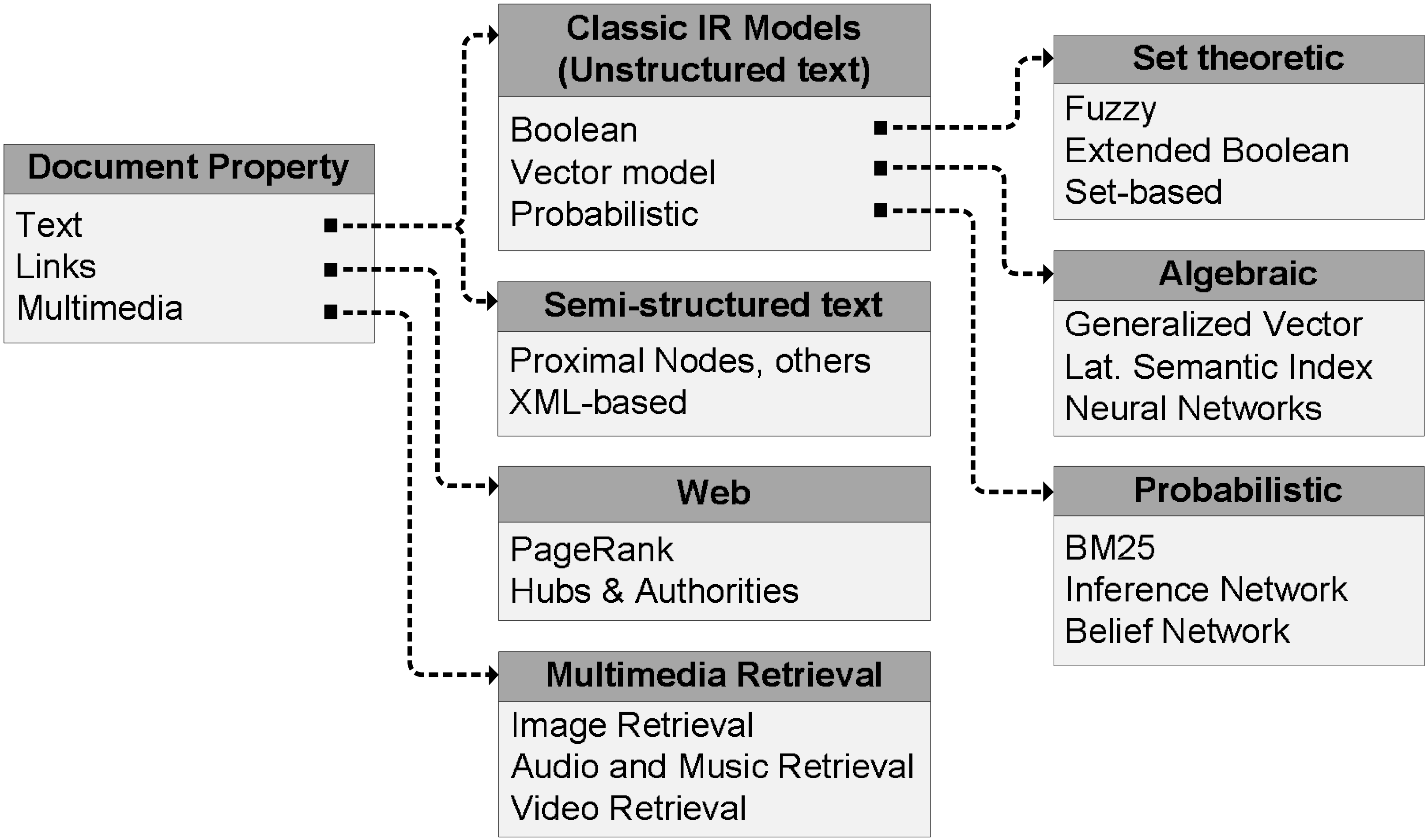
IR Models

- An **IR model** is a quadruple $[\mathbf{D}, \mathbf{Q}, \mathcal{F}, R(q_i, d_j)]$ where
  1. $\mathbf{D}$ is a set of logical views for the documents in the collection
  2. $\mathbf{Q}$ is a set of logical views for the user queries
  3. $\mathcal{F}$ is a framework for modeling document representations, queries, and their relationships
  4. $R(q_i, d_j)$ is a ranking function which associates a real number with a query $q_i \in \mathbf{Q}$ and a document $d_j \in \mathbf{D}$.



IR Models

- A taxonomy of information retrieval models



IR Models

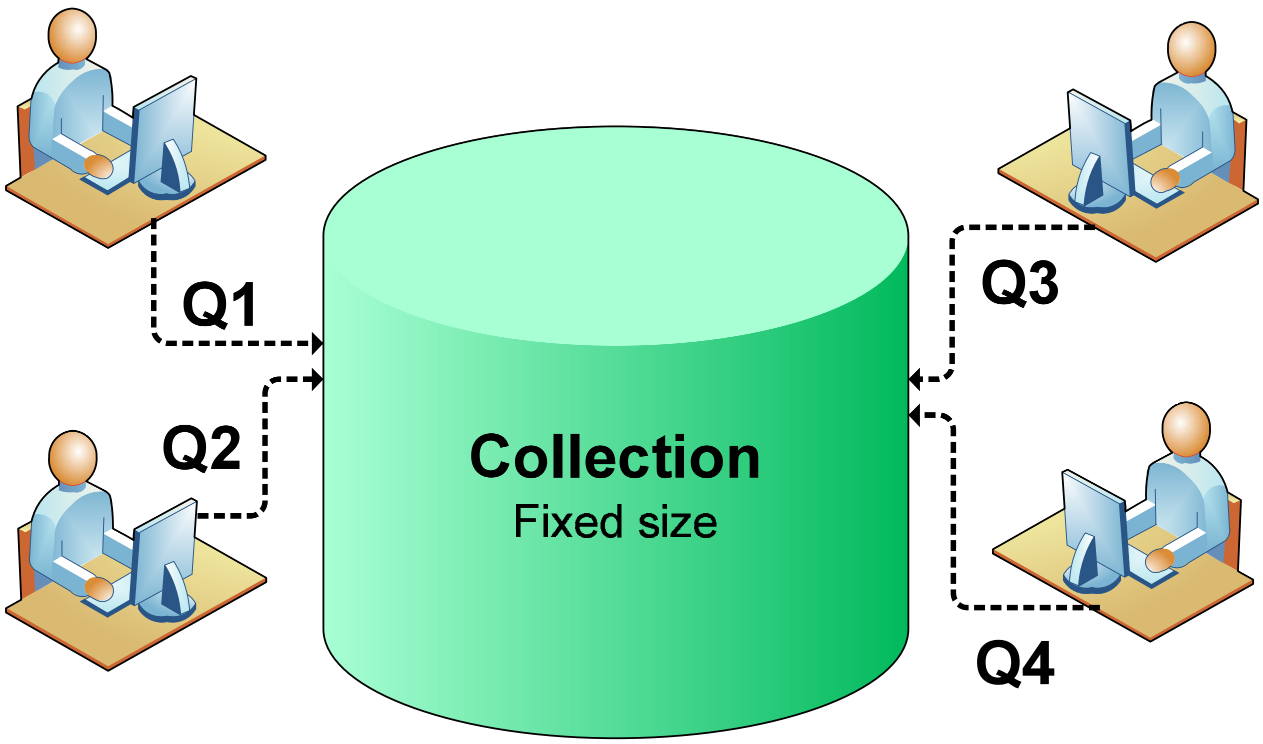
- The IR model, the logical view of the docs, and the retrieval task are distinct aspects of the system

Logical View of Documents

User Tasks		Index terms	Full Text	Full Text + Structure
	Retrieval	Classic Set theoretic Algebraic Probabilistic	Classic Set theoretic Algebraic Probabilistic	Structured
	Browsing	Flat	Flat Hypertext	Structure guided Hypertext

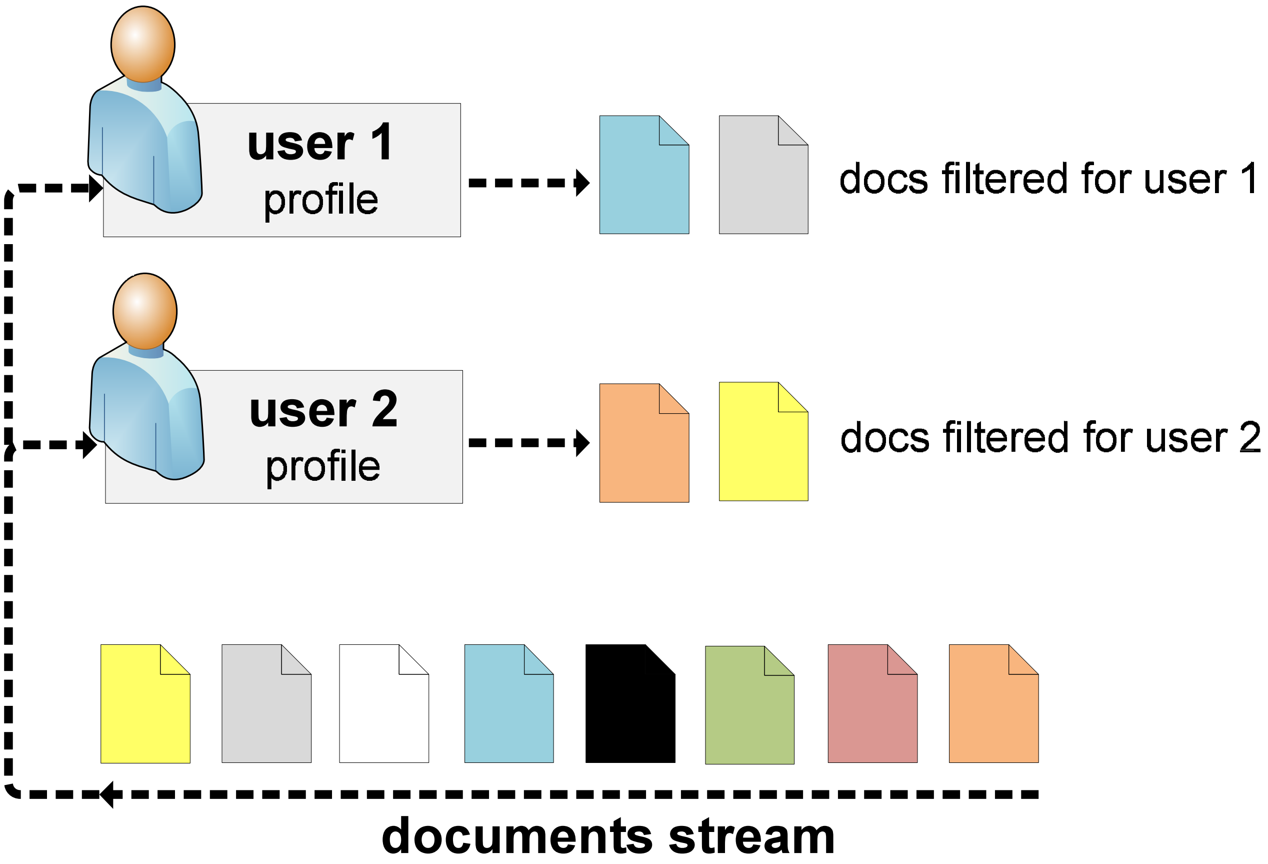
Retrieval: Ad Hoc x Filtering

- Ad Hoc Retrieval:



Retrieval: Ad Hoc x Filtering

- Filtering



Classic IR Models

Classic IR Models: Basic Concepts

- Each document represented by a set of representative keywords or index terms
- An index term is a document word useful for remembering the document main themes
- Usually, index terms are nouns because nouns have meaning by themselves
- However, it might be interesting to assume that all words are index terms (full text representation)

Classic IR Models: Basic Concepts

- Let,
  - t the number of index terms in the document collection
  - k_i a generic index term
- Then,
  - The **vocabulary** $V = \{k_1, \ldots, k_t\}$ is the set of all distinct index terms in the collection

$$V = \boxed{k_1 \quad k_2 \quad k_3 \; \ldots \; k_t}$$

vocabulary of *t*
index terms

Classic IR Models: Basic Concepts

- Documents and queries can be represented by **patterns of term co-occurrence of terms**

$$V = \boxed{k_1 \quad k_2 \quad k_3 \ \ldots \ k_t}$$

$$\boxed{1 \quad 0 \quad 0 \ \ldots \ 0}$$ pattern that represents documents (and queries) with the term k_1 and no other

$$\vdots$$

$$\boxed{1 \quad 1 \quad 1 \ \ldots \ 1}$$ pattern that represents documents (and queries) with all index terms

- Each of these patterns of term co-occurence is called a **term conjunctive component**
- For each document d_j (or query q) we associate a unique term conjunctive component $c(d_j)$ (or $c(q)$)

Classic IR Models: Basic Concepts

- Term-term correlation matrix
  - Let $\vec{m} = (m_{ij})$ be a term-document matrix $t \times N$
  - The matrix $\vec{c} = \vec{m}\vec{m}^t$ is a term-term correlation matrix

$$
\underbrace{
\begin{array}{c} \\ k_1 \\ k_2 \\ k_3 \end{array}
\begin{array}{c} \begin{array}{cc} d_1 & d_2 \end{array} \\ \begin{bmatrix} f_{1,1} & f_{1,2} \\ f_{2,1} & f_{2,2} \\ f_{3,1} & f_{3,2} \end{bmatrix} \\ \vec{m} \end{array}
\quad * \quad
\begin{array}{c} \\ d_1 \\ d_2 \end{array}
\begin{array}{c} \begin{array}{ccc} k_1 & k_2 & k_3 \end{array} \\ \begin{bmatrix} f_{1,1} & f_{2,1} & f_{3,1} \\ f_{1,2} & f_{2,2} & f_{3,2} \end{bmatrix} \\ \vec{m}^t \end{array}
}
$$

$$
\begin{array}{c} \\ k_1 \\ k_2 \\ k_3 \end{array}
\begin{array}{c} \begin{array}{ccc} k_1 & k_2 & k_3 \end{array} \\ \begin{bmatrix} f_{1,1}f_{1,1} + f_{1,2}f_{1,2} & f_{1,1}f_{2,1} + f_{1,2}f_{2,2} & f_{1,1}f_{3,1} + f_{1,2}f_{3,2} \\ f_{2,1}f_{1,1} + f_{2,2}f_{1,2} & f_{2,1}f_{2,1} + f_{2,2}f_{2,2} & f_{2,1}f_{3,1} + f_{2,2}f_{3,2} \\ f_{3,1}f_{1,1} + f_{3,2}f_{1,2} & f_{3,1}f_{2,1} + f_{3,2}f_{2,2} & f_{3,1}f_{3,1} + f_{3,2}f_{3,2} \end{bmatrix} \end{array}
$$

term-term matrix

Classic IR Models: Basic Concepts

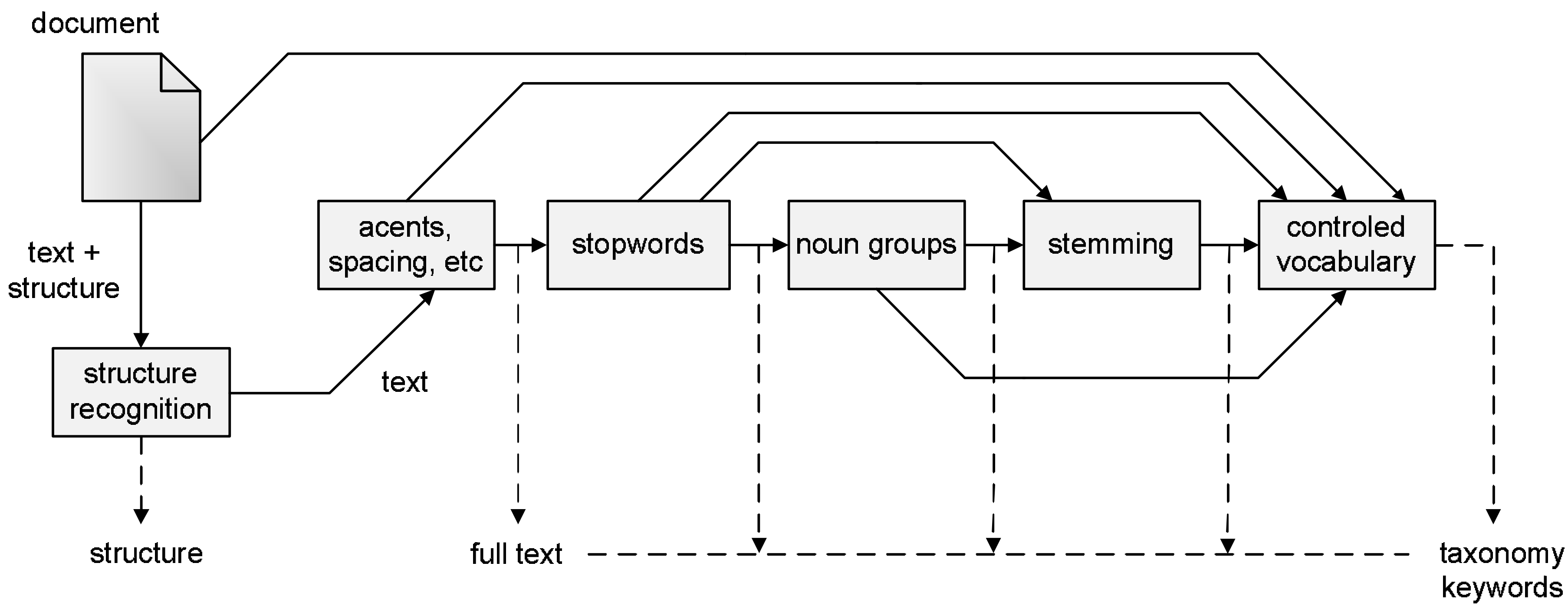
- This matrix establishes a relationship between any two terms of a collection
- This relationship is based on their joint co-occurrences inside documents
- Each element $c_{u,v} \in \vec{c}$ expresses a correlation between terms k_u and k_v, given by

$$c_{u,v} = \sum_{d_j} f_{u,j} \times f_{v,j}$$

- Higher the number of documents in which the terms k_u and k_v co-occur, stronger is this correlation

Classic IR Models: Basic Concepts

- Logical view of a document: from full text to a set of index terms



The Boolean Model

The Boolean Model

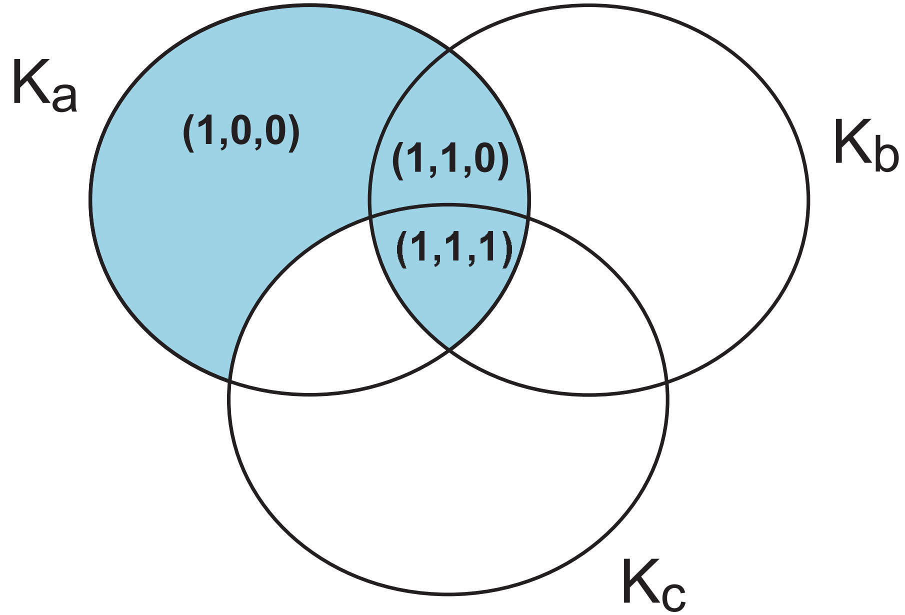
- Simple model based on **set theory** and **boolean algebra**
- Queries specified as boolean expressions
  - quite intuitive and precise semantics
  - neat formalism
  - example of query: $q = k_a \wedge (k_b \vee \neg k_c)$
- Term-document frequencies in the term-document matrix are all binary
  - w_{iq}: weight associated with pair (k_i, q)
  - $w_{iq} \in \{0, 1\}$: terms either present or absent
  - $\vec{d}_q = (w_{1q}, w_{2q}, ..., w_{tq})$: weighted vector associated with q

The Boolean Model

- A term conjunctive component that satisfies the conditions of a query q is called a **query conjunctive component** $c(q)$
- A query q rewritten as a disjunction of those components is called the **disjunct normal form** q_{DNF}
- To illustrate, consider
  - query $q = k_a \wedge (k_b \vee \neg k_c)$
  - vocabulary $V = \{k_a, k_b, k_c\}$
- Then
  - $q_{DNF} = (1,1,1) \vee (1,1,0) \vee (1,0,0)$
  - $c(q)$: a conjunctive component for q

The Boolean Model

- $q = k_a \wedge (k_b \vee \neg k_c)$



- Given a document d_j, let $c(d_j)$ be the corresponding document conjunctive component
  - $sim(q, d_j) = 1$, if $\exists c(q) | c(q) = c(d_j)$
  - $sim(q, d_j) = 0$, otherwise

Drawbacks of the Boolean Model

- Retrieval based on binary decision criteria with no notion of partial matching
- No ranking of the documents is provided (absence of a grading scale)
- Information need has to be translated into a Boolean expression, which most users find awkward
- The Boolean queries formulated by the users are most often too simplistic
- As a consequence, the Boolean model frequently returns either too few or too many documents in response to a user query

Term Weighting

Term Weighting

- The terms of a document are not equally useful for describing the document contents
- In fact, there are index terms which are simply vaguer than others
- There are properties of an index term which are useful for evaluating the importance of the term in a document
  - For instance, a word which appears in all documents of a collection is completely useless for retrieval tasks

Term Weighting

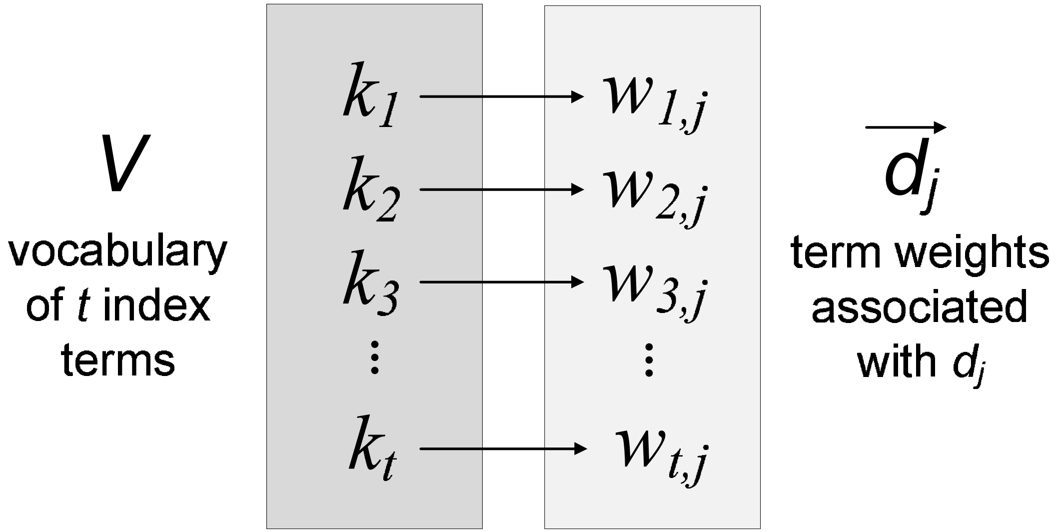
- To characterize term importance, we associate a weight $w_{i,j} > 0$ for each term k_i that occurs in the document d_j
  - If k_i that does not appear in the document d_j, then $w_{i,j} = 0$.
- The weight $w_{i,j}$ quantifies the importance of the index term k_i for describing the contents of d_j document
- These weights are useful to compute a numeric rank for each document in the collection with regard to a given query

Term Weighting

- For classic information retrieval models, the index term weights are assumed to be **mutually independent**
- This means that $w_{i,j}$ tells us nothing about $w_{i+1,j}$
- This is clearly a simplification because occurrences of index terms in a document are not uncorrelated
- This simplify term weighting computing and allows for fast ranking computation
- Furthermore, taking advantage of index term correlations is not a simple task

Term Weighting

- Let,
  - k_i an index term, and d_j a document
  - $V = \{k_1, k_2, ..., k_t\}$ the set of all index terms
  - $w_{i,j} \geqslant 0$ the weight associated with (k_i, d_j)
- Then we define $\vec{d_j} = (w_{1,j}, w_{2,j}, ..., w_{t,j})$ as a weighted vector that contains the weight $w_{i,j}$ of each term $k_i \in V$ in the document d_j



Term Weighting

- The weights $w_{i,j}$ are computed based on the **frequencies of occurrence** of the terms within documents
- Let $f_{i,j}$ be the frequency of occurrence of index term k_i in the document d_j
- Then we define the **total frequency of occurrence** F_i of term k_i in the collection as

$$F_i = \sum_{j=1}^{N} f_{i,j}$$

where N is the number of documents in the collection

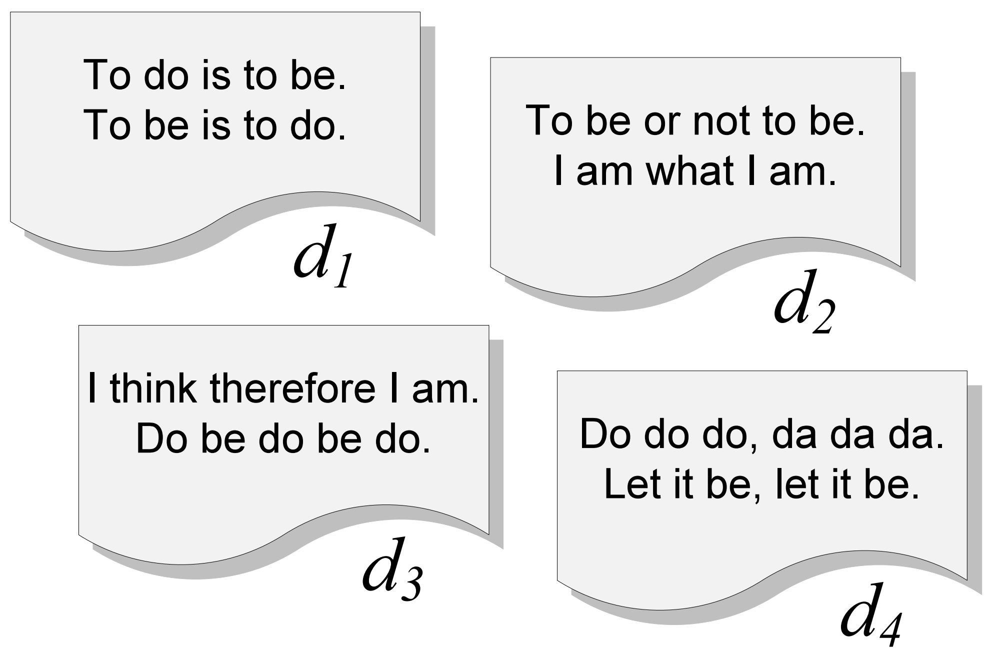
Term Weighting

- The **document frequency** n_i of a term k_i is the number of documents in which it occurs
  - Notice that $n_i \leq F_i$.
- For instance, in the document collection below, the values $f_{i,j}$, F_i and n_i associated to the term *do* are

$$f(do, d_1) = 2$$
$$f(do, d_2) = 0$$
$$f(do, d_3) = 3$$
$$f(do, d_4) = 3$$

$$F(do) = 8$$

$$n(do) = 3$$



Term Frequency (TF) Weights

- **Luhn Assumption**. The value of $w_{i,j}$ is proportional to the term frequency $f_{i,j}$
  - That is, the more often a term occurs in the text of the document, the higher its weight
- Based on the observation that high frequency terms are important for describing the key topics of a document
- This leads directly to the following tf weight formulation:

$$tf_{i,j} = f_{i,j}$$

Term Frequency (TF) Weights

- A variant of tf weight used in the literature is

$$tf_{i,j} = \begin{cases} 1 + \log f_{i,j} & \texttt{if} \;\; f_{i,j} > 0 \\ 0 & \texttt{otherwise} \end{cases}$$

  where the log is taken in base 2

- The log expression is a the preferred form because it makes them directly comparable to idf weights

Term Frequency (TF) Weights

- Log tf weights $tf_{i,j}$ for the example collection

To do is to be.
To be is to do.

d_1

To be or not to be.
I am what I am.

d_2

I think therefore I am.
Do be do be do.

d_3

Do do do, da da da.
Let it be, let it be.

d_4

Vocabulary		$tf_{i,1}$	$tf_{i,2}$	$tf_{i,3}$	$tf_{i,4}$
1	to	3	2	-	-
2	do	2	-	2.585	2.585
3	is	2	-	-	-
4	be	2	2	2	2
5	or	-	1	-	-
6	not	-	1	-	-
7	I	-	2	2	-
8	am	-	2	1	-
9	what	-	1	-	-
10	think	-	-	1	-
11	therefore	-	-	1	-
12	da	-	-	-	2.585
13	let	-	-	-	2
14	it	-	-	-	2

Inverse Document Frequency (IDF)

- We call **document exhaustivity** the number of index terms assigned to a document
- The more index terms are assigned to a document, the higher is the probability of retrieval for that document (by a random query)
  - If too many terms are assigned to a document, it will be retrieved by queries for which it is not relevant
- **Optimal exhaustivity**. A approach to circumvent this problem is to optimize the average number of index terms per document
- Another approach is by weighting the terms of a same document differently, by exploring the notion of **term specificity**

Inverse Document Frequency (IDF)

- **Specificity** is a property of the term semantics
  - A term is more or less specific depending on its meaning
  - To exemplify, the term *beverage* is less specific than the terms *tea* and *beer*
  - We could expect that the term *beverage* occurs in more documents than the terms *tea* and *beer*
- Term specificity should be interpreted as a statistical rather than semantic property of the term
- **Statistical term specificity**. Specificity of a term k_i can be quantified as the inverse of the number of documents in which k_i occurs

Inverse Document Frequency (IDF)

- Terms are distributed in a text according to Zipf's Law
- Thus, if we sort the vocabulary terms in decreasing order of document frequencies, and let $n(r)$ refer to the rth largest document frequency, we have

$$n(r) \sim r^{-\alpha}$$

  where α is an empirical constant
- That is, the document frequency of term k_i is an exponential function of its rank.

$$n(r) = Cr^{-\alpha}$$

  where C is a second empirical constant

Inverse Document Frequency (IDF)

- For English, $\alpha = 1$ provides a simple approximation. Setting $\alpha = 1$ and taking logs we have

$$\log n(r) = \log C - \log r$$

- For $r = 1$, we have $C = n(1)$, i.e., the value of C is the largest document frequency and works as a normalization constant
- An alternative is to do the normalization assuming $C = N$, where N is the number of documents in the collection

$$\log r \sim \log N - \log n(r)$$

Inverse Document Frequency (IDF)

- Let k_i be the term with the rth largest document frequency, i.e., $n(r) = n_i$. Then,

$$idf_i = \log \frac{N}{n_i}$$

  where idf_i is called the **inverse document frequency** of term k_i

- Idf weights provide a foundation for modern term weighting schemes and are used by almost any IR system of today

Inverse Document Frequency (IDF)

- Idf values for example collection

To do is to be.
To be is to do.

d_1

To be or not to be.
I am what I am.

d_2

I think therefore I am.
Do be do be do.

d_3

Do do do, da da da.
Let it be, let it be.

d_4

	term	n_i	$idf_i = \log(N/n_i)$
1	to	2	1
2	do	3	0.415
3	is	1	2
4	be	4	0
5	or	1	2
6	not	1	2
7	I	2	1
8	am	2	1
9	what	1	2
10	think	1	2
11	therefore	1	2
12	da	1	2
13	let	1	2
14	it	1	2

tf-idf weighting scheme

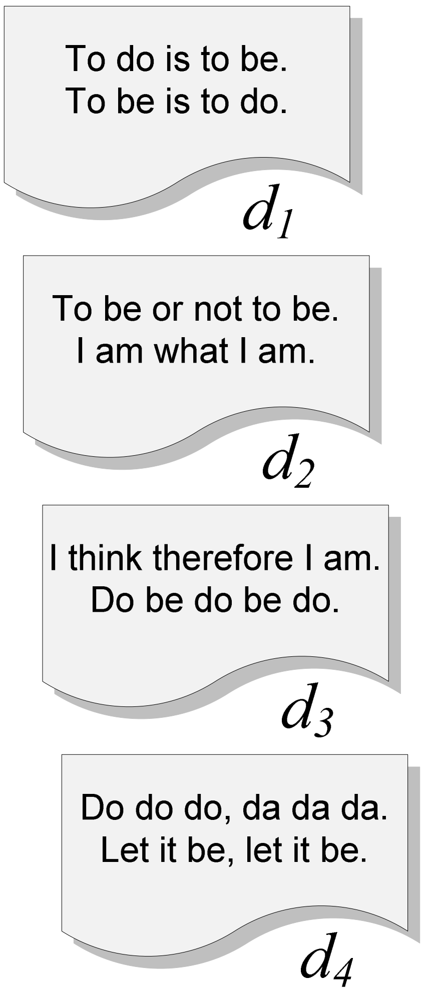
- The best known term weighting schemes use weights that combine idf factors with term frequencies
- Let $w_{i,j}$ be the term weight associated with the term k_i and the document d_j
- Then, we define

$$w_{i,j} = \begin{cases} (1 + \log f_{i,j}) \times \log \frac{N}{n_i} & \texttt{if} \quad f_{i,j} > 0 \\ 0 & \texttt{otherwise} \end{cases}$$

which is referred to as a **tf-idf weighting scheme**

tf-idf weighting scheme

- Tf-idf weights of all terms present in the example document collection



		d_1	d_2	d_3	d_4
1	to	3	2	-	-
2	do	0.830	-	1.073	1.073
3	is	4	-	-	-
4	be	-	-	-	-
5	or	-	2	-	-
6	not	-	2	-	-
7	I	-	2	2	-
8	am	-	2	1	-
9	what	-	2	-	-
10	think	-	-	2	-
11	therefore	-	-	2	-
12	da	-	-	-	5.170
13	let	-	-	-	4
14	it	-	-	-	4

Variants of TF-IDF

- Several variations of the above expression for tf-idf weights are described in the literature
- For tf weights, five distinct variants are illustrated below

	tf weight
binary	{0,1}
raw frequency	$f_{i,j}$
log normalization	$1 + \log f_{i,j}$
double normalization 0.5	$0.5 + 0.5 \frac{f_{i,j}}{max_i f_{i,j}}$
double normalization K	$K + (1 - K) \frac{f_{i,j}}{max_i f_{i,j}}$

Variants of TF-IDF

- Five distinct variants of idf weight

	idf weight
unary	1
inverse frequency	$\log \frac{N}{n_i}$
inv frequency smooth	$\log(1 + \frac{N}{n_i})$
inv frequeny max	$\log(1 + \frac{max_i n_i}{n_i})$
probabilistic inv frequency	$\log \frac{N-n_i}{n_i}$

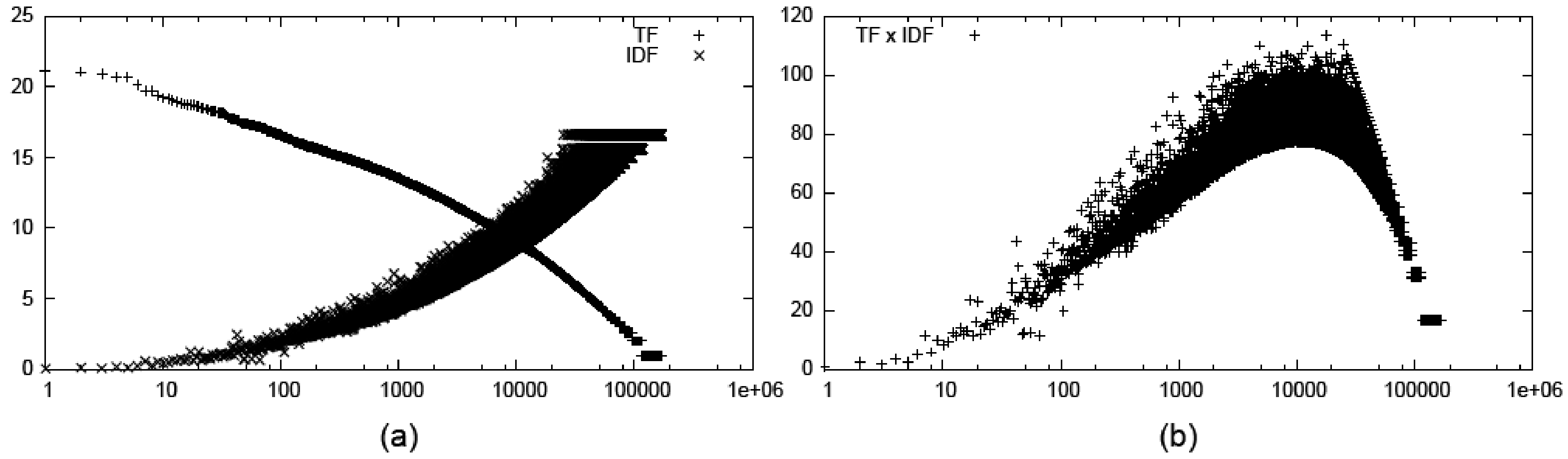
Variants of TF-IDF

- Recommended tf-idf weighting schemes

weighting scheme	document term weight	query term weight
1	$f_{i,j} * \log \frac{N}{n_i}$	$(0.5 + 0.5 \frac{f_{i,q}}{max_i \ f_{i,q}}) * \log \frac{N}{n_i}$
2	$1 + \log f_{i,j}$	$\log(1 + \frac{N}{n_i})$
3	$(1 + \log f_{i,j}) * \log \frac{N}{n_i}$	$(1 + \log f_{i,q}) * \log \frac{N}{n_i}$

TF-IDF Properties

- Tf, idf, and tf-idf weights for the *Wall Street Journal* reference collection sorted by decreasing tf weights
- To represent the tf weight in the graph, we sum the term frequencies accross all documents
  - That is, we used the term collection frequency F_i
- Plotted in logarithmic scale



TF-IDF Properties

- Weights used on graph:

$$tf_i \;=\; 1 + \log \sum_{j=1}^{N} f_{i,j} \qquad\qquad idf_i \;=\; \log \frac{N}{n_i}$$

- We observe that tf and idf weights present power-law behaviors that balance each other
- The terms of intermediate idf values display maximum tf-idf weights

Document Length Normalization

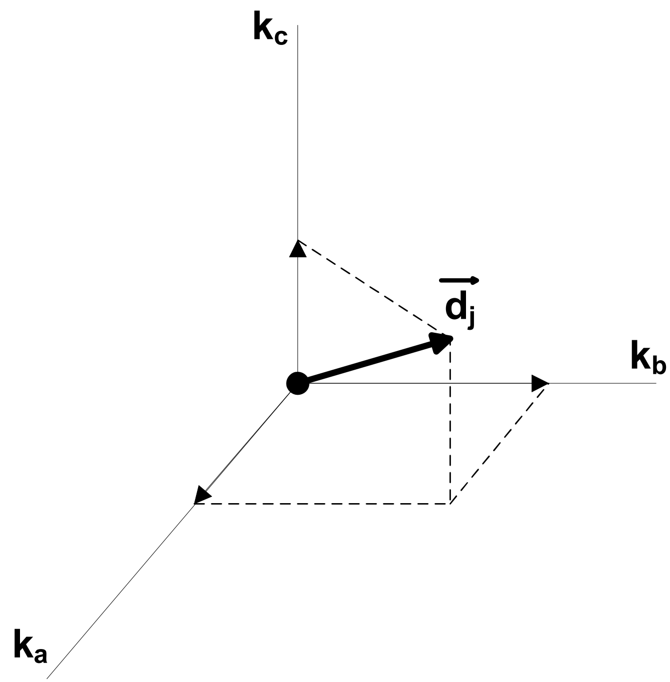
- Document sizes might vary widely
- This is a problem because longer documents are more likely to be retrieved by a given query
- To compensate for this undesired effect, we can divide the rank of each document by its length
- This procedure consistently leads to better ranking, and it is called **document length normalization**

Document Length Normalization

- Document length normalization can be done in different ways depending on the representation adopted for the documents:
  - **Size in bytes**: consider that each document is represented simply as a stream of bytes
  - **Number of words**: each document is represented as a single string, and the document length is the number of words in it
  - **Vector norms**: documents are represented as vectors of weighted terms

Document Length Normalization

- Documents represented as vectors of weighted terms
  - Each term of a collection is associated with an orthonormal unit vector $\vec{k}_i$ in a t-dimensional space
  - For each term k_i of a document d_j is associated the term vector component $w_{i,j} \times \vec{k}_i$



Document Length Normalization

- The document representation $\vec{d_j}$ is a vector composed of all its term vector components

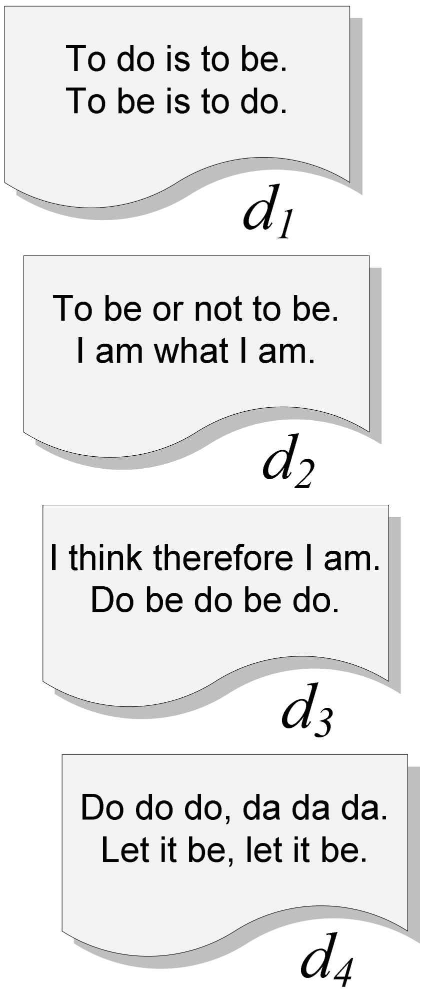
$$\vec{d_j} = (w_{1,j}, w_{2,j}, ..., w_{t,j})$$

- The document length is given by the norm of this vector, which is computed as follows

$$|\vec{d_j}| = \sqrt{\sum_{i}^{t} w_{i,j}^2}$$

Document Length Normalization

- Three variants of document lengths for the example collection



	d_1	d_2	d_3	d_4
size in bytes	34	37	41	43
number of words	10	11	10	12
vector norm	5.068	4.899	3.762	7.738

The Vector Model

The Vector Model

- Boolean matching and binary weights is too limiting
- In Vector model, **non-binary** term weights provide consideration for partial matches
- Term weights are used to compute a **degree of similarity** between a query and each document
- The documents are **ranked** in decreasing order of their degree of similarity

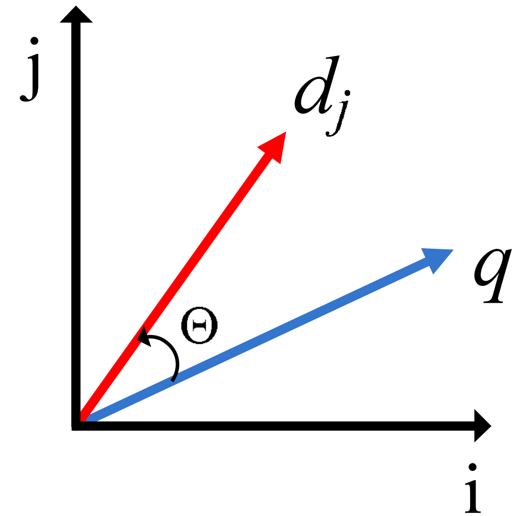
The Vector Model

- For the vector model:
  - The weight $w_{i,j}$ associated with a pair (k_i, d_j) is positive and non-binary
  - The index terms are assumed to be all mutually independent
  - They are represented as unit vectors of a t-dimensionsal space (t is the total number of index terms)
  - The representations of document d_j and query q are t-dimensional vectors given by

$$\vec{d_j} = (w_{1j}, w_{2j}, \ldots, w_{tj})$$
$$\vec{d_q} = (w_{1q}, w_{2q}, \ldots, w_{tq})$$

The Vector Model

- Similarity



$$sim(d_j, q) = cos(\theta) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{j=1}^{t} w_{i,q}^2}}$$

- Since $w_{ij} > 0$ and $w_{iq} > 0$ then $0 \leqslant sim(d_j, q) \leqslant 1$
- A document is retrieved even if it matches the query terms only partially

The Vector Model

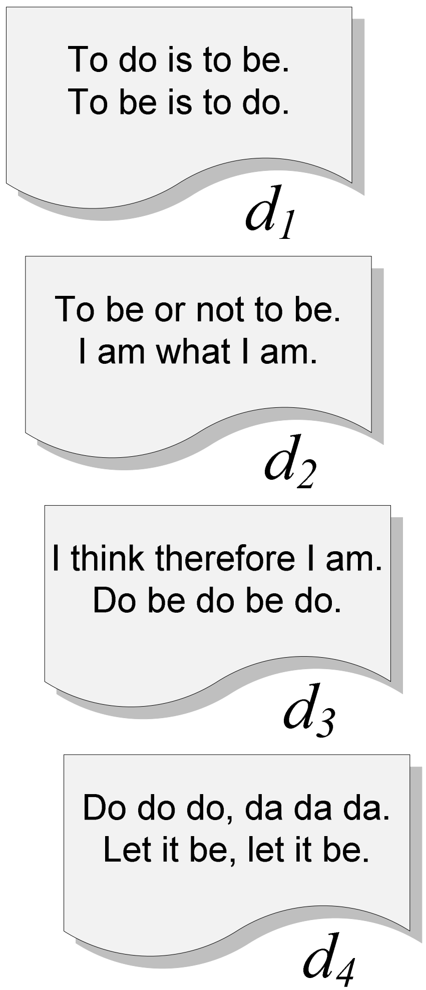
- Weights in the Vector model are basically tf-idf weights

$$
\begin{aligned}
w_{i,q} &= (1 + \log f_{i,q}) \times \log \frac{N}{n_i} \\
w_{i,j} &= (1 + \log f_{i,j}) \times \log \frac{N}{n_i}
\end{aligned}
$$

- These equations should only be applied for values of term frequency greater than zero
- If the term frequency is zero, the respective weight is also zero

The Vector Model

- Document ranks computed by the Vector model for the query “to do”



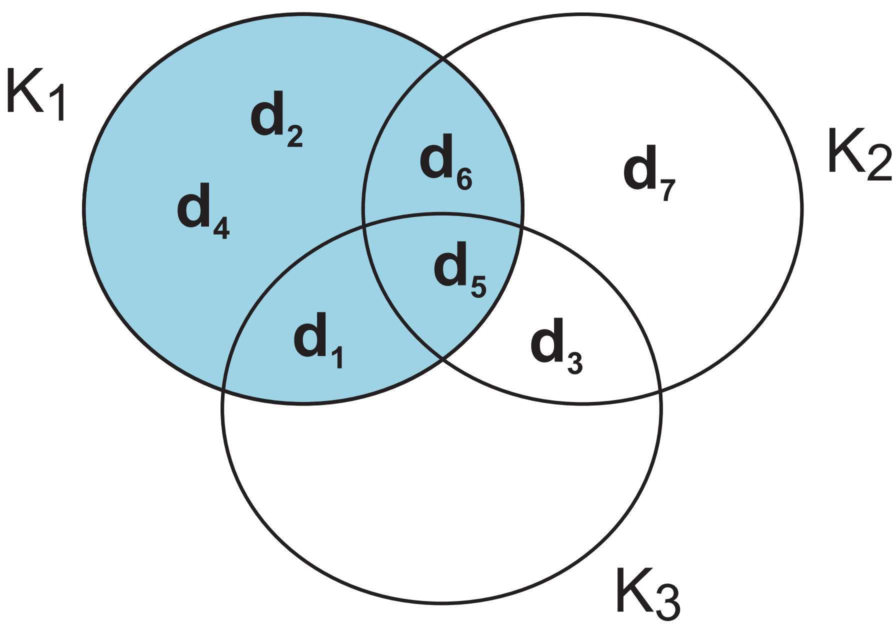
doc	rank computation	rank
d_1	$\frac{1*3+0.415*0.830}{5.068}$	0.660
d_2	$\frac{1*2+0.415*0}{4.899}$	0.408
d_3	$\frac{1*0+0.415*1.073}{3.762}$	0.118
d_4	$\frac{1*0+0.415*1.073}{7.738}$	0.058

The Vector Model

- Advantages:
  - term-weighting improves quality of the answer set
  - partial matching allows retrieval of docs that approximate the query conditions
  - cosine ranking formula sorts documents according to degree of similarity to the query
  - document length normalization is naturally built-in into the ranking
- Disadvantages:
  - assumes independence of index terms

The Vector Model

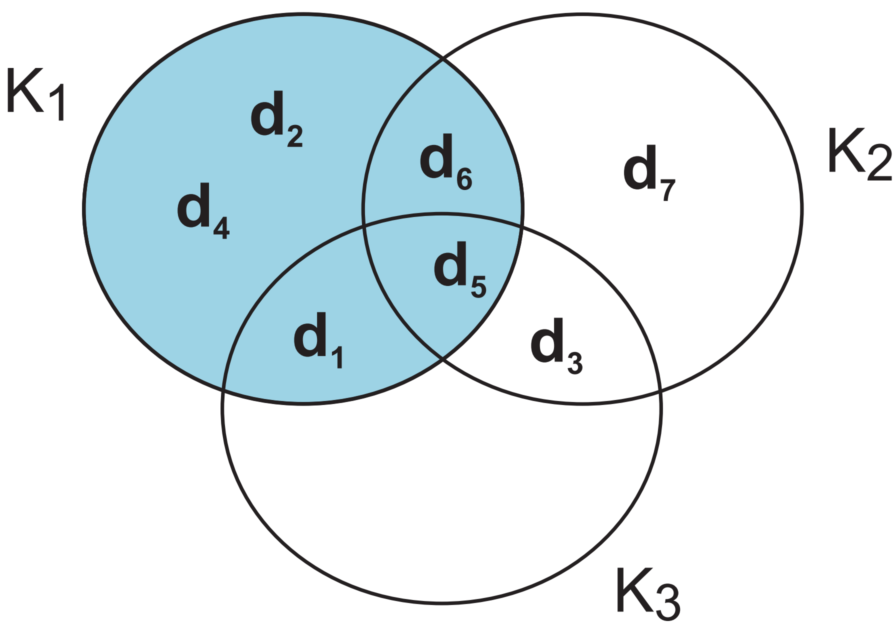
- Example 1



	K_1	K_2	K_3	$q \bullet d_j$
d_1	1	0	1	2
d_2	1	0	0	1
d_3	0	1	1	2
d_4	1	0	0	1
d_5	1	1	1	3
d_6	1	1	0	2
d_7	0	1	0	1
q	1	1	1	

The Vector Model

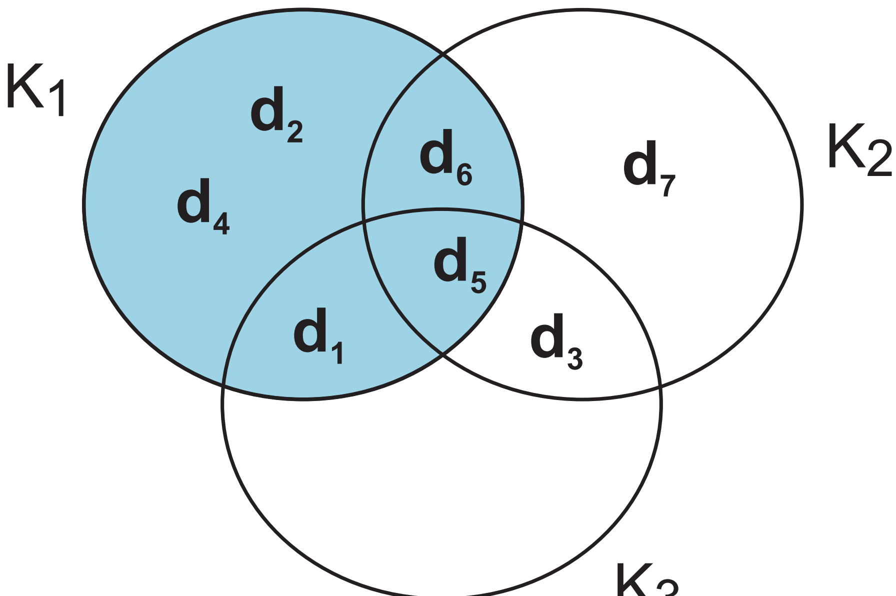
- Example 2



	K_1	K_2	K_3	$q \bullet d_j$
d_1	1	0	1	4
d_2	1	0	0	1
d_3	0	1	1	5
d_4	1	0	0	1
d_5	1	1	1	6
d_6	1	1	0	3
d_7	0	1	0	2
q	1	2	3	

The Vector Model

- Example 3



	K_1	K_2	K_3	$q \bullet d_j$
d_1	2	0	1	5
d_2	1	0	0	1
d_3	0	1	3	11
d_4	2	0	0	2
d_5	1	2	4	17
d_6	1	2	0	5
d_7	0	5	0	10
q	1	2	3	

Probabilistic Model

Probabilistic Model

- Objective: to capture the IR problem using a probabilistic framework
- Given a user query, there is an ideal answer set
- Querying as specification of the properties of this ideal answer set (clustering)
- But, what are these properties?
- Guess at the beginning what they could be (i.e., guess initial description of ideal answer set)
- Improve by iteration

Probabilistic Model

- An initial set of documents is retrieved somehow
- User inspects these docs looking for the relevant ones (in truth, only top 10-20 need to be inspected)
- IR system uses this information to refine description of ideal answer set
- By repeating this process, it is expected that the description of the ideal answer set will improve
- Have always in mind the need to guess at the very beginning the description of the ideal answer set
- Description of ideal answer set is modeled in probabilistic terms

Probabilistic Ranking Principle

- The probabilistic model
  - Tries to estimate the probability that a document will be relevante to a user query
  - Assumes that this probability depends on the query and the document representations only
  - Ideal answer set is referred to as R and should maximize the probability of relevance
- But,
  - how to compute these probabilities?
  - what is the sample space?

The Ranking

- Let,
  - $\overline{R}$ be the set of non-relevant documents
  - $P(R|\vec{d}_j)$ be the probability that the document d_j is relevant to the query q
  - $P(\overline{R}|\vec{d}_j)$ be the probability that the document d_j is non-relevant to q
- The similarity $sim(d_j, q)$ can be defined as

$$sim(d_j, q) = \frac{P(R|\vec{d}_j)}{P(\overline{R}|\vec{d}_j)}$$

The Ranking

- Using Bayes' rule,

$$sim(d_j, q) = \frac{P(\vec{d_j}|R, q) \times P(R, q)}{P(\vec{d_j}|\overline{R}, q) \times P(\overline{R}, q)} \sim \frac{P(\vec{d_j}|R, q)}{P(\vec{d_j}|\overline{R}, q)}$$

  where

  - $P(\vec{d_j}|R, q)$: probability of randomly selecting the document d_j from the set R of relevant documents (to query q)
  - $P(R, q)$: probability that a document randomly selected from the entire collection is relevant to query q
  - $P(\vec{d_j}|\overline{R}, q)$ and $P(\overline{R}, q)$: analogous and complementary

The Ranking

- Assuming that the weights $w_{i,j}$ are binary values and assuming the independence among the index terms:

$$sim(d_j, q) \sim \frac{(\prod_{k_i|w_{i,j}=1} P(k_i|R,q)) \times (\prod_{k_i|w_{i,j}=0} P(\overline{k}_i|R,q))}{(\prod_{k_i|w_{i,j}=1} P(k_i|\overline{R},q)) \times (\prod_{k_i|w_{i,j}=0} P(\overline{k}_i|\overline{R},q))}$$

  where

  - $P(k_i|R,q)$: probability that the term k_i is present in a document randomly selected from the set R of relevant documents (to the query q)
  - $P(\overline{k}_i|R,q)$: probability that k_i is not present in a document randomly selected from the set R
  - probabilities with $\overline{R}$: analogous to the ones just described

The Ranking

- To simplify our notation, let us adopt the following conventions
  - $p_{iR} = P(k_i|R, q)$
  - $q_{iR} = P(k_i|\overline{R}, q)$
- Since
  - $P(k_i|R, q) + P(\overline{k}_i|R, q) = 1$
  - $P(k_i|\overline{R}, q) + P(\overline{k}_i|\overline{R}, q) = 1$

  then we can write:

$$sim(d_j, q) \sim \frac{(\prod_{k_i|w_{i,j}=1} p_{iR}) \times (\prod_{k_i|w_{i,j}=0}(1 - p_{iR}))}{(\prod_{k_i|w_{i,j}=1} q_{iR}) \times (\prod_{k_i|w_{i,j}=0}(1 - q_{iR}))}$$

The Ranking

- Taking logarithms, we write

$$\begin{aligned} sim(d_j, q) \sim \; & \log \prod_{k_i | w_{i,j}=1} p_{iR} + \log \prod_{k_i | w_{i,j}=0} (1 - p_{iR}) \\ & - \log \prod_{k_i | w_{i,j}=1} q_{iR} - \log \prod_{k_i | w_{i,j}=0} (1 - q_{iR}) \end{aligned}$$

The Ranking

- Summing up terms that cancel each other, we obtain

$$
\begin{aligned}
sim(d_j, q) \;\sim\; & \log \prod_{k_i|w_{i,j}=1} p_{iR} + \log \prod_{k_i|w_{i,j}=0} (1 - p_{ir}) \\
& - \log \prod_{k_i|w_{i,j}=1} (1 - p_{ir}) + \log \prod_{k_i|w_{i,j}=1} (1 - p_{ir}) \\
& - \log \prod_{k_i|w_{i,j}=1} q_{iR} - \log \prod_{k_i|w_{i,j}=0} (1 - q_{iR}) \\
& + \log \prod_{k_i|w_{i,j}=1} (1 - q_{iR}) - \log \prod_{k_i|w_{i,j}=1} (1 - q_{iR})
\end{aligned}
$$

The Ranking

- Using logarithm operations, we obtain

$$
\begin{aligned}
sim(d_j, q) \quad \sim \quad & \log \prod_{k_i | w_{i,j}=1} \frac{p_{iR}}{(1 - p_{iR})} + \log \prod_{k_i} (1 - p_{iR}) \\
& + \log \prod_{k_i | w_{i,j}=1} \frac{(1 - q_{iR})}{q_{iR}} - \log \prod_{k_i} (1 - q_{iR})
\end{aligned}
$$

- Notice that there are two terms that are constants for *all* index terms, and can be disregarded for the purpose of ranking

The Ranking

- Assuming that
  - $\forall\ k_i \notin q,\ \ p_{iR} = q_{iR}$

  and converting the log products into sums of logs, we finally obtain

$$sim(d_j, q) \ \sim \ \sum_{k_i \in q \wedge k_i \in d_j} \left(\log \frac{p_{iR}}{1 - p_{iR}} + \log \frac{1 - q_{iR}}{q_{iR}} \right)$$

  which is a key expression for ranking computation in the probabilistic model

Term Incidence Contingency Table

- Let,
  - N be the number of document in the collection
  - n_i the number of documents that contain term k_i
  - R be the total number of relevant documents to query q
  - r_i be the number of relevant documents that contain term k_i
- Based in these values, we can build the following contingency table

	relevant	non-relevant	all docs
docs that contain k_i	r_i	$n_i - r_i$	n_i
docs that do not contain k_i	$R - r_i$	$N - n_i - (R - r_i)$	$N - n_i$
all docs	R	$N - R$	N

Term Incidence Contingency Table

- If the information of the values of the contingency table were available for any given query, we could write
  - $p_{iR} = \frac{r_i}{R}$
  - $q_{iR} = \frac{n_i - r_i}{N - R}$
- Then, the equation expression for ranking computation in the probabilistic model can be rewritten as

$$sim(d_j, q) \sim \sum_{k_i \in q \wedge k_i \in d_j} \log \frac{r_i(N - n_i - R + r_i)}{(R - r_i)(n_i - r_i)}$$

Term Incidence Contingency Table

- For the previous formula to work, we are still dependent on estimating what are the relevant documents for the query
- For handling small values of r_i, it is convenient to add 0.5 to each of the terms in the formula above, which yields

$$sim(d_j, q) \sim \sum_{k_i \in q \wedge k_i \in d_j} \log \frac{(r_i + 0.5)(N - n_i - R + r_i + 0.5)}{(R - r_i + 0.5)(n_i - r_i + 0.5)}$$

- This formula is known as the Robertson-Sparck Jones equation, and is considered as the classic ranking equation for the probabilistic model

Term Incidence Contingency Table

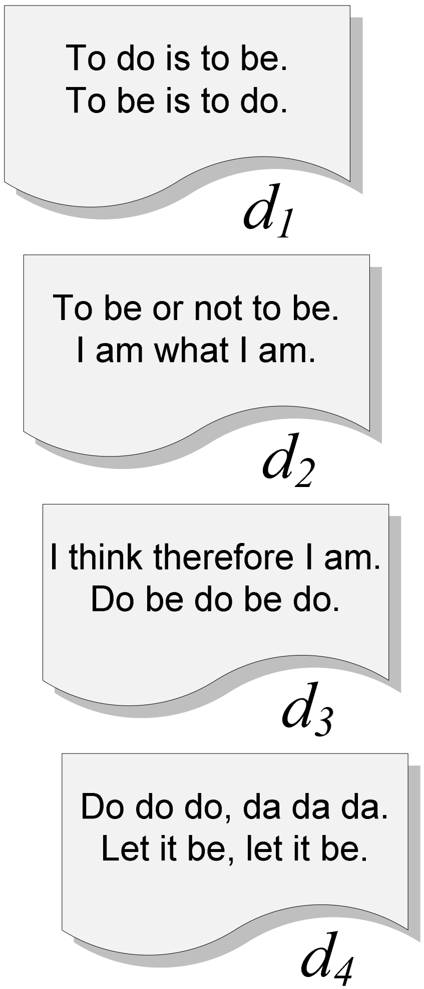
- The previous equation cannot be computed without estimates of r_i and R
- One possibility is to assume $R = r_i = 0$, as a way to boostrap the ranking equation, which leads to

$$sim(d_j, q) \sim \sum_{k_i \in q \wedge k_i \in d_j} \log \frac{N - n_i + 0.5}{n_i + 0.5}$$

- This equation provides an idf-like ranking formula
- In the absence of relevance information, this is the equation for ranking computation in the probabilistic model

Term Incidence Contingency Table

- Document ranks computed by the previous probabilistic ranking equation for the query “to do”



doc	rank computation	rank
d_1	$\log \frac{4-2+0.5}{2+0.5} + \log \frac{4-3+0.5}{3+0.5}$	- 1.222
d_2	$\log \frac{4-2+0.5}{2+0.5}$	0
d_3	$\log \frac{4-3+0.5}{3+0.5}$	- 1.222
d_4	$\log \frac{4-3+0.5}{3+0.5}$	- 1.222

Term Incidence Contingency Table

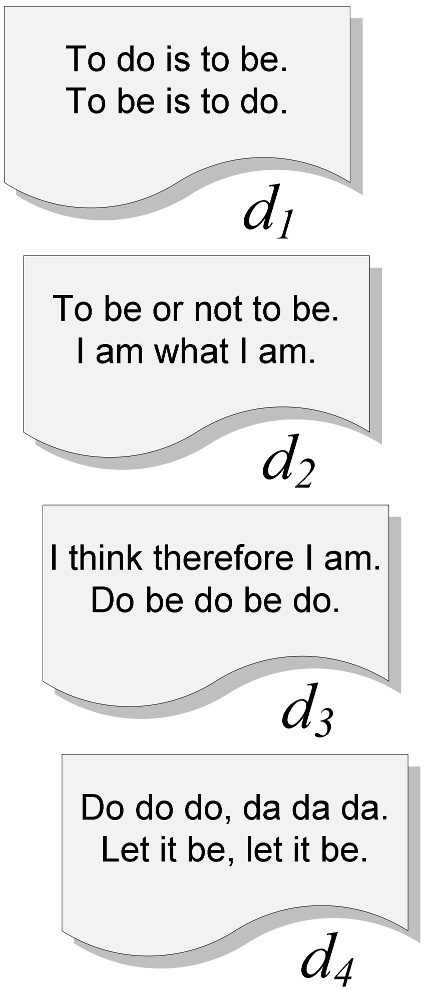
- The previous probabilistic ranking equation produced negative weights by the term “do”
- This equation produces negative terms whenever $n_i > N/2$
- One possible artifact to contain the effect of negative weights is to change the previous equation to:

$$sim(d_j, q) \sim \sum_{k_i \in q \wedge k_i \in d_j} \log \frac{N - \frac{n_i}{2} + 0.5}{\frac{n_i}{2} + 0.5}$$

- In this Equation, a term i that occurs in all documents ($n_i = N$) produces a weight equal to zero

Term Incidence Contingency Table

- Using this formulation, we redo the ranking computation for our example collection for the query “to do”



doc	rank computation	rank
d_1	$\log \frac{4-2/2+0.5}{2/2+0.5} + \log \frac{4-3/2+0.5}{3/2+0.5}$	1.807
d_2	$\log \frac{4-2/2+0.5}{2/2+0.5}$	1.222
d_3	$\log \frac{4-3/2+0.5}{3/2+0.5}$	0.585
d_4	$\log \frac{4-3/2+0.5}{3/2+0.5}$	0.585

Term Incidence Contingency Table

- Our examples above considered that $r_i = R = 0$
- An alternative is to estimate r_i and R performing an initial search:
  - select the top 10-20 ranked documents
  - inspect them to gather new estimates for r_i and R
  - remove the 10-20 documents used from the collection
  - rerun the query with the estimates obtained for r_i and R
- Unfortunately, procedures such as these require human intervention to initially select the relevant documents

Improving the Initial Ranking

- Consider the equation

$$sim(d_j, q) \sim \sum_{k_i \in q \wedge k_i \in d_j} \left(\log \frac{p_{iR}}{1 - p_{iR}} + \log \frac{1 - q_{iR}}{q_{iR}} \right)$$

- How obtain the probabilities p_{iR} and q_{iR} ?
- Estimates based on assumptions:
  - $p_{iR} = 0.5$
  - $q_{iR} = \frac{n_i}{N}$ where n_i is the number of docs that contain k_i
  - Use this initial guess to retrieve an initial ranking
  - Improve upon this initial ranking

Improving the Initial Ranking

$$sim(d_j, q) \sim \sum_{k_i \in q \wedge k_i \in d_j} \left(\log \frac{p_{iR}}{1 - p_{iR}} + \log \frac{1 - q_{iR}}{q_{iR}} \right)$$

- Let
  - V : set of docs initially retrieved
  - V_i : subset of docs retrieved that contain k_i
- Reevaluate estimates:
  - $p_{iR} = \frac{V_i}{V}$
  - $q_{iR} = \frac{n_i - V_i}{N - V}$
- Repeat recursively

Improving the Initial Ranking

$$sim(d_j, q) \sim \sum_{k_i \in q \wedge k_i \in d_j} \left(\log \frac{p_{iR}}{1 - p_{iR}} + \log \frac{1 - q_{iR}}{q_{iR}} \right)$$

- To avoid problems with $V = 1$ and $V_i = 0$:
  - $p_{iR} = \frac{V_i + 0.5}{V + 1}$
  - $q_{iR} = \frac{n_i - V_i + 0.5}{N - V + 1}$
- Also,
  - $p_{iR} = \frac{V_i + \frac{n_i}{N}}{V + 1}$
  - $q_{iR} = \frac{n_i - V_i + \frac{n_i}{N}}{N - V + 1}$

Pluses and Minuses

- Advantages:
  - Docs ranked in decreasing order of probability of relevance
- Disadvantages:
  - need to guess initial estimates for p_{iR}
  - method does not take into account tf factors
  - the lack of document length normalization

Comparison of Classic Models

- Boolean model does not provide for partial matches and is considered to be the weakest classic model
- Salton and Buckley did a series of experiments that indicate that, in general, the vector model outperforms the probabilistic model with general collections
- This seems also to be the view of the research community