Lecture 3. Classic IR Models

Basic Concepts

- Each document is represented by a set of representative keywords or index terms
- An index term is a word or group of consecutive words in a document
- A pre-selected set of index terms can be used to summarize the document contents
- However, it might be interesting to assume that all words are index terms (full text representation)

Basic Concepts

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

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

vocabulary of *t* index terms

Basic Concepts

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

$$V = \begin{array}{|ccccc|}\hline k_1 & k_2 & k_3 & \ldots & k_t \\ \hline\end{array}$$

$1 \quad 0 \quad 0 \ \ldots \ 0$	pattern that represents documents (and queries) with the term k_1 and no other
$\vdots$	
$1 \quad 1 \quad 1 \ \ldots \ 1$	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 $c(q)$)

Bag of words: word ordering is ignored

The Term-Document Matrix

- The occurrence of a term k_i in a document d_j establishes a relation between k_i and d_j
- A **term-document relation** between k_i and d_j can be quantified by the frequency of the term in the document
- In matrix form, this can written as

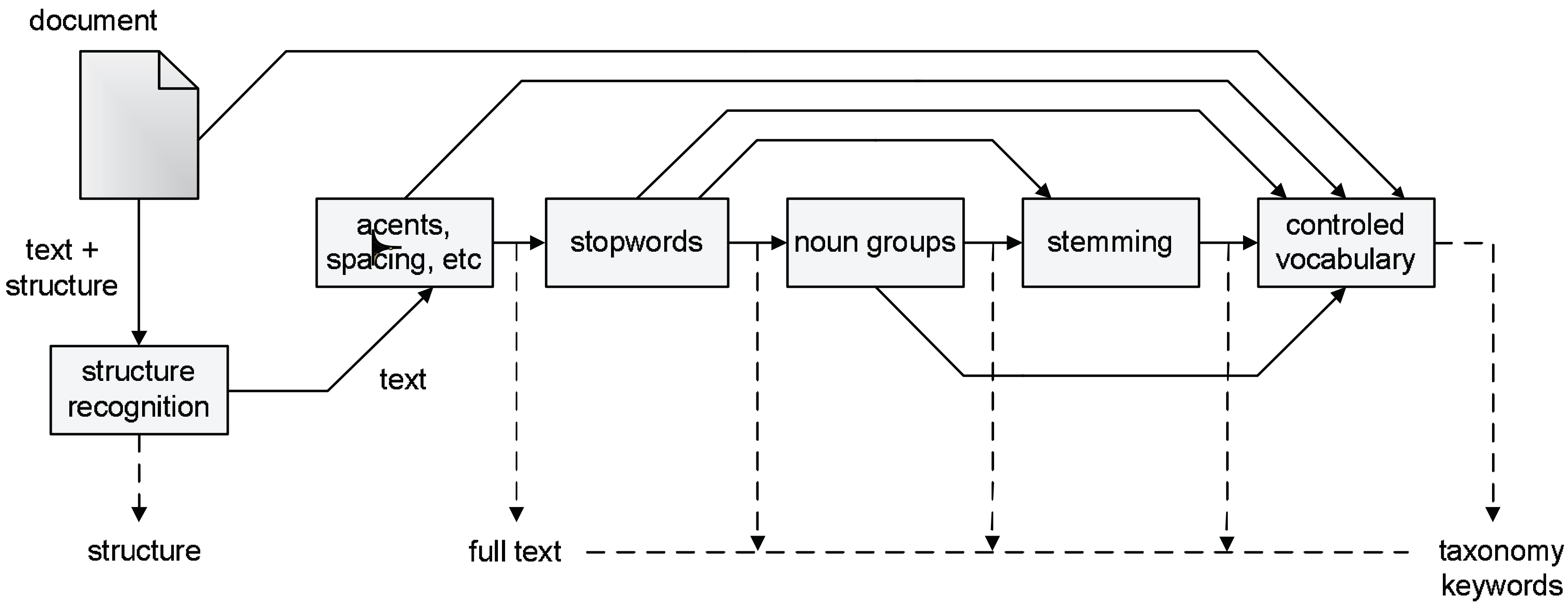
$$\begin{array}{cc} & \begin{array}{cc} d_1 & d_2 \end{array} \\ \begin{array}{c} k_1 \\ k_2 \\ k_3 \end{array} & \left[\begin{array}{cc} f_{1,1} & f_{1,2} \\ f_{2,1} & f_{2,2} \\ f_{3,1} & f_{3,2} \end{array}\right] \end{array}$$

where each $f_{i,j}$ element stands for the frequency of term k_i in document d_j

A simplistic approach

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

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

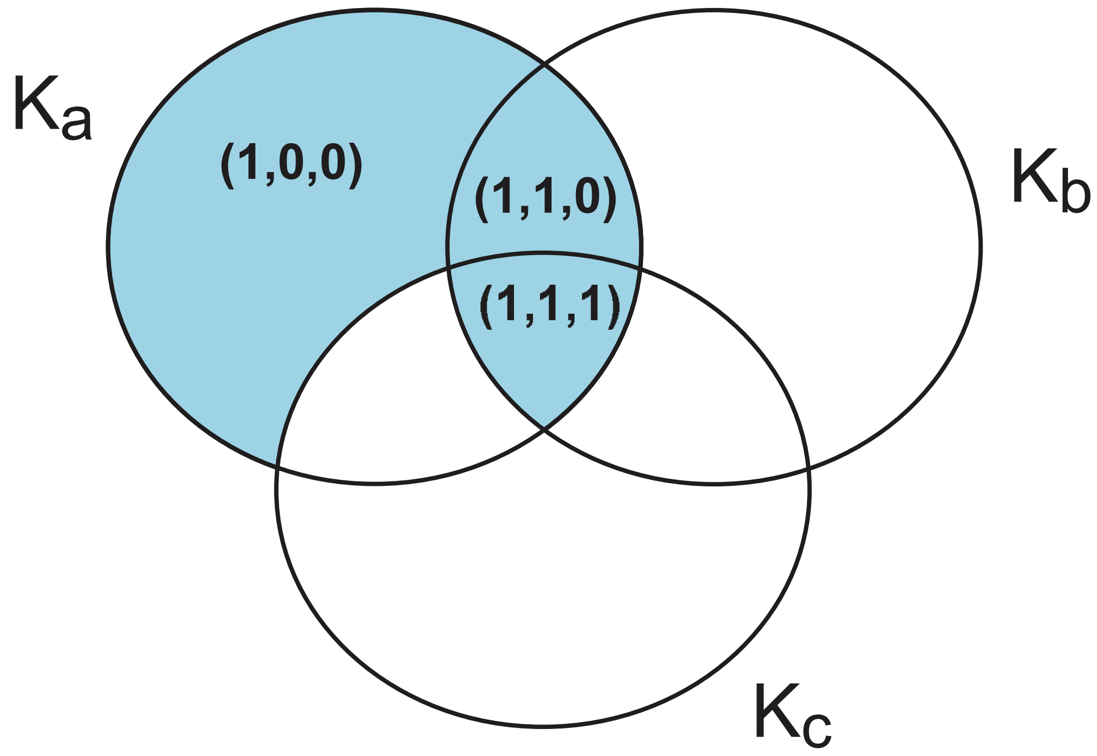
- Term-document frequencies in the term-document matrix are all binary
  - $w_{ij} \in \{0, 1\}$: weight associated with pair (k_i, d_j)
  - $w_{iq} \in \{0, 1\}$: weight associated with pair (k_i, q)

The Boolean Model

- A term conjunctive component that satisfies 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

- The three conjunctive components for the query $q = k_a \wedge (k_b \vee \neg k_c)$



The Boolean Model

- This approach works even if the vocabulary of the collection includes terms not in the query
- Consider that the vocabulary is given by $V = \{k_a, k_b, k_c, k_d\}$
- Then, a document d_j that contains only terms k_a, k_b, and k_c is represented by $c(d_j) = (1, 1, 1, 0)$
- The query $[q = k_a \ \wedge \ (k_b \ \vee \ \neg k_c)]$ is represented in disjunctive normal form as

$$
\begin{aligned}
q_{DNF} \quad = \quad & (1,1,1,0) \vee (1,1,1,1) \vee \\
& (1,1,0,0) \vee (1,1,0,1) \vee \\
& (1,0,0,0) \vee (1,0,0,1)
\end{aligned}
$$

The Boolean Model

- The similarity of the document d_j to the query q is defined as

$$sim(d_j, q) = \begin{cases} 1 & \text{if } \exists c(q) \mid c(q) = c(d_j) \\ 0 & \text{otherwise} \end{cases}$$

- The Boolean model predicts that each document is either relevant or non-relevant

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
- The model frequently returns either too few or too many documents in response to a user query

Natural language query??

Term Weighting

Term Weighting

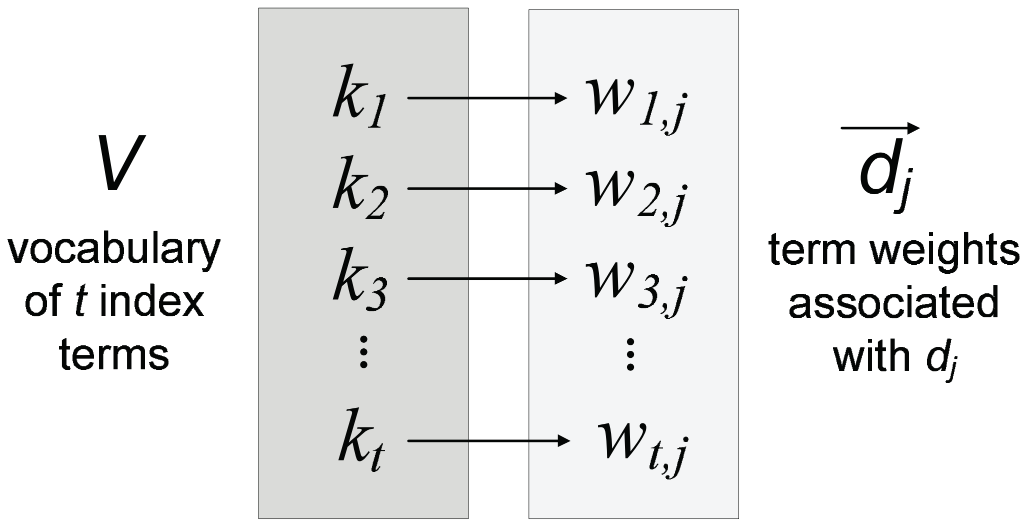
- The terms of a document are not equally useful for describing the document contents
- In fact, there are index terms which are simply vaguer than others
- There are properties of an index term which are useful for evaluating the importance of the term in a document
  - For instance, a word which appears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

Term Weighting

- To characterize term importance, we associate a weight $w_{i,j} > 0$ with 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 document d_j
- These weights are useful to compute a rank for each document in the collection with regard to a given query

Term Weighting

- Let,
  - k_i be an index term and d_j be a document
  - $V = \{k_1, k_2, ..., k_t\}$ be the set of all index terms
  - $w_{i,j} \geqslant 0$ be 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 $w_{i,j}$ can be computed using 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
- The **total frequency of occurrence** F_i of term k_i in the collection is defined as

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

where N is the number of documents in the collection

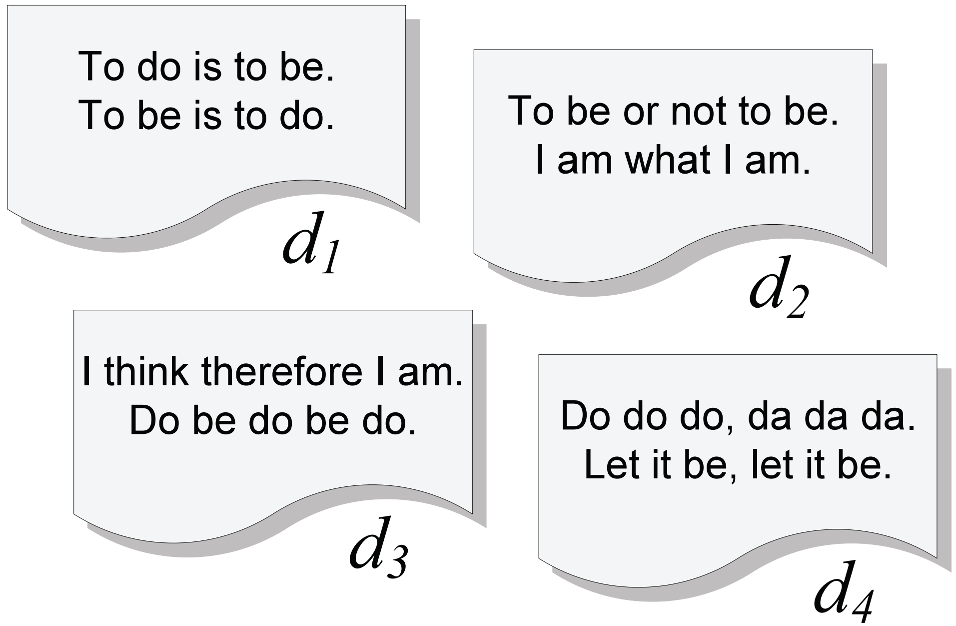
Term Weighting

- 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 with 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-term correlation matrix

- 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
- For instance, the terms **computer** and **network** tend to appear together in a document about **computer networks**
  - In this document, the appearance of one of these terms attracts the appearance of the other
  - Thus, they are correlated and their weights should reflect this correlation.

Term-term correlation matrix

- To take into account term-term correlations, we can compute a correlation matrix
- Let $\vec{M} = (m_{ij})$ be a term-document matrix $t \times N$ where $m_{ij} = w_{i,j}$
- The matrix $\vec{C} = \vec{M}\vec{M}^t$ is a term-term correlation matrix
- Each element $c_{u,v} \in \mathbf{C}$ expresses a correlation between terms k_u and k_v, given by

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

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

Co-occur on which level? Document, paragraph, sentence, or a fixed window

Term-term correlation matrix

- Term-term correlation matrix for a sample collection

$$
\begin{array}{cc}
 & \begin{array}{cc} d_1 & d_2 \end{array} \\
\begin{array}{c} k_1 \\ k_2 \\ k_3 \end{array} &
\left[\begin{array}{cc} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \end{array}\right]
\end{array}
\qquad
\begin{array}{cc}
 & \begin{array}{ccc} k_1 & k_2 & k_3 \end{array} \\
\begin{array}{c} d_1 \\ d_2 \end{array} &
\left[\begin{array}{ccc} w_{1,1} & w_{2,1} & w_{3,1} \\ w_{1,2} & w_{2,2} & w_{3,2} \end{array}\right]
\end{array}
$$

$$
\underbrace{\mathbf{M} \qquad \times \qquad \mathbf{M}^T}
$$

$$\Downarrow$$

$$
\begin{array}{cc}
 & \begin{array}{ccc} k_1 & k_2 & k_3 \end{array} \\
\begin{array}{c} k_1 \\ k_2 \\ k_3 \end{array} &
\left[\begin{array}{ccc}
w_{1,1}w_{1,1} + w_{1,2}w_{1,2} & w_{1,1}w_{2,1} + w_{1,2}w_{2,2} & w_{1,1}w_{3,1} + w_{1,2}w_{3,2} \\
w_{2,1}w_{1,1} + w_{2,2}w_{1,2} & w_{2,1}w_{2,1} + w_{2,2}w_{2,2} & w_{2,1}w_{3,1} + w_{2,2}w_{3,2} \\
w_{3,1}w_{1,1} + w_{3,2}w_{1,2} & w_{3,1}w_{2,1} + w_{3,2}w_{2,2} & w_{3,1}w_{3,1} + w_{3,2}w_{3,2}
\end{array}\right]
\end{array}
$$

TF-IDF Weights

TF-IDF Weights

- TF-IDF term weighting scheme:
  - Term frequency (TF)
  - Inverse document frequency (IDF)
  - Foundations of the most popular term weighting scheme in IR

Term-term correlation matrix

- **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
- This is based on the observation that high frequency terms are important for describing documents
- Which 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

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

  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, as we later discuss

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

- 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
  - If too many terms are assigned to a document, it will be retrieved by queries for which it is not relevant
- **Optimal exhaustivity**. We can circumvent this problem by optimizing the number of terms per document
- Another approach is by weighting the terms differently, by exploring the notion of **term specificity**

Inverse Document Frequency

- **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**. The inverse of the number of documents in which the term occurs

Inverse Document Frequency

- Terms are distributed in a text according to Zipf's Law
- Thus, if we sort the vocabulary terms in decreasing order of document frequencies we have

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

  where $n(r)$ refer to the rth largest document frequency and 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

Word	Freq. (f)	Rank (r)	$f \cdot r$
the	3332	1	3332
and	2972	2	5944
a	1775	3	5235
he	877	10	8770
but	410	20	8400
be	294	30	8820
there	222	40	8880
one	172	50	8600
about	158	60	9480
more	138	70	9660
never	124	80	9920
Oh	116	90	10440
two	104	100	10400

Word	Freq. (f)	Rank (r)	$f \cdot r$
turned	51	200	10200
you'll	30	300	9000
name	21	400	8400
comes	16	500	8000
group	13	600	7800
lead	11	700	7700
friends	10	800	8000
begin	9	900	8100
family	8	1000	8000
brushed	4	2000	8000
sins	2	3000	6000
Could	2	4000	8000
Applausive	1	8000	8000

Table 1.3 Empirical evaluation of Zipf's law on *Tom Sawyer*.

50^{th} most common word should occur with three times the frequency of the 150^{th} most common word

Inverse Document Frequency

- Setting $\alpha = 1$ (simple approximation for english collections) 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
  - This value works as a normalization constant
- An alternative is to do the normalization assuming $C = N$, where N is the number of docs in the collection

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

Inverse Document Frequency

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

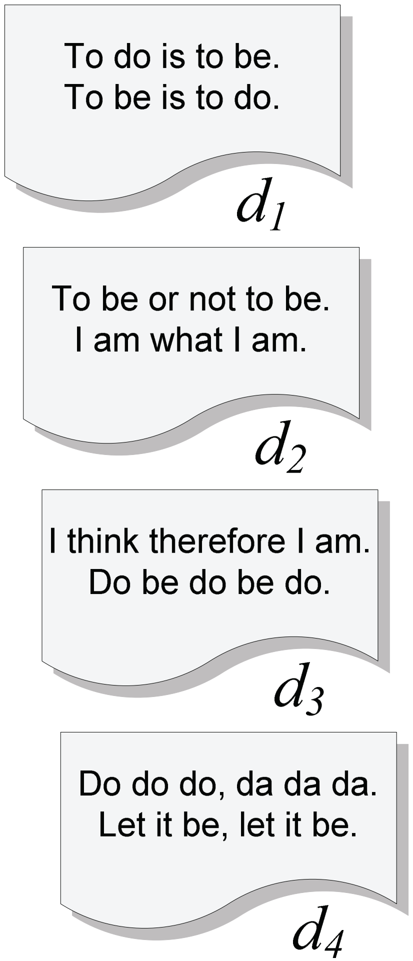
$$idf_i = \log \frac{N}{n_i}$$

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

- Idf provides a foundation for modern term weighting schemes and is used for ranking in almost all IR systems

Inverse Document Frequency

- Idf values for example collection



	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

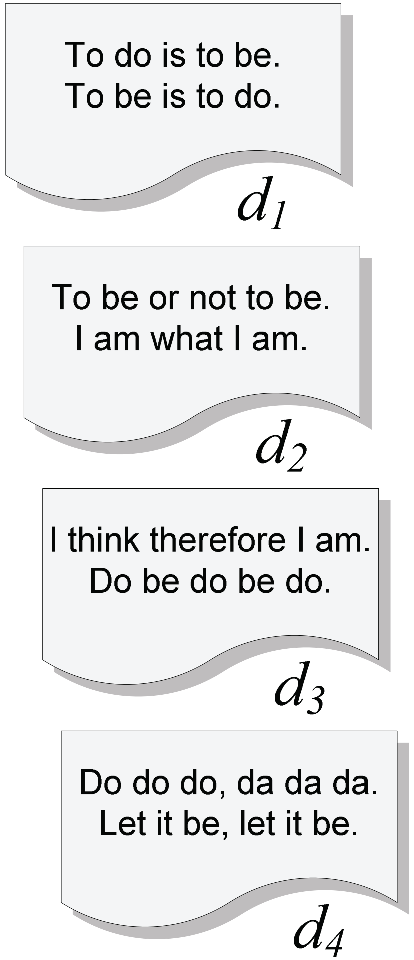
- The best known term weighting schemes use weights that combine idf factors with term frequencies
- Let $w_{i,j}$ be the term weight associated with the term k_i and the document d_j
- Then, we define

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

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

TF-IDF weighting scheme

- Tf-idf weights of all terms present in our 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	$(1 + \log f_{i,j}) * \log \frac{N}{n_i}$	$(1 + \log f_{i,q}) * \log \frac{N}{n_i}$

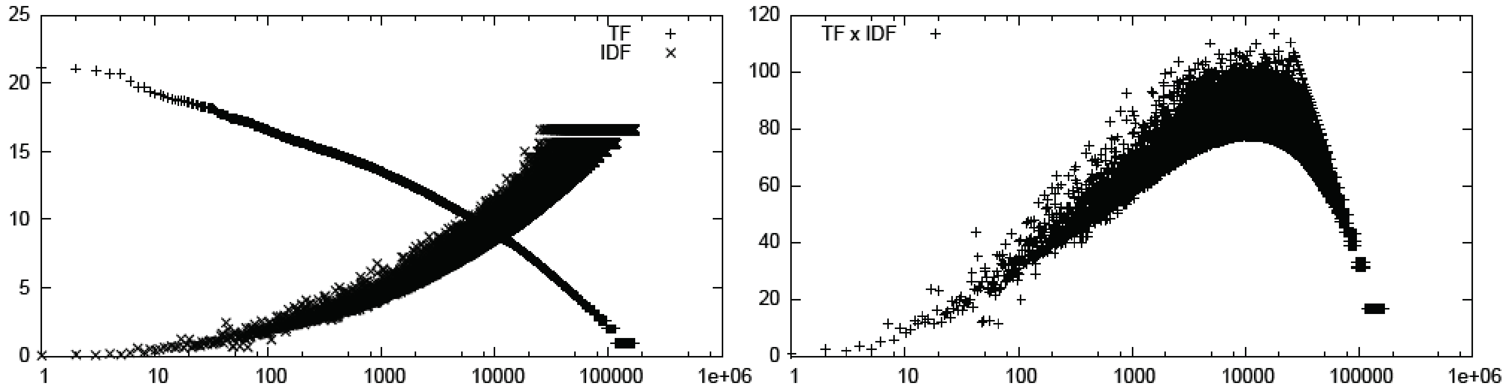
TF-IDF Properties

- Consider the tf, idf, and tf-idf weights for the *Wall Street Journal* reference collection
- To study their behavior, we would like to plot them together
- While idf is computed over all the collection, tf is computed on a per document basis. Thus, we need a representation of tf based on all the collection, which is provided by the term collection frequency F_i
- This reasoning leads to the following tf and idf term weights:

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

TF-IDF Properties

- Plotting tf and idf in logarithmic scale yields



common terms (a) rank rare terms (b) rank

- 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 and are most interesting for ranking

Document Length Normalization

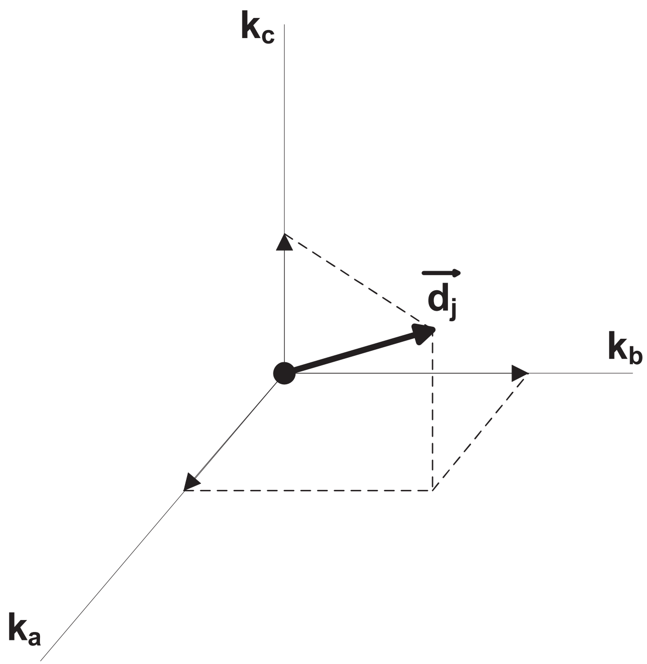
- Document sizes might vary widely
- This is a problem because longer documents are more likely to be retrieved by a given query
- To compensate for this undesired effect, we can divide the rank of each document by its length
- This procedure consistently leads to better ranking, and it is called **document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

Document Length Normalization

- Methods of document length normalization depend 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

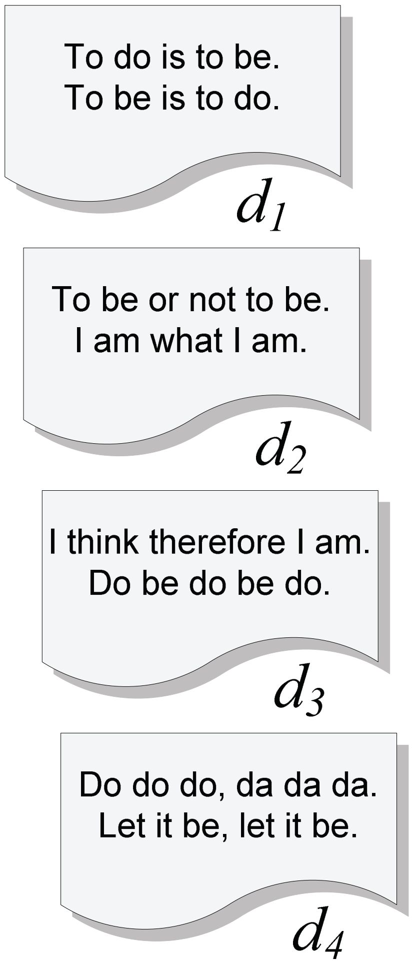
$$\vec{d_j} = (w_{1,j}, w_{2,j}, ..., w_{t,j})$$

- 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
- The vector model proposes a framework in which partial matching is possible
- This is accomplished by assigning non-binary weights to index terms in queries and in documents
- 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

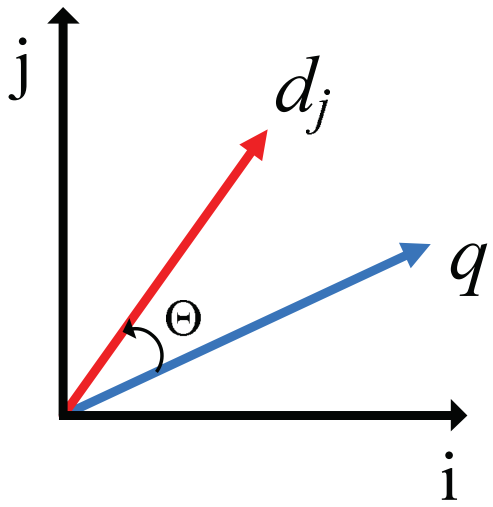
The Vector Model

- 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

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

The Vector Model

- Similarity between a document d_j and a query q



$$cos(\theta) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|}$$

$$\boxed{sim(d_j, q) = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{j=1}^{t} w_{i,q}^2}}}$$

Since $w_{ij} > 0$ and $w_{iq} > 0$, we have $0 \leqslant sim(d_j, q) \leqslant 1$

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” (see tf-idf weight values in Slide 43)

	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

		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

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

	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

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 a degree of similarity to the query
  - document length normalization is naturally built-in into the ranking
- Disadvantages:
  - It assumes independence of index terms

Probabilistic Model

Probabilistic Model

- The probabilistic model captures the IR problem using a probabilistic framework
- Given a user query, there is an **ideal answer set** for this query
- Given a description of this ideal answer set, we could retrieve the relevant documents
- Querying is seen as a specification of the **properties** of this ideal answer set
  - But, what are these properties?

Probabilistic Model

- An initial set of documents is retrieved somehow
- The user inspects these docs looking for the relevant ones (in truth, only top 10-20 need to be inspected)
- The IR system uses this information to refine the description of the ideal answer set
- By repeating this process, it is expected that the description of the ideal answer set will improve

Probabilistic Ranking Principle

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

The Ranking

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

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

$$\operatorname{sim}\left(\overrightarrow{d_j}, \vec{q}\right) = \frac{P(R|\overrightarrow{d_j}, \vec{q})}{P(\bar{R}|\overrightarrow{d_j}, \vec{q})}$$

By Bayes rule,

$$P(R|\overrightarrow{d_j}, \vec{q}) = \frac{P\left(\overrightarrow{d_j}\middle|R, \vec{q}\right)P(R|\vec{q})}{P(\overrightarrow{d_j}|\vec{q})}$$

$$P(\bar{R}|\overrightarrow{d_j}, \vec{q}) = \frac{P\left(\overrightarrow{d_j}\middle|\bar{R}, \vec{q}\right)P(\bar{R}|\vec{q})}{P(\overrightarrow{d_j}|\vec{q})}$$

$$\operatorname{sim}\left(\overrightarrow{d_j}, \vec{q}\right) = \frac{P(R|\overrightarrow{d_j}, \vec{q})}{P(\bar{R}|\overrightarrow{d_j}, \vec{q})} = \frac{\dfrac{P\left(\overrightarrow{d_j}\middle|R, \vec{q}\right)P(R|\vec{q})}{P(\overrightarrow{d_j}|\vec{q})}}{\dfrac{P\left(\overrightarrow{d_j}\middle|\bar{R}, \vec{q}\right)P(\bar{R}|\vec{q})}{P(\overrightarrow{d_j}|\vec{q})}} = \frac{P\left(\overrightarrow{d_j}\middle|R, \vec{q}\right)P(R|\vec{q})}{P\left(\overrightarrow{d_j}\middle|\bar{R}, \vec{q}\right)P(\bar{R}|\vec{q})}$$

$$= \frac{P\left(\overrightarrow{d_j}\middle|R, \vec{q}\right)P(R|\vec{q})}{P\left(\overrightarrow{d_j}\middle|\bar{R}, \vec{q}\right)P(\bar{R}|\vec{q})} = \frac{\dfrac{P\left(\overrightarrow{d_j}\middle|R, \vec{q}\right)P(R,\vec{q})}{P(\vec{q})}}{\dfrac{P\left(\overrightarrow{d_j}\middle|\bar{R}, \vec{q}\right)P(\bar{R},\vec{q})}{P(\vec{q})}} = \frac{P\left(\overrightarrow{d_j}\middle|R, \vec{q}\right)P(R,\vec{q})}{P\left(\overrightarrow{d_j}\middle|\bar{R}, \vec{q}\right)P(\bar{R},\vec{q})}$$

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
  - $P(R, q)$: 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 $\boxed{\text{all binary}}$ and assuming $\boxed{\text{independence}}$ 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
  - $P(\overline{k}_i|R,q)$: 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
  - $P(k_i|R, q) + P(\overline{k}_i|R, q) = 1$
  - $P(k_i|\overline{R}, q) + P(\overline{k}_i|\overline{R}, q) = 1$

  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

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

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

The Ranking

- Summing up terms that cancel each other, we obtain

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

The Ranking

- Using logarithm operations, we obtain

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

- Notice that two of the factors in the formula above are a function of all index terms and do not depend on document d_j. They are constants for a given query and can be disregarded for the purpose of ranking

The Ranking

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

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

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

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

Term Incidence Contingency Table

- Let,
  - N be the number of documents in the collection
  - n_i be 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 on these variables, 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

Ranking Formula

- If information on the contingency table were available for a given query, we could write
  - $p_{iR} = \frac{r_i}{R}$
  - $q_{iR} = \frac{n_i - r_i}{N - R}$

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

- Then, the equation for ranking computation in the probabilistic model could be rewritten as

$$sim(d_j, q) \sim \sum_{k_i[q,d_j]} \log\left(\frac{r_i}{R - r_i} \times \frac{N - n_i - R + r_i}{n_i - r_i}\right)$$

where $k_i[q, d_j]$ is a short notation for $k_i \in q \wedge k_i \in d_j$

	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

Ranking Formula

- In the previous formula, we are still dependent on an estimation of the relevant dos for the query
- For handling small values of r_i, we add 0.5 to each of the terms in the formula above, which changes $sim(d_j, q)$ into

$$\sum_{k_i[q,d_j]} \log \left(\frac{r_i + 0.5}{R - r_i + 0.5} \times \frac{N - n_i - R + r_i + 0.5}{n_i - r_i + 0.5} \right)$$

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

Ranking Formula

- The previous equation cannot be computed without estimates of r_i and R

$$\sum_{k_i[q,d_j]} \log \left(\frac{r_i + 0.5}{R - r_i + 0.5} \times \frac{N - n_i - R + r_i + 0.5}{n_i - r_i + 0.5} \right)$$

- One possibility is to assume $R = r_i = 0$, as a way to boostrap the ranking equation, which leads to

	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	$\log \frac{N - n_i}{n_i}$

$$\boxed{sim(d_j, q) \sim \sum_{k_i[q,d_j]} \log \left(\frac{N - n_i + 0.5}{n_i + 0.5} \right)}$$

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

Ranking Example

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

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

$$sim(d_j, q) \sim \sum_{k_i[q,d_j]} \log\left(\frac{N-n_i+0.5}{n_i+0.5}\right)$$

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

Ranking Example

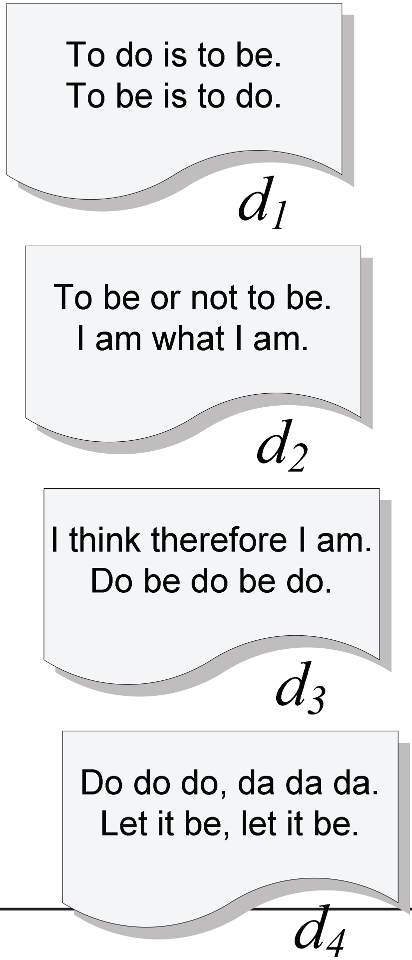
- The ranking computation led to negative weights because of the term “do”
- Actually, the probabilistic ranking equation produces negative terms whenever $n_i > N/2$
- One possible artifact to contain the effect of negative weights is to change the previous equation to:

$$sim(d_j, q) \sim \sum_{k_i[q,d_j]} \log \left(\frac{N + 0.5}{n_i + 0.5} \right)$$

- By doing so, a term that occurs in all documents ($n_i = N$) produces a weight equal to zero

Ranking Example

- Using this latest formulation, we redo the ranking computation for our example collection for the query "to do" and obtain



doc	rank computation	rank
d_1	$\log \frac{4+0.5}{2+0.5} + \log \frac{4+0.5}{3+0.5}$	1.210
d_2	$\log \frac{4+0.5}{2+0.5}$	0.847
d_3	$\log \frac{4+0.5}{3+0.5}$	0.362
d_4	$\log \frac{4+0.5}{3+0.5}$	0.362

Estimaging r_i and R

- 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

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

- 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

- Substituting p_{iR} and q_{iR} into the previous Equation, we obtain:

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

- That is the equation used when no relevance information is provided, without the 0.5 correction factor
- Given this initial guess, we can provide an initial probabilistic ranking
- After that, we can attempt to improve this initial ranking as follows

Improving the Initial Ranking

- We can attempt to improve this initial ranking as follows
- Let
  - D : set of docs initially retrieved
  - D_i : subset of docs retrieved that contain k_i
- Reevaluate estimates:
  - $p_{iR} = \frac{D_i}{D}$
  - $q_{iR} = \frac{n_i - D_i}{N - D}$
- This process can then be repeated recursively

Improving the Initial Ranking

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

- To avoid problems with $D = 1$ and $D_i = 0$:

$$p_{iR} = \frac{D_i + 0.5}{D + 1}; \quad q_{iR} = \frac{n_i - D_i + 0.5}{N - D + 1}$$

- Also,

$$p_{iR} = \frac{D_i + \frac{n_i}{N}}{D + 1}; \quad q_{iR} = \frac{n_i - D_i + \frac{n_i}{N}}{N - D + 1}$$

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
- There is some controversy as to whether the probabilistic model outperforms the vector model
- Croft suggested that the probabilistic model provides a better retrieval performance
- However, Salton *et al* showed that the vector model outperforms it with general collections
- This also seems to be the dominant thought among researchers and practitioners of IR.