# Chapter 3

# Modeling

**Part I: Classic Models**

# IR Models

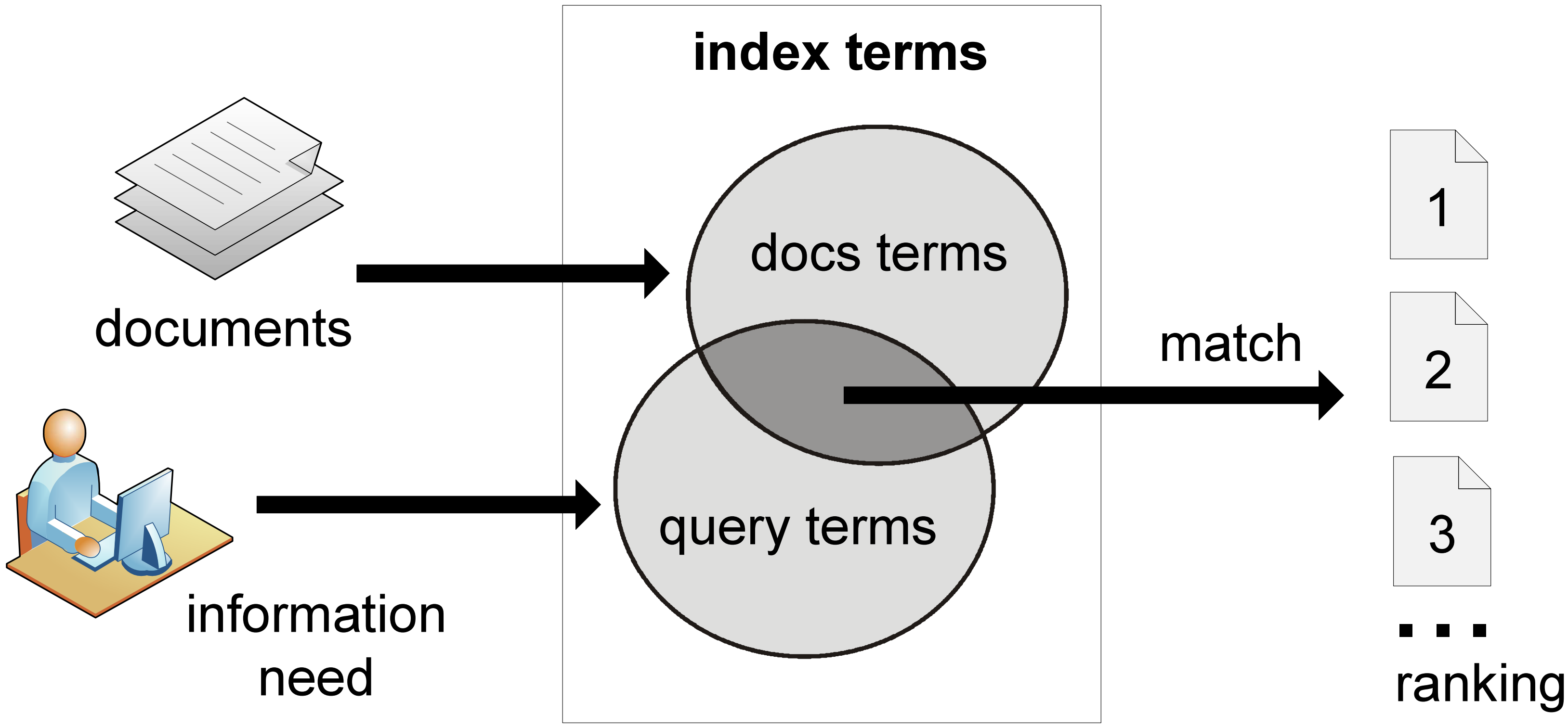
- **Modeling** in IR is a complex process aimed at producing a ranking function
  - **Ranking function**: a function that assigns scores to documents with regard to a given query
- This process consists of two main tasks:
  - The conception of a logical framework for representing documents and queries
  - The definition of a ranking function that allows quantifying the similarities among documents and queries

# Modeling and Ranking

- IR systems usually adopt **index terms** to index and retrieve documents
- Index term:
  - In a restricted sense: it is 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
- Retrieval based on index terms can be implemented efficiently
- Also, index terms are simple to refer to in a query
- Simplicity is important because it reduces the effort of query formulation

# Introduction

- Information retrieval process



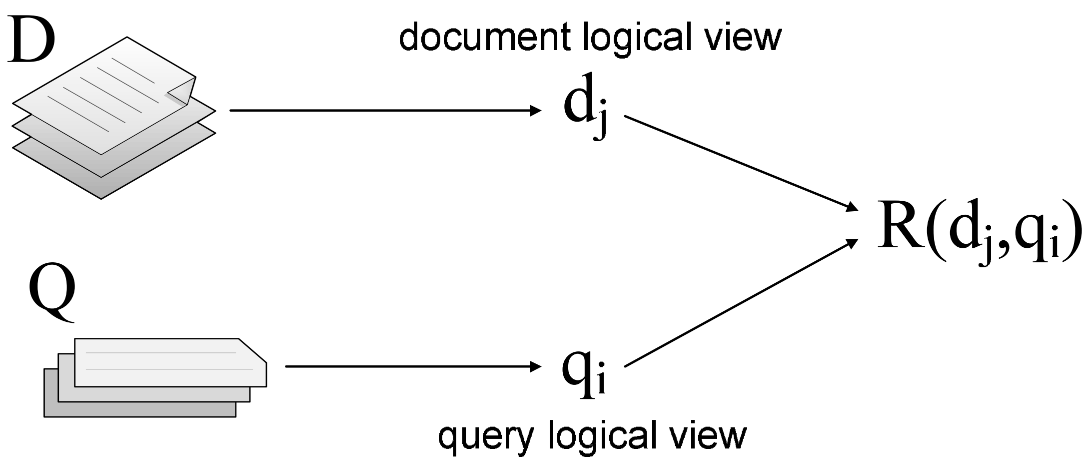
# Introduction

- A **ranking** is an ordering of the documents that (hopefully) reflects their **relevance** to a user query
- Thus, any IR system has to deal with the problem of predicting which documents the users will find relevant
- This problem naturally embodies a degree of uncertainty, or vagueness

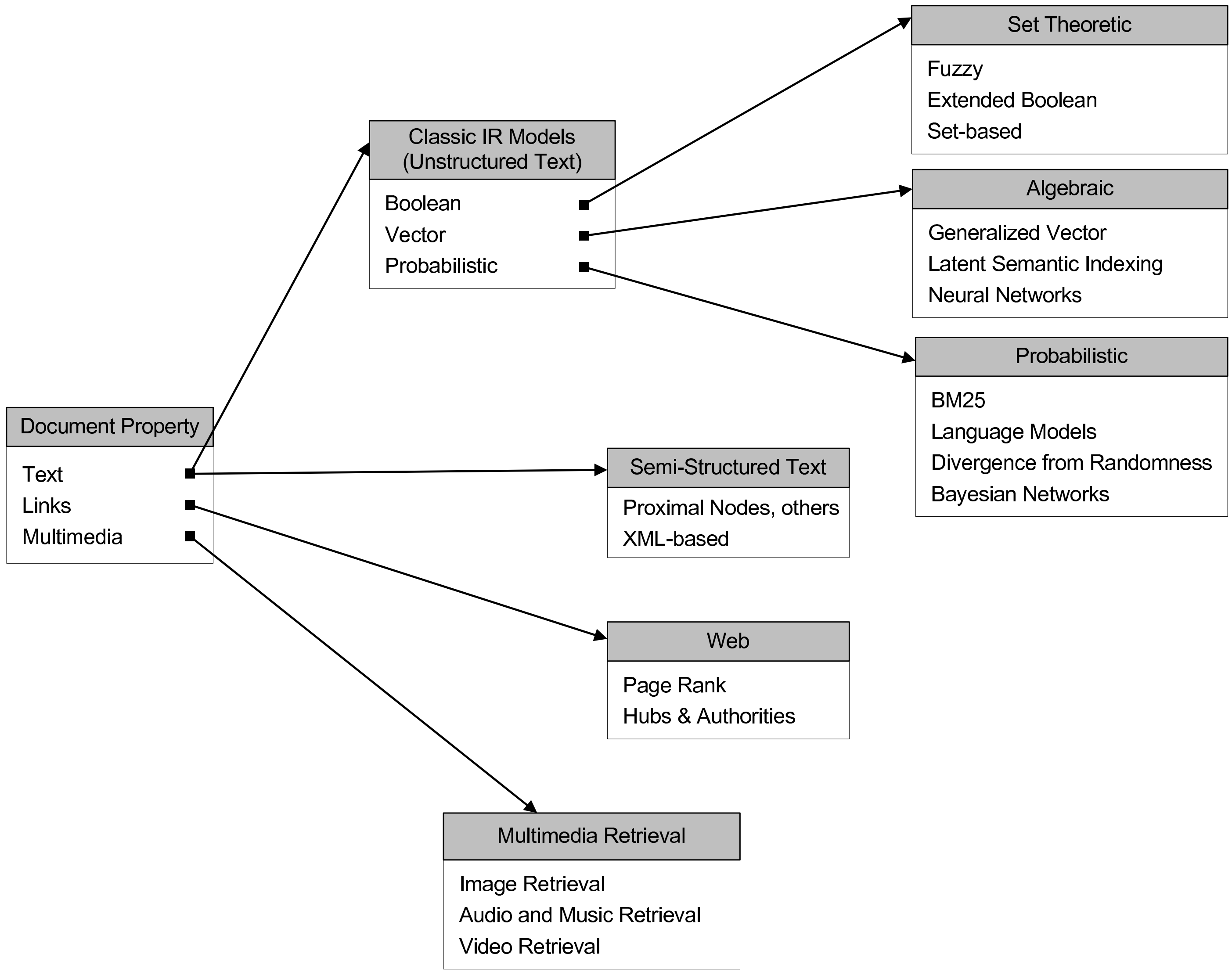
# IR Models

- 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 documents and queries
  4. $R(q_i, d_j)$ is a ranking function

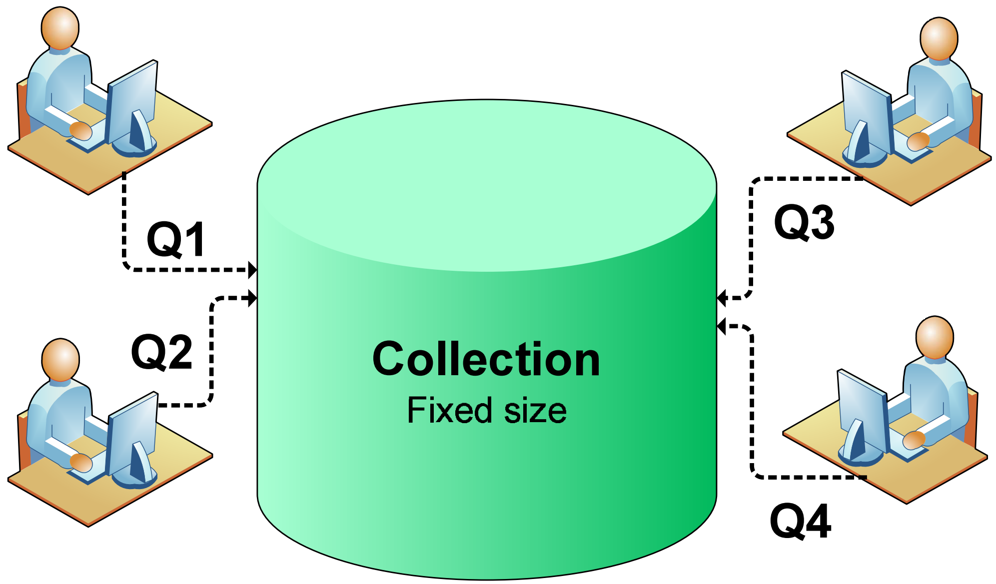


# A Taxonomy of IR Models



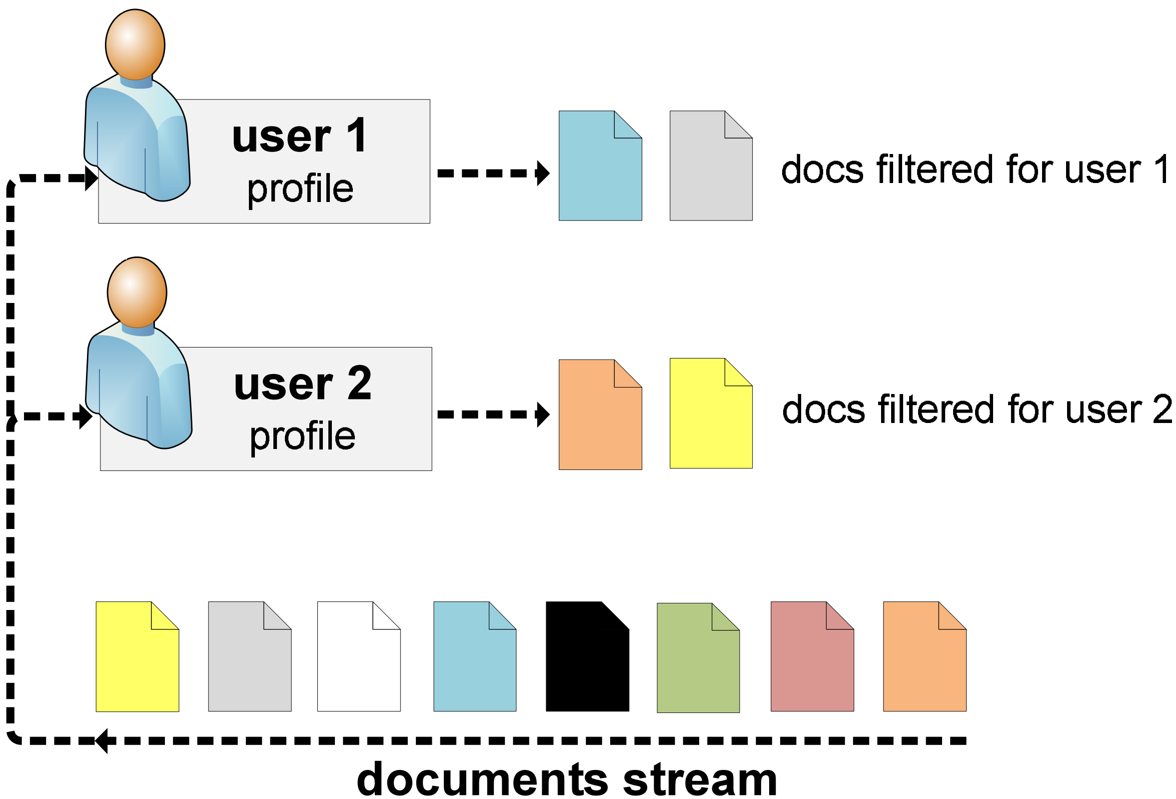
# Retrieval: Ad Hoc x Filtering

- Ad Hoc Retrieval:



# Retrieval: Ad Hoc x Filtering

- Filtering



# 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**

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

$$\begin{array}{|ccccc|}\hline 1 & 0 & 0 & \ldots & 0 \\ \hline\end{array}$$

pattern that represents documents (and queries) with the term $k_1$ and no other

$$\vdots$$

$$\begin{array}{|ccccc|}\hline 1 & 1 & 1 & \ldots & 1 \\ \hline\end{array}$$

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 $c(q)$)

# The Term-Document Matrix

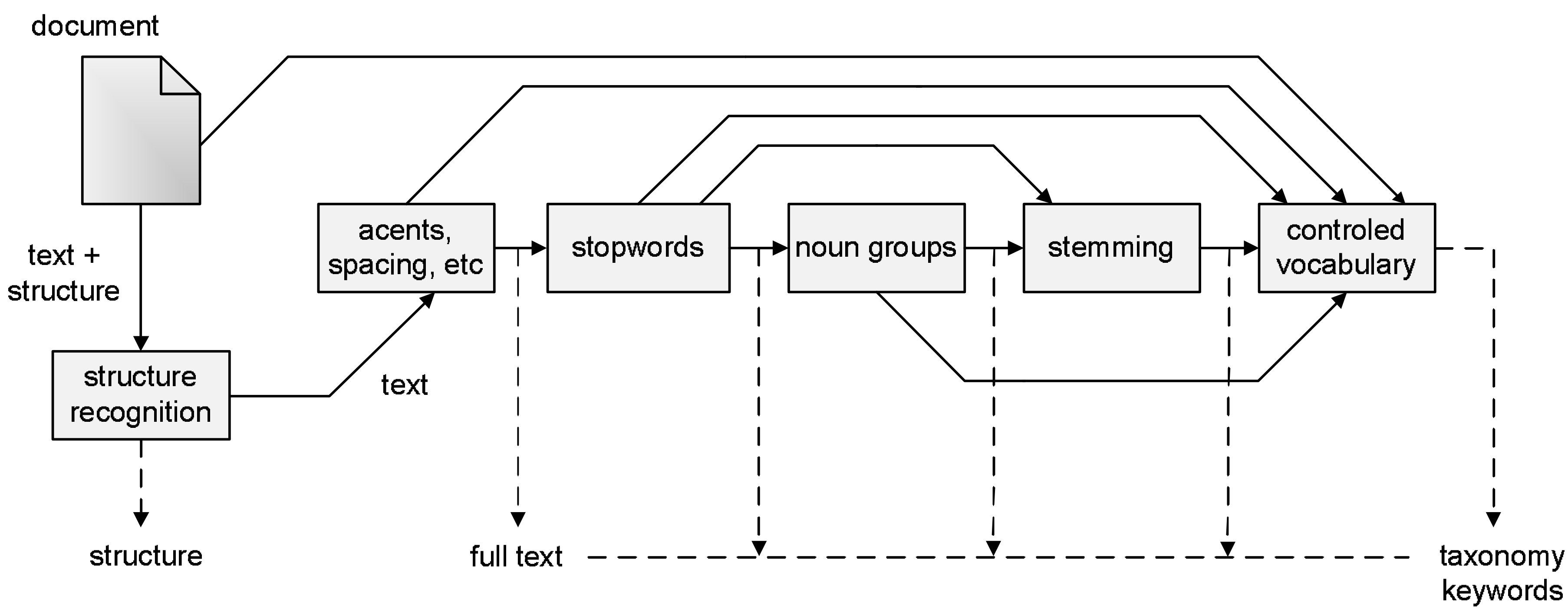
- The occurrence of a term $k_i$ in a document $d_j$ establishes a relation between $k_i$ and $d_j$
- A **term-document relation** between $k_i$ and $d_j$ can be quantified by the frequency of the term in the document
- In matrix form, this can written as

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

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

# 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

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

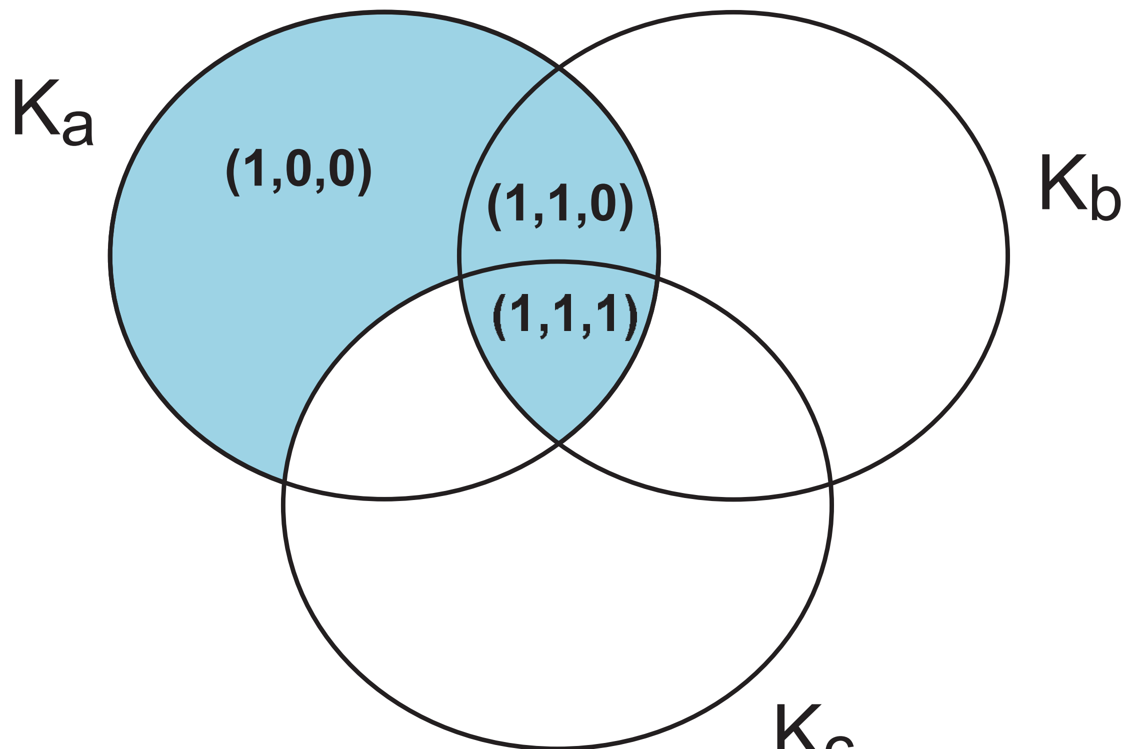
- Term-document frequencies in the term-document matrix are all binary
  - $w_{ij} \in \{0, 1\}$: weight associated with pair $(k_i, d_j)$
  - $w_{iq} \in \{0, 1\}$: weight associated with pair $(k_i, q)$

# 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

# Term Weighting

# Term Weighting

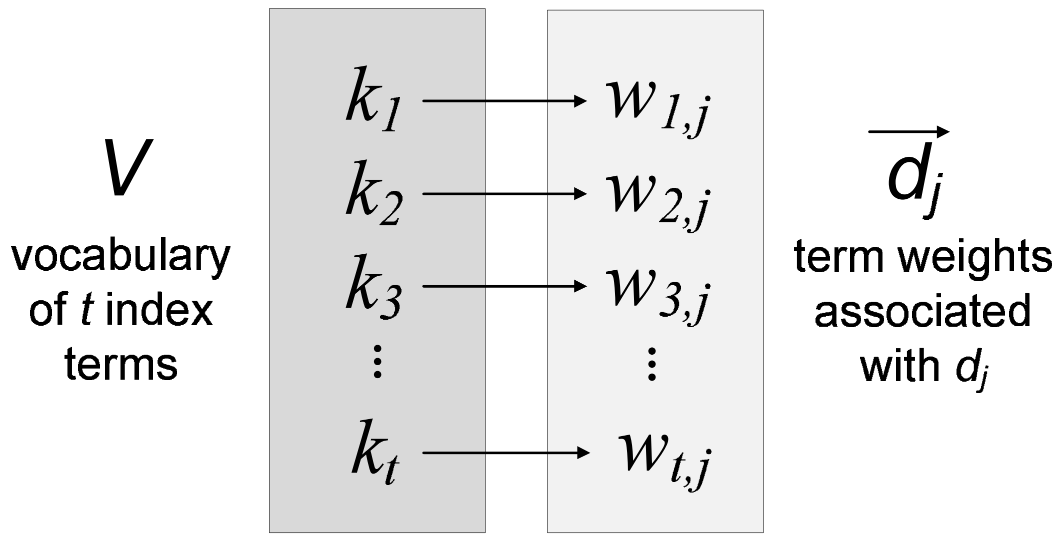
- The terms of a document are not equally useful for describing the document contents
- In fact, there are index terms which are simply vaguer than others
- There are properties of an index term which are useful for evaluating the importance of the term in a document
  - For instance, a word which appears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

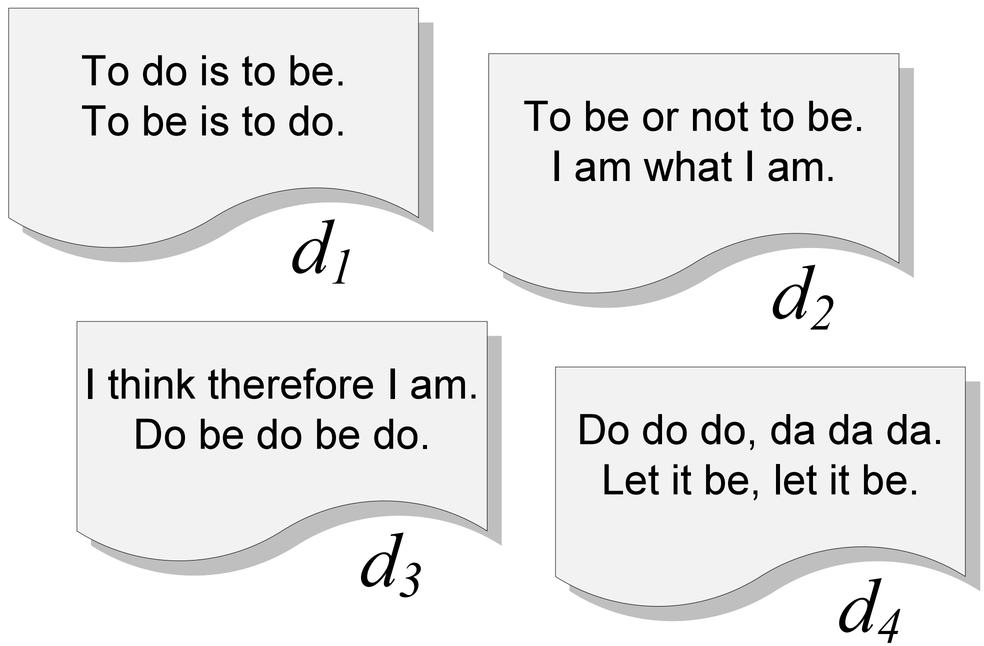
# Term Weighting

- 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

# Term-term correlation matrix

- Term-term correlation matrix for a sample collection

$$
\underbrace{
\begin{array}{cc}
 & \begin{array}{cc} d_1 & d_2 \end{array} \\
\begin{array}{c} k_1 \\ k_2 \\ k_3 \end{array} &
\left[\begin{array}{cc}
w_{1,1} & w_{1,2} \\
w_{2,1} & w_{2,2} \\
w_{3,1} & w_{3,2}
\end{array}\right] \\
 & \mathbf{M}
\end{array}
\quad \times \quad
\begin{array}{cc}
 & \begin{array}{ccc} k_1 & k_2 & k_3 \end{array} \\
\begin{array}{c} d_1 \\ d_2 \end{array} &
\left[\begin{array}{ccc}
w_{1,1} & w_{2,1} & w_{3,1} \\
w_{1,2} & w_{2,2} & w_{3,2}
\end{array}\right] \\
 & \mathbf{M}^T
\end{array}
}
$$

$$\Downarrow$$

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

# 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 $r$th largest document frequency and $\alpha$ 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

- 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,

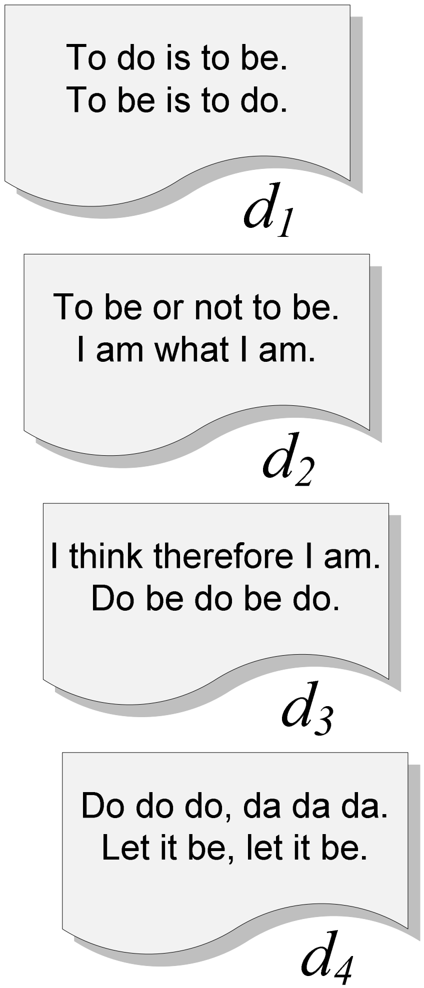
$$idf_i = \log \frac{N}{n_i}$$

  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

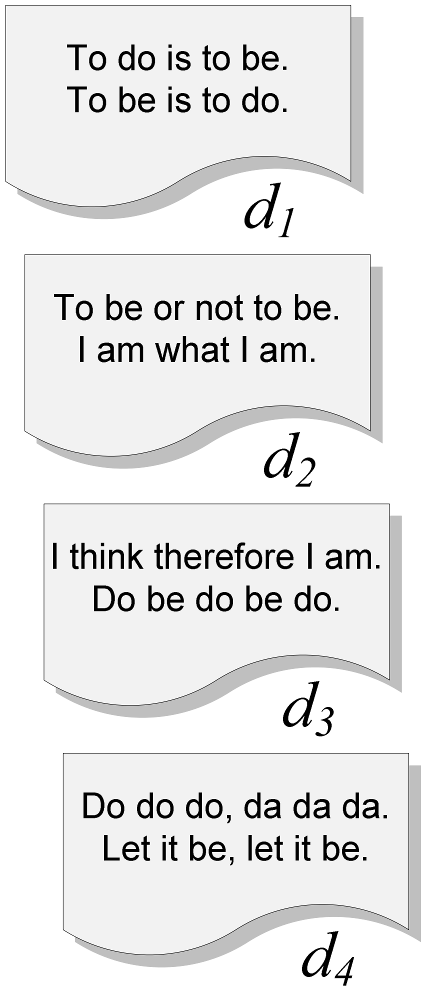
- The best known term weighting schemes use weights that combine idf factors with term frequencies
- Let $w_{i,j}$ be the term weight associated with the term $k_i$ and the document $d_j$
- Then, we define

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

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 | $(1 + \log f_{i,j}) * \log \frac{N}{n_i}$ | $(1 + \log f_{i,q}) * \log \frac{N}{n_i}$ |

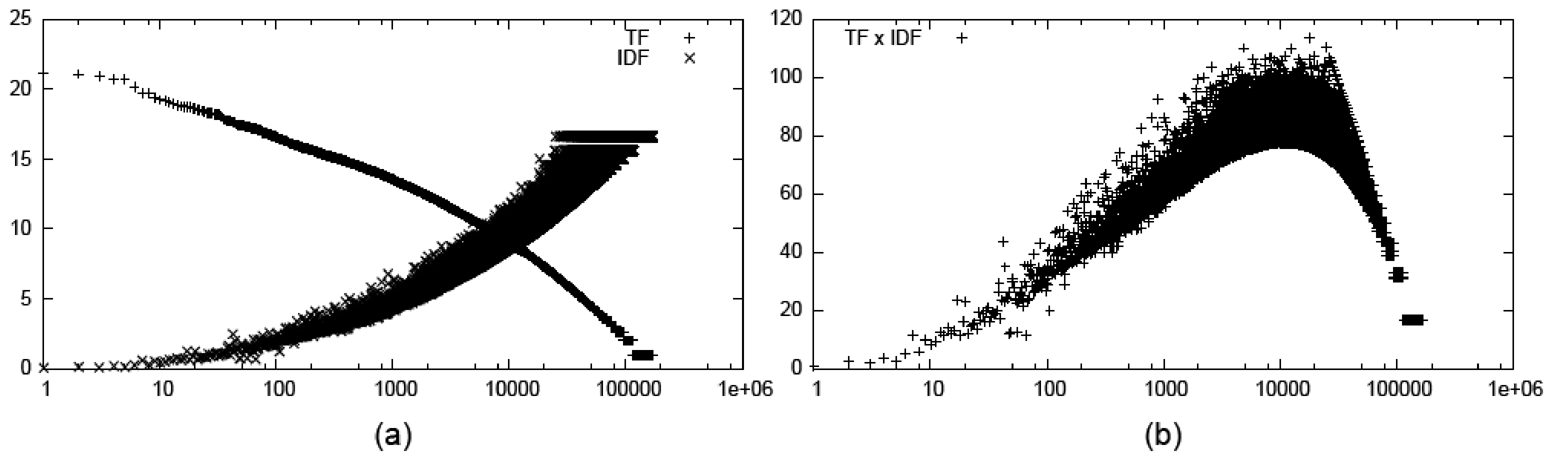
# TF-IDF Properties

- 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:

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

# TF-IDF Properties

- Plotting tf and idf in logarithmic scale yields




- 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

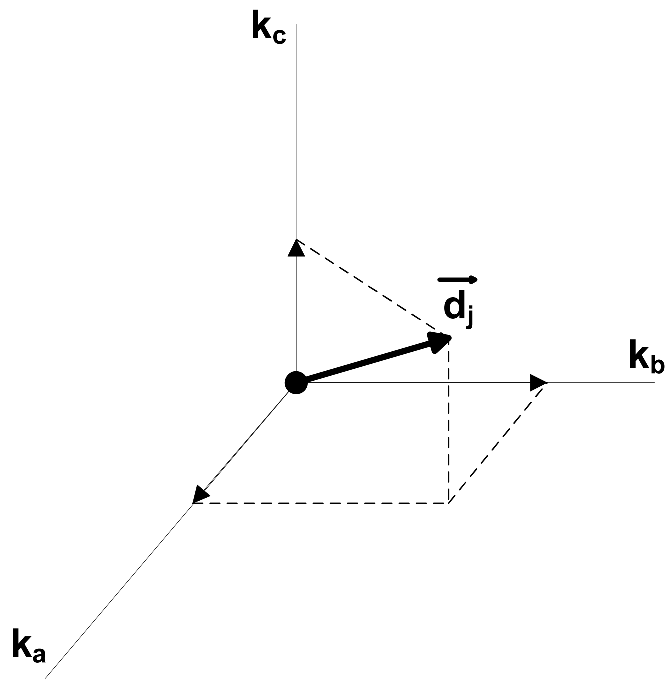
- Document sizes might vary widely
- This is a problem because longer documents are more likely to be retrieved by a given query
- To compensate for this undesired effect, we can divide the rank of each document by its length
- This procedure consistently leads to better ranking, and it is called **document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

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 |

# 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

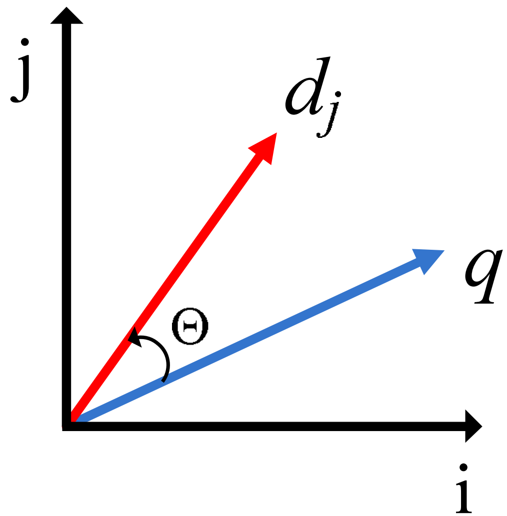
# The Vector Model

- 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

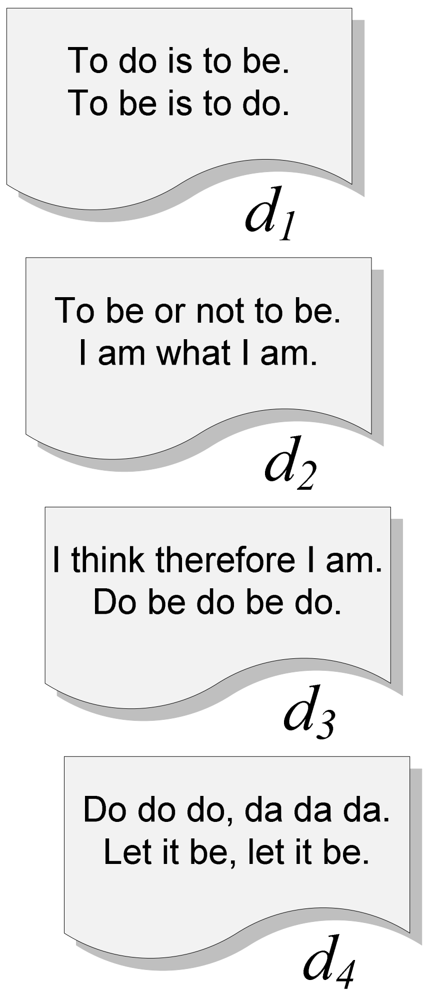
- Weights in the Vector model are basically tf-idf weights

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

- 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)




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

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

# 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 $R$
  - $P(R, q)$ : 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 all binary and assuming 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

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

# 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

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

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

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

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

  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
  - $p_{iR} = \frac{r_i}{R}$
  - $q_{iR} = \frac{n_i - r_i}{N - R}$
- 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 $k_i \in q \wedge k_i \in d_j$

# 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

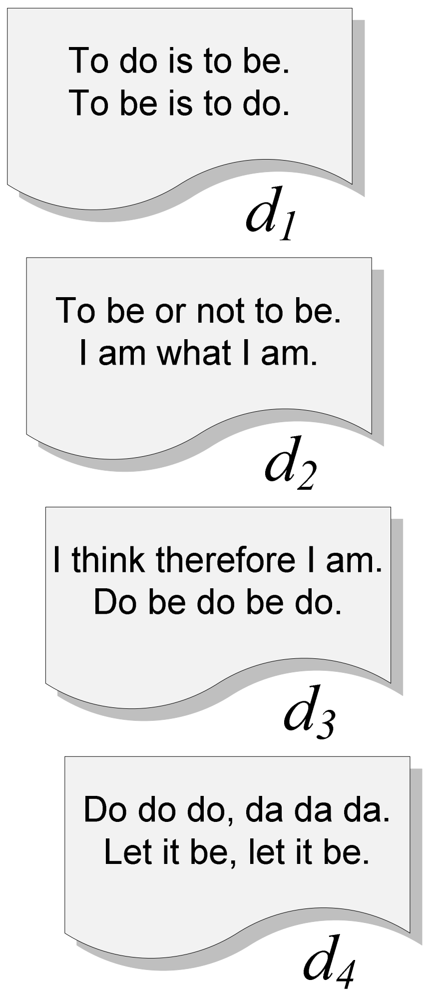
- The previous equation cannot be computed without estimates of $r_i$ and $R$
- One possibility is to assume $R = r_i = 0$, as a way to boostrap the ranking equation, which leads to

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

- 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”




| 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

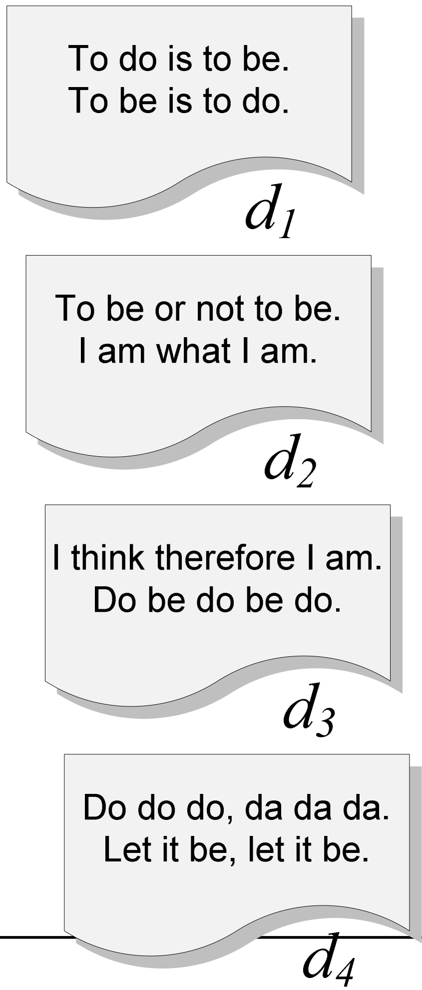
- The ranking computation led to negative weights because of the term “do”
- Actually, the probabilistic ranking equation produces negative terms whenever $n_i > N/2$
- One possible artifact to contain the effect of negative weights is to change the previous equation to:

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

- 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 $D_i = 0$:

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

- Also,

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

# 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.

# Modeling

**Part II: Alternative Set and Vector Models**

Set-Based Model
Extended Boolean Model
Fuzzy Set Model
The Generalized Vector Model
Latent Semantic Indexing
Neural Network for IR

# Alternative Set Theoretic Models

- Set-Based Model
- Extended Boolean Model
- Fuzzy Set Model

# Set-Based Model

# Set-Based Model

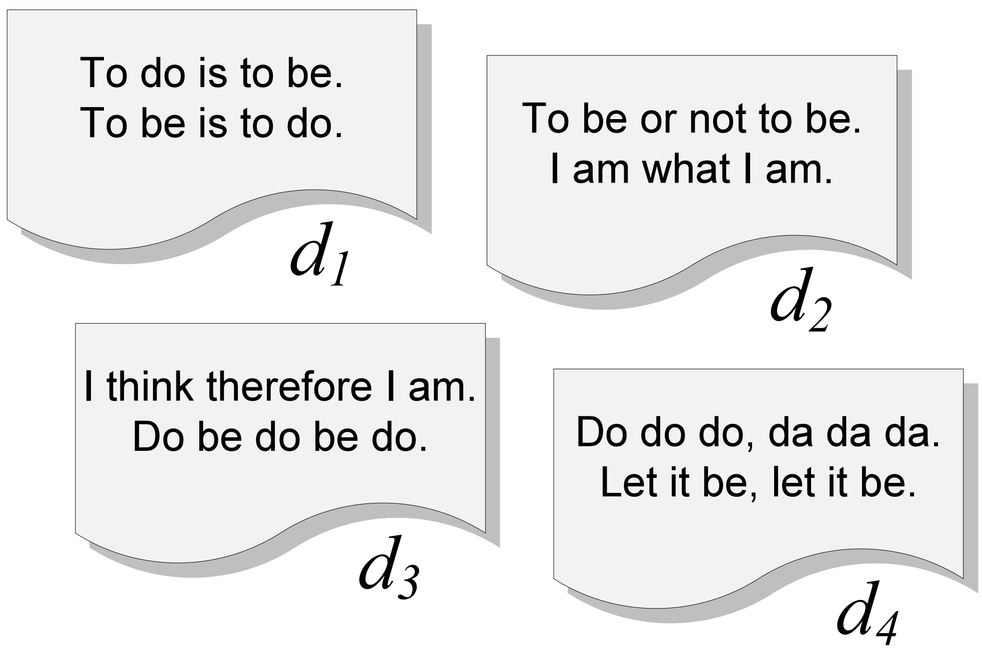
- This is a more recent approach (2005) that combines set theory with a vectorial ranking
- The fundamental idea is to use mutual dependencies among index terms to improve results
- Term dependencies are captured through **termsets**, which are sets of correlated terms
- The approach, which leads to improved results with various collections, constitutes the first IR model that effectively took advantage of term dependence with general collections

# Termsets

- **Termset** is a concept used in place of the index terms
- A termset $S_i = \{k_a, k_b, ..., k_n\}$ is a subset of the terms in the collection
- If all index terms in $S_i$ occur in a document $d_j$ then we say that the termset $S_i$ occurs in $d_j$
- There are $2^t$ termsets that might occur in the documents of a collection, where $t$ is the vocabulary size
  - However, most combinations of terms have no semantic meaning
  - Thus, the actual number of termsets in a collection is far smaller than $2^t$

# Termsets

- Let $t$ be the number of terms of the collection
- Then, the set $V_S = \{S_1, S_2, ..., S_{2^t}\}$ is the **vocabulary-set** of the collection
- To illustrate, consider the document collection below

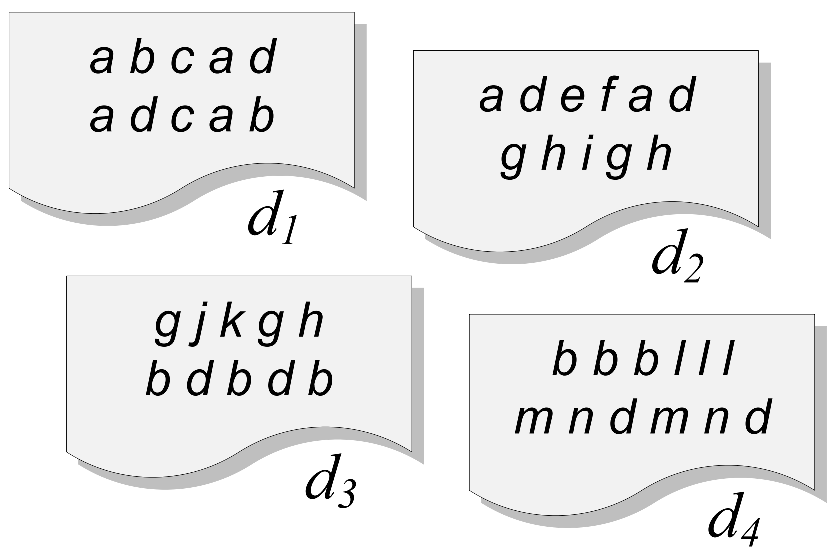


# Termsets

- To simplify notation, let us define

| | | | | |
|---|---|---|---|---|
| $k_a$ = to | $k_d$ = be | $k_g$ = I | $k_j$ = think | $k_m$ = let |
| $k_b$ = do | $k_e$ = or | $k_h$ = am | $k_k$ = therefore | $k_n$ = it |
| $k_c$ = is | $k_f$ = not | $k_i$ = what | $k_l$ = da | |

- Further, let the letters $a...n$ refer to the index terms $k_a...k_n$ , respectively



# Termsets

- Consider the query $q$ as “to do be it”, i.e. $q = \{a, b, d, n\}$
- For this query, the vocabulary-set is as below

| Termset | Set of Terms | Documents |
|---|---|---|
| $S_a$ | $\{a\}$ | $\{d_1, d_2\}$ |
| $S_b$ | $\{b\}$ | $\{d_1, d_3, d_4\}$ |
| $S_d$ | $\{d\}$ | $\{d_1, d_2, d_3, d_4\}$ |
| $S_n$ | $\{n\}$ | $\{d_4\}$ |
| $S_{ab}$ | $\{a, b\}$ | $\{d_1\}$ |
| $S_{ad}$ | $\{a, d\}$ | $\{d_1, d_2\}$ |
| $S_{bd}$ | $\{b, d\}$ | $\{d_1, d_3, d_4\}$ |
| $S_{bn}$ | $\{b, n\}$ | $\{d_4\}$ |
| $S_{abd}$ | $\{a, b, d\}$ | $\{d_1\}$ |
| $S_{bdn}$ | $\{b, d, n\}$ | $\{d_4\}$ |

Notice that there are 11 termsets that occur in our collection, out of the maximum of 15 termsets that can be formed with the terms in $q$

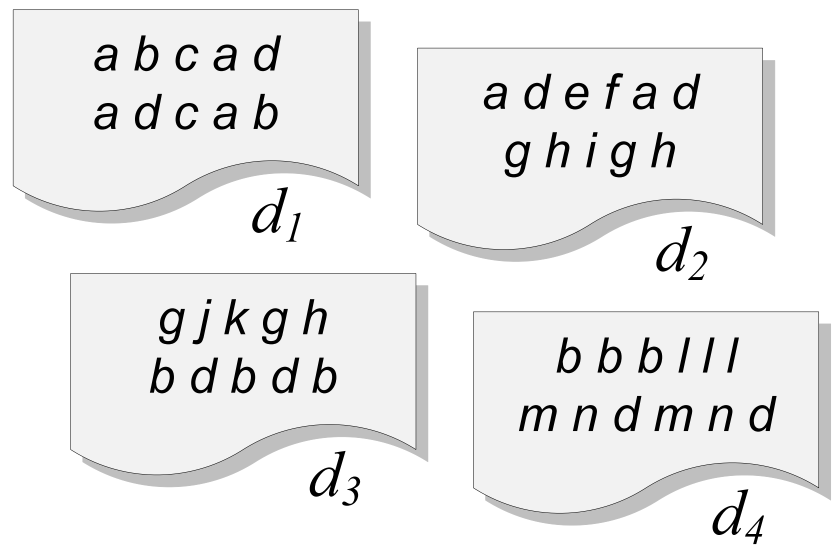
# Termsets

- At query processing time, only the termsets generated by the query need to be considered
- A termset composed of $n$ terms is called an $n$-termset
- Let $\mathcal{N}_i$ be the number of documents in which $S_i$ occurs
- An $n$-termset $S_i$ is said to be **frequent** if $\mathcal{N}_i$ is greater than or equal to a given threshold
  - This implies that an $n$-termset is frequent if and only if all of its $(n-1)$-termsets are also frequent
  - **Frequent termsets** can be used to reduce the number of termsets to consider with long queries

# Termsets

- Let the threshold on the frequency of termsets be 2
- To compute all frequent termsets for the query $q = \{a, b, d, n\}$ we proceed as follows

1. Compute the frequent 1-termsets and their inverted lists:
   - $S_a = \{d_1, d_2\}$
   - $S_b = \{d_1, d_3, d_4\}$
   - $S_d = \{d_1, d_2, d_3, d_4\}$
2. Combine the inverted lists to compute frequent 2-termsets:
   - $S_{ad} = \{d_1, d_2\}$
   - $S_{bd} = \{d_1, d_3, d_4\}$
3. Since there are no frequent 3-termsets, stop



# Termsets

- Notice that there are only 5 *frequent* termsets in our collection
- Inverted lists for frequent $n$-termsets can be computed by starting with the inverted lists of frequent 1-termsets
  - Thus, the only indice that is required are the standard inverted lists used by any IR system
- This is reasonably fast for short queries up to 4-5 terms

# Ranking Computation

- The ranking computation is based on the vector model, but adopts termsets instead of index terms
- Given a query $q$, let
  - $\{S_1, S_2, \ldots\}$ be the set of all termsets originated from $q$
  - $\mathcal{N}_i$ be the number of documents in which termset $S_i$ occurs
  - $N$ be the total number of documents in the collection
  - $\mathcal{F}_{i,j}$ be the frequency of termset $S_i$ in document $d_j$
- For each pair $[S_i, d_j]$ we compute a weight $\mathcal{W}_{i,j}$ given by

$$\mathcal{W}_{i,j} = \begin{cases} (1 + \log \mathcal{F}_{i,j}) \log(1 + \frac{N}{\mathcal{N}_i}) & \texttt{if} \;\; \mathcal{F}_{i,j} > 0 \\ 0 & \mathcal{F}_{i,j} = 0 \end{cases}$$

- We also compute a $\mathcal{W}_{i,q}$ value for each pair $[S_i, q]$

# Ranking Computation

- Consider
  - query $q = \{a, b, d, n\}$
  - document $d_1$ = ''a b c a d a d c a b''

| Termset | Weight | |
|---|---|---|
| $S_a$ | $\mathcal{W}_{a,1}$ | $(1 + \log 4) * \log(1 + 4/2) = 4.75$ |
| $S_b$ | $\mathcal{W}_{b,1}$ | $(1 + \log 2) * \log(1 + 4/3) = 2.44$ |
| $S_d$ | $\mathcal{W}_{d,1}$ | $(1 + \log 2) * \log(1 + 4/4) = 2.00$ |
| $S_n$ | $\mathcal{W}_{n,1}$ | $0 * \log(1 + 4/1) = 0.00$ |
| $S_{ab}$ | $\mathcal{W}_{ab,1}$ | $(1 + \log 2) * \log(1 + 4/1) = 4.64$ |
| $S_{ad}$ | $\mathcal{W}_{ad,1}$ | $(1 + \log 2) * \log(1 + 4/2) = 3.17$ |
| $S_{bd}$ | $\mathcal{W}_{bd,1}$ | $(1 + \log 2) * \log(1 + 4/3) = 2.44$ |
| $S_{bn}$ | $\mathcal{W}_{bn,1}$ | $0 * \log(1 + 4/1) = 0.00$ |
| $S_{dn}$ | $\mathcal{W}_{dn,1}$ | $0 * \log(1 + 4/1) = 0.00$ |
| $S_{abd}$ | $\mathcal{W}_{abd,1}$ | $(1 + \log 2) * \log(1 + 4/1) = 4.64$ |
| $S_{bdn}$ | $\mathcal{W}_{bdn,1}$ | $0 * \log(1 + 4/1) = 0.00$ |

# Ranking Computation

- A document $d_j$ and a query $q$ are represented as vectors in a $2^t$-dimensional space of termsets

$$\begin{aligned}\vec{d_j} &= (\mathcal{W}_{1,j}, \mathcal{W}_{2,j}, \ldots, \mathcal{W}_{2^t,j}) \\ \vec{q} &= (\mathcal{W}_{1,q}, \mathcal{W}_{2,q}, \ldots, \mathcal{W}_{2^t,q})\end{aligned}$$

- The rank of $d_j$ to the query $q$ is computed as follows

$$sim(d_j, q) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_{S_i} \mathcal{W}_{i,j} \times \mathcal{W}_{i,q}}{|\vec{d_j}| \times |\vec{q}|}$$

- For termsets that are not in the query $q$, $\mathcal{W}_{i,q} = 0$

# Ranking Computation

- The document norm $|\vec{d}_j|$ is hard to compute in the space of termsets
- Thus, its computation is restricted to 1-termsets
- Let again $q = \{a, b, d, n\}$ and $d_1$
- The document norm in terms of 1-termsets is given by

$$
\begin{aligned}
|\vec{d}_1| &= \sqrt{\mathcal{W}_{a,1}^2 + \mathcal{W}_{b,1}^2 + \mathcal{W}_{c,1}^2 + \mathcal{W}_{d,1}^2} \\
&= \sqrt{4.75^2 + 2.44^2 + 4.64^2 + 2.00^2} \\
&= 7.35
\end{aligned}
$$

# Ranking Computation

- To compute the rank of $d_1$, we need to consider the seven termsets $S_a$, $S_b$, $S_d$, $S_{ab}$, $S_{ad}$, $S_{bd}$, and $S_{abd}$
- The rank of $d_1$ is then given by

$$\begin{aligned}
sim(d_1, q) &= (\mathcal{W}_{a,1} * \mathcal{W}_{a,q} + \mathcal{W}_{b,1} * \mathcal{W}_{b,q} + \mathcal{W}_{d,1} * \mathcal{W}_{d,q} + \\
&\quad \mathcal{W}_{ab,1} * \mathcal{W}_{ab,q} + \mathcal{W}_{ad,1} * \mathcal{W}_{ad,q} + \mathcal{W}_{bd,1} * \mathcal{W}_{bd,q} + \\
&\quad \mathcal{W}_{abd,1} * \mathcal{W}_{abd,q}) / |\vec{d_1}| \\
&= (4.75 * 1.58 + 2.44 * 1.22 + 2.00 * 1.00 + \\
&\quad 4.64 * 2.32 + 3.17 * 1.58 + 2.44 * 1.22 + \\
&\quad 4.64 * 2.32)/7.35 \\
&= 5.71
\end{aligned}$$

# Closed Termsets

- The concept of frequent termsets allows simplifying the ranking computation
- Yet, there are many frequent termsets in a large collection
  - The number of termsets to consider might be prohibitively high with large queries
- To resolve this problem, we can further restrict the ranking computation to a smaller number of termsets
- This can be accomplished by observing some properties of termsets such as the notion of **closure**

# Closed Termsets

- The **closure of a termset** $S_i$ is the set of all frequent termsets that co-occur with $S_i$ in the same set of docs
- Given the closure of $S_i$, the largest termset in it is called a **closed termset** and is referred to as $\Phi_i$
- We formalize, as follows
  - Let $D_i \subseteq C$ be the subset of all documents in which termset $S_i$ occurs and is frequent
  - Let $S(D_i)$ be a set composed of the frequent termsets that occur in all documents in $D_i$ and only in those

# Closed Termsets

- Then, the closed termset $S_{\Phi_i}$ satisfies the following property

$$\nexists S_j \in S(D_i) \mid S_{\Phi_i} \subset S_j$$

- Frequent and closed termsets for our example collection, considering a minimum threshold equal to 2

| frequency($S_i$) | frequent termset | closed termset |
|---|---|---|
| 4 | d | d |
| 3 | b, bd | bd |
| 2 | a, ad | ad |
| 2 | g, h, gh, ghd | ghd |

# Closed Termsets

- Closed termsets encapsulate smaller termsets occurring in the same set of documents
- The ranking $sim(d_1, q)$ of document $d_1$ with regard to query $q$ is computed as follows:
  - $d_1 =$''a b c a d a d c a b ''
  - $q = \{a, b, d, n\}$
  - minimum frequency threshold = 2

$$
\begin{aligned}
sim(d_1, q) &= (\mathcal{W}_{d,1} * \mathcal{W}_{d,q} + \mathcal{W}_{ab,1} * \mathcal{W}_{ab,q} + \mathcal{W}_{ad,1} * \mathcal{W}_{ad,q} + \\
&\quad \mathcal{W}_{bd,1} * \mathcal{W}_{bd,q} + \mathcal{W}_{abd,1} * \mathcal{W}_{abd,q})/|\vec{d_1}| \\
&= (2.00 * 1.00 + 4.64 * 2.32 + 3.17 * 1.58 + \\
&\quad 2.44 * 1.22 + 4.64 * 2.32)/7.35 \\
&= 4.28
\end{aligned}
$$

# Closed Termsets

- Thus, if we restrict the ranking computation to closed termsets, we can expect a reduction in query time
- Smaller the number of closed termsets, sharper is the reduction in query processing time

# Extended Boolean Model

# Extended Boolean Model

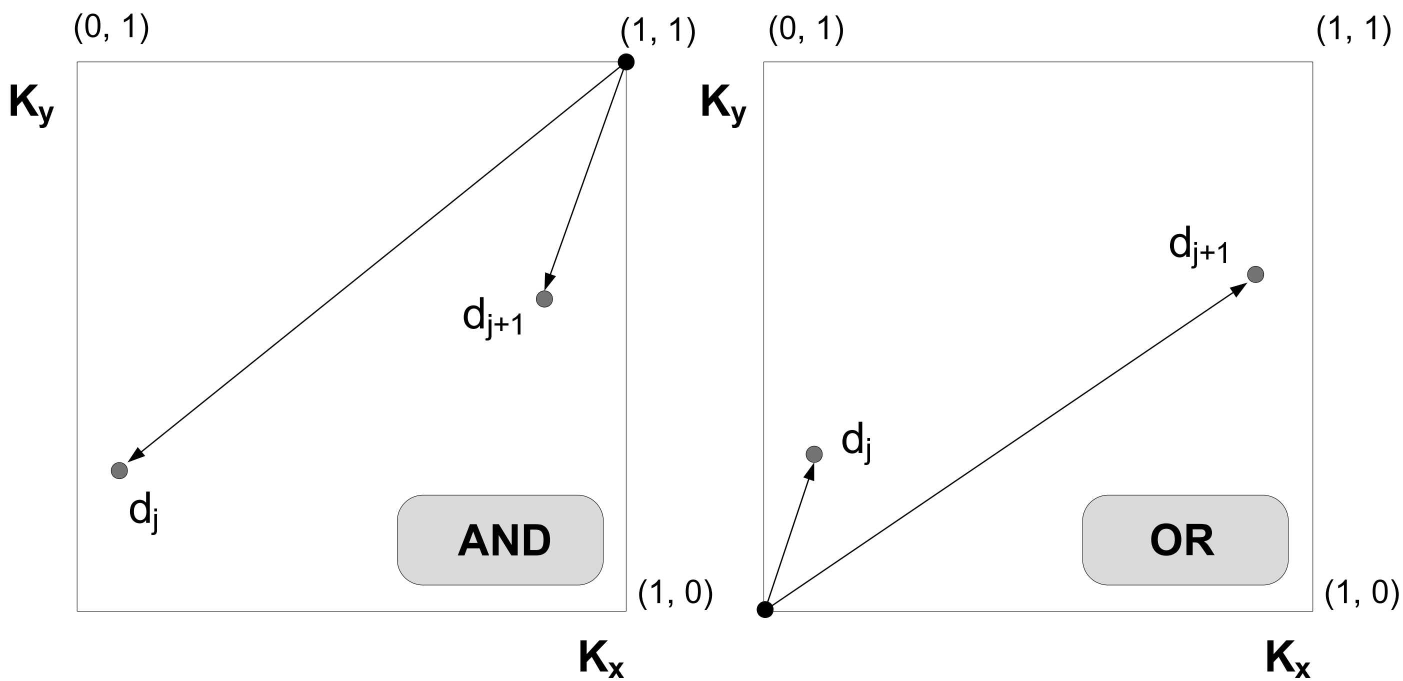
- In the Boolean model, no **ranking** of the answer set is generated
- One alternative is to extend the Boolean model with the notions of **partial matching** and **term weighting**
- This strategy allows one to combine characteristics of the Vector model with properties of Boolean algebra

# The Idea

- Consider a conjunctive Boolean query given by $q = k_x \wedge k_y$
- For the boolean model, a doc that contains a single term of $q$ is as irrelevant as a doc that contains none
- However, this **binary decision** criteria frequently is not in accordance with common sense
- An analogous reasoning applies when one considers purely disjunctive queries

# The Idea

- When only two terms $x$ and $y$ are considered, we can plot queries and docs in a two-dimensional space




- A document $d_j$ is positioned in this space through the adoption of weights $w_{x,j}$ and $w_{y,j}$

# The Idea

- These weights can be computed as normalized tf-idf factors as follows

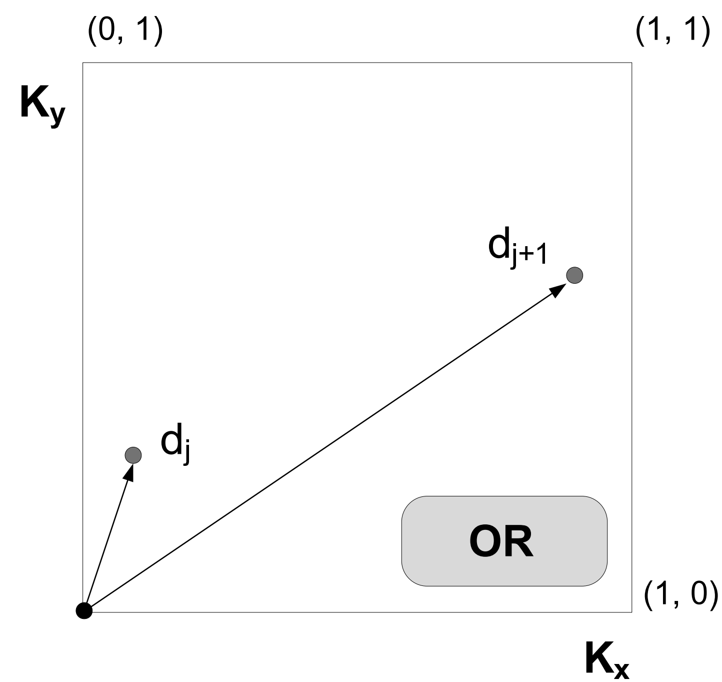
$$w_{x,j} = \frac{f_{x,j}}{max_x \ \ f_{x,j}} \times \frac{idf_x}{max_i \ \ idf_i}$$

- where
  - $f_{x,j}$ is the frequency of term $k_x$ in document $d_j$
  - $idf_i$ is the inverse document frequency of term $k_i$, as before
- To simplify notation, let
  - $w_{x,j} = x$ and $w_{y,j} = y$
  - $\vec{d_j} = (w_{x,j}, w_{y,j})$ as the point $d_j = (x, y)$

# The Idea

- For a disjunctive query $q_{or} = k_x \vee k_y$, the point $(0,0)$ is the least interesting one
- This suggests taking the distance from $(0,0)$ as a measure of similarity

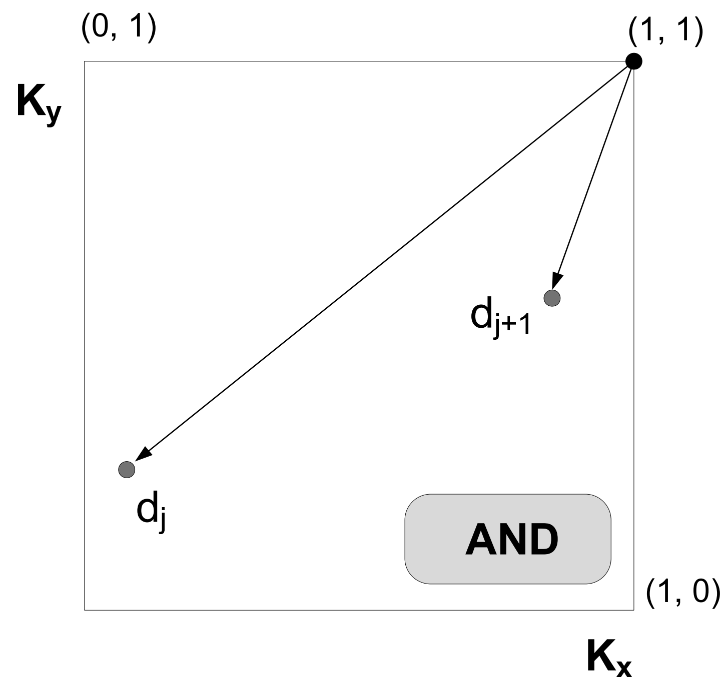
$$sim(q_{or}, d) \;=\; \sqrt{\frac{x^2+y^2}{2}}$$



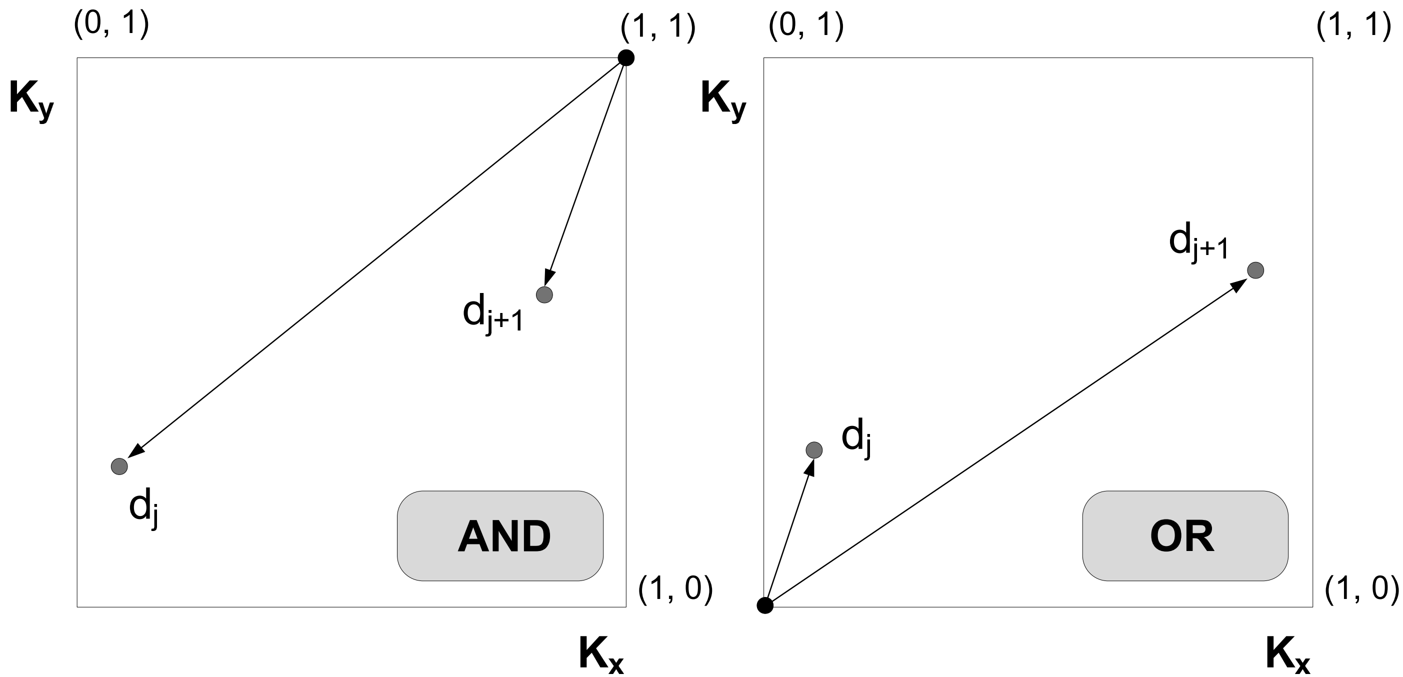
# The Idea

- For a conjunctive query $q_{and} = k_x \wedge k_y$, the point $(1, 1)$ is the most interesting one
- This suggests taking the complement of the distance from the point $(1, 1)$ as a measure of similarity

$$sim(q_{and}, d) \quad = \quad 1 - \sqrt{\frac{(1-x)^2 + (1-y)^2}{2}}$$



# The Idea




$$sim(q_{or}, d) = \sqrt{\frac{x^2 + y^2}{2}}$$

$$sim(q_{and}, d) = 1 - \sqrt{\frac{(1-x)^2 + (1-y)^2}{2}}$$

# Generalizing the Idea

- We can extend the previous model to consider Euclidean distances in a t-dimensional space
- This can be done using *p-norms* which extend the notion of distance to include p-distances, where $1 \leq p \leq \infty$
- A generalized conjunctive query is given by
  - $q_{and} = k_1 \ \wedge^p \ k_2 \ \wedge^p \ \ldots \wedge^p k_m$
- A generalized disjunctive query is given by
  - $q_{or} = k_1 \ \vee^p \ k_2 \ \vee^p \ \ldots \vee^p k_m$

# Generalizing the Idea

- The query-document similarities are now given by

$$sim(q_{or}, d_j) = \left(\frac{x_1^p + x_2^p + \ldots + x_m^p}{m}\right)^{\frac{1}{p}}$$

$$sim(q_{and}, d_j) = 1 - \left(\frac{(1-x_1)^p + (1-x_2)^p + \ldots + (1-x_m)^p}{m}\right)^{\frac{1}{p}}$$

  where each $x_i$ stands for a weight $w_{i,d}$

- If $p = 1$ then (vector-like)
  - $sim(q_{or}, d_j) = sim(q_{and}, d_j) = \frac{x_1 + \ldots + x_m}{m}$
- If $p = \infty$ then (Fuzzy like)
  - $sim(q_{or}, d_j) = max(x_i)$
  - $sim(q_{and}, d_j) = min(x_i)$

# Properties

- By varying *p*, we can make the model behave as a vector, as a fuzzy, or as an intermediary model
- The processing of more general queries is done by grouping the operators in a predefined order
- For instance, consider the query $q = (k_1 \wedge^p k_2) \vee^p k_3$
  - $k_1$ and $k_2$ are to be used as in a vectorial retrieval while the presence of $k_3$ is required
- The similarity $sim(q, d_j)$ is computed as

$$sim(q,d) = \left( \frac{\left(1 - \left(\frac{(1-x_1)^p + (1-x_2)^p}{2}\right)^{\frac{1}{p}}\right)^p + x_3^p}{2} \right)^{\frac{1}{p}}$$

# Conclusions

- Model is quite powerful
- Properties are interesting and might be useful
- Computation is somewhat complex
- However, distributivity operation does not hold for ranking computation:
  - $q_1 = (k_1 \ \vee \ k_2) \ \wedge \ k_3$
  - $q_2 = (k_1 \ \wedge \ k_3) \ \vee \ (k_2 \ \wedge \ k_3)$
  - $sim(q_1, d_j) \neq sim(q_2, d_j)$

# Fuzzy Set Model

# Fuzzy Set Model

- Matching of a document to a query terms is approximate or vague
- This **vagueness** can be modeled using a fuzzy framework, as follows:
  - each query term defines a **fuzzy** set
  - each doc has a **degree of membership** in this set
- This interpretation provides the foundation for many IR models based on fuzzy theory
- In here, we discuss the model proposed by Ogawa, Morita, and Kobayashi

# Fuzzy Set Theory

- Fuzzy set theory deals with the representation of classes whose boundaries are not well defined
- Key idea is to introduce the notion of a **degree of membership** associated with the elements of the class
- This degree of membership varies from 0 to 1 and allows modelling the notion of **marginal** membership
- Thus, membership is now a **gradual** notion, contrary to the crispy notion enforced by classic Boolean logic

# Fuzzy Set Theory

- A fuzzy subset $A$ of a universe of discourse $U$ is characterized by a membership function

$$\mu_A : U \rightarrow [0,1]$$

- This function associates with each element $u$ of $U$ a number $\mu_A(u)$ in the interval $[0,1]$
- The three most commonly used operations on fuzzy sets are:
  - the complement of a fuzzy set
  - the union of two or more fuzzy sets
  - the intersection of two or more fuzzy sets

# Fuzzy Set Theory

- Let,
  - $U$ be the universe of discourse
  - $A$ and $B$ be two fuzzy subsets of $U$
  - $\overline{A}$ be the complement of $A$ relative to $U$
  - $u$ be an element of $U$
- Then,

$$\begin{aligned}
\mu_{\overline{A}}(u) &= 1 - \mu_A(u) \\
\mu_{A \cup B}(u) &= max(\mu_A(u), \mu_B(u)) \\
\mu_{A \cap B}(u) &= min(\mu_A(u), \mu_B(u))
\end{aligned}$$

# Fuzzy Information Retrieval

- Fuzzy sets are modeled based on a thesaurus, which defines term relationships
- A thesaurus can be constructed by defining a term-term correlation matrix $C$
- Each element of $C$ defines a normalized correlation factor $c_{i,\ell}$ between two terms $k_i$ and $k_\ell$

$$c_{i,l} = \frac{n_{i,l}}{n_i + n_l - n_{i,l}}$$

where

  - $n_i$: number of docs which contain $k_i$
  - $n_l$: number of docs which contain $k_l$
  - $n_{i,l}$: number of docs which contain both $k_i$ and $k_l$

# Fuzzy Information Retrieval

- We can use the term correlation matrix $C$ to associate a fuzzy set with each index term $k_i$
- In this fuzzy set, a document $d_j$ has a degree of membership $\mu_{i,j}$ given by

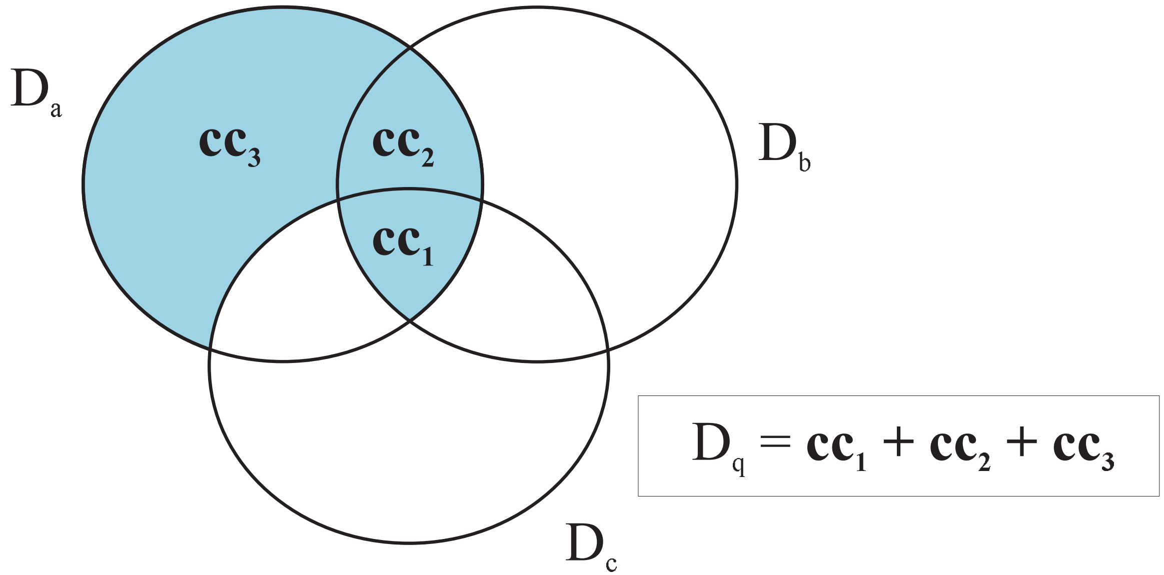
$$\mu_{i,j} = 1 - \prod_{k_l \,\in\, d_j} (1 - c_{i,l})$$

- The above expression computes an algebraic sum over all terms in $d_j$
- A document $d_j$ belongs to the fuzzy set associated with $k_i$, if its own terms are associated with $k_i$

# Fuzzy Information Retrieval

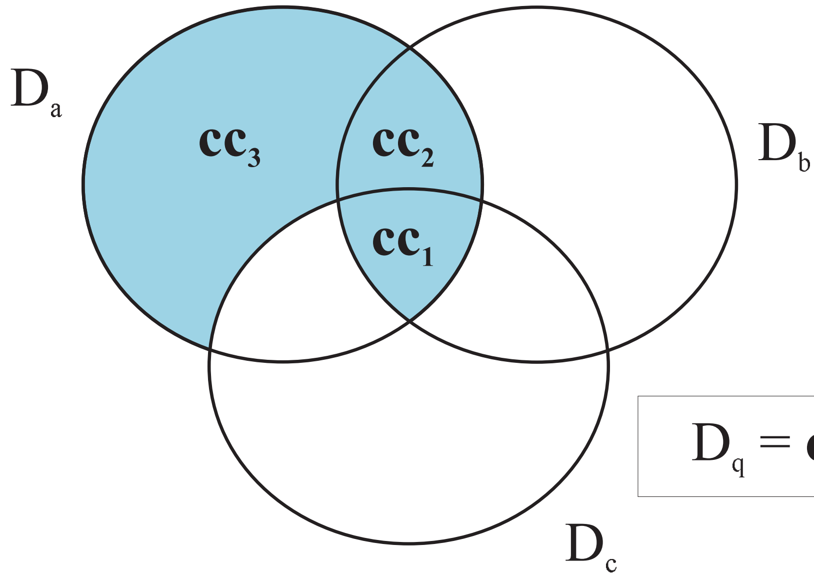
- If $d_j$ contains a term $k_l$ which is closely related to $k_i$, we have
  - $c_{i,l} \sim 1$
  - $\mu_{i,j} \sim 1$
  - and $k_i$ is a good fuzzy index for $d_j$

# Fuzzy IR: An Example




- Consider the query $q = k_a \wedge (k_b \vee \neg k_c)$
- The disjunctive normal form of $q$ is composed of 3 conjunctive components (cc), as follows:
  $\vec{q}_{dnf} = (1,1,1) + (1,1,0) + (1,0,0) = cc_1 + cc_2 + cc_3$
- Let $D_a$, $D_b$ and $D_c$ be the fuzzy sets associated with the terms $k_a$, $k_b$ and $k_c$, respectively

# Fuzzy IR: An Example

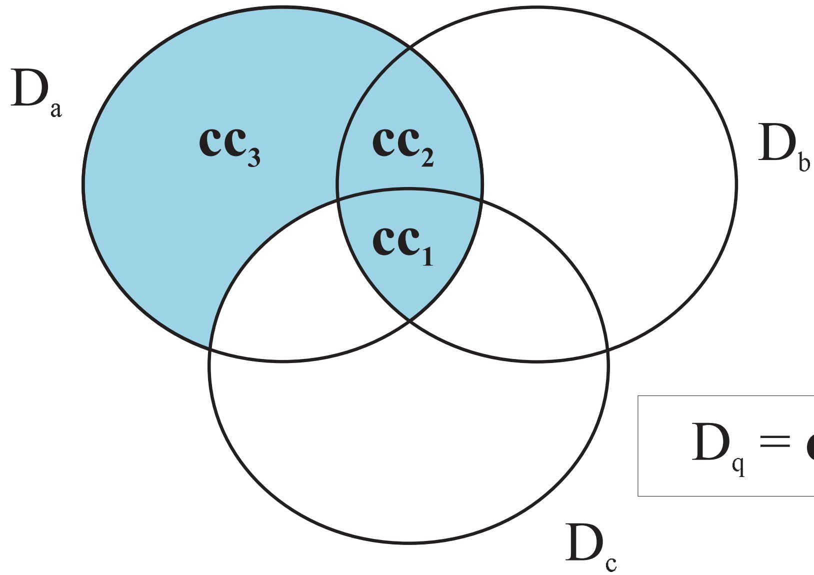



$$D_q = \mathbf{cc}_1 + \mathbf{cc}_2 + \mathbf{cc}_3$$

- Let $\mu_{a,j}$, $\mu_{b,j}$, and $\mu_{c,j}$ be the degrees of memberships of document $d_j$ in the fuzzy sets $D_a$, $D_b$, and $D_c$. Then,

$$\begin{aligned} cc_1 &= \mu_{a,j}\mu_{b,j}\mu_{c,j} \\ cc_2 &= \mu_{a,j}\mu_{b,j}(1-\mu_{c,j}) \\ cc_3 &= \mu_{a,j}(1-\mu_{b,j})(1-\mu_{c,j}) \end{aligned}$$

# Fuzzy IR: An Example




$D_q = \mathbf{cc_1} + \mathbf{cc_2} + \mathbf{cc_3}$

$$
\begin{aligned}
\mu_{q,j} &= \mu_{cc_1+cc_2+cc_3,j} \\
&= 1 - \prod_{i=1}^{3} (1 - \mu_{cc_i,j}) \\
&= 1 - (1 - \mu_{a,j}\mu_{b,j}\mu_{c,j}) \times \\
&\quad (1 - \mu_{a,j}\mu_{b,j}(1 - \mu_{c,j})) \times (1 - \mu_{a,j}(1 - \mu_{b,j})(1 - \mu_{c,j}))
\end{aligned}
$$

# Conclusions

- Fuzzy IR models have been discussed mainly in the literature associated with fuzzy theory
- They provide an interesting framework which naturally embodies the notion of term dependencies
- Experiments with standard test collections are not available

# Alternative Algebraic Models

- Generalized Vector Model
- Latent Semantic Indexing
- Neural Network Model

# Generalized Vector Model

# Generalized Vector Model

- Classic models enforce independence of index terms
- For instance, in the Vector model
  - A set of term vectors $\{\vec{k}_1, \vec{k}_2, \ldots, \vec{k}_t\}$ are linearly independent
  - Frequently, this is interpreted as $\forall_{i,j} \Rightarrow \vec{k}_i \bullet \vec{k}_j = 0$
- In the generalized vector space model, two index term vectors might be non-orthogonal

# Key Idea

- As before, let $w_{i,j}$ be the weight associated with $[k_i, d_j]$ and $V$ = {$k_1$, $k_2$, ..., $k_t$} be the set of all terms
- If the $w_{i,j}$ weights are binary, all patterns of occurrence of terms within docs can be represented by minterms:

$$
\begin{array}{rcl}
 & & (k_1, k_2, k_3, \ldots, k_t) \\
m_1 & = & (0,\ 0,\ 0,\ \ \ldots,\ 0) \\
m_2 & = & (1,\ 0,\ 0,\ \ \ldots,\ 0) \\
m_3 & = & (0,\ 1,\ 0,\ \ \ldots,\ 0) \\
m_4 & = & (1,\ 1,\ 0,\ \ \ldots,\ 0) \\
 & \vdots & \\
m_{2^t} & = & (1,\ 1,\ 1,\ \ \ldots,\ 1)
\end{array}
$$

For instance, $m_2$ indicates documents in which solely the term $k_1$ occurs

# Key Idea

- For any document $d_j$, there is a minterm $m_r$ that includes exactly the terms that occur in the document
- Let us define the following set of minterm vectors $\vec{m}_r$,

$$
\begin{array}{rcl}
 & & 1, 2, \ldots, 2^t \\
\vec{m}_1 & = & (1, 0, \ldots, 0) \\
\vec{m}_2 & = & (0, 1, \ldots, 0) \\
 & \vdots & \\
\vec{m}_{2^t} & = & (0, 0, \ldots, 1)
\end{array}
$$

Notice that we can associate each unit vector $\vec{m}_r$ with a minterm $m_r$, and that $\vec{m}_i \bullet \vec{m}_j = 0$ for all $i \neq j$

# Key Idea

- Pairwise orthogonality among the $\vec{m}_r$ vectors does not imply independence among the index terms
- On the contrary, index terms are now correlated by the $\vec{m}_r$ vectors
  - For instance, the vector $\vec{m}_4$ is associated with the minterm $m_4 = (1, 1, \ldots, 0)$
  - This minterm induces a dependency between terms $k_1$ and $k_2$
  - Thus, if such document exists in a collection, we say that the minterm $m_4$ is active
- The model adopts the idea that co-occurrence of terms induces dependencies among these terms

# Forming the Term Vectors

- Let $on(i, m_r)$ return the weight $\{0, 1\}$ of the index term $k_i$ in the minterm $m_r$
- The vector associated with the term $k_i$ is computed as:

$$\vec{k}_i = \frac{\sum_{\forall r} on(i, m_r)\ c_{i,r}\ \vec{m}_r}{\sqrt{\sum_{\forall r} on(i, m_r)\ c_{i,r}^2}}$$

$$c_{i,r} = \sum_{d_j \mid c(d_j) = m_r} w_{i,j}$$

- Notice that for a collection of size $N$, only $N$ minterms affect the ranking (and not $2^t$)

# Dependency between Index Terms

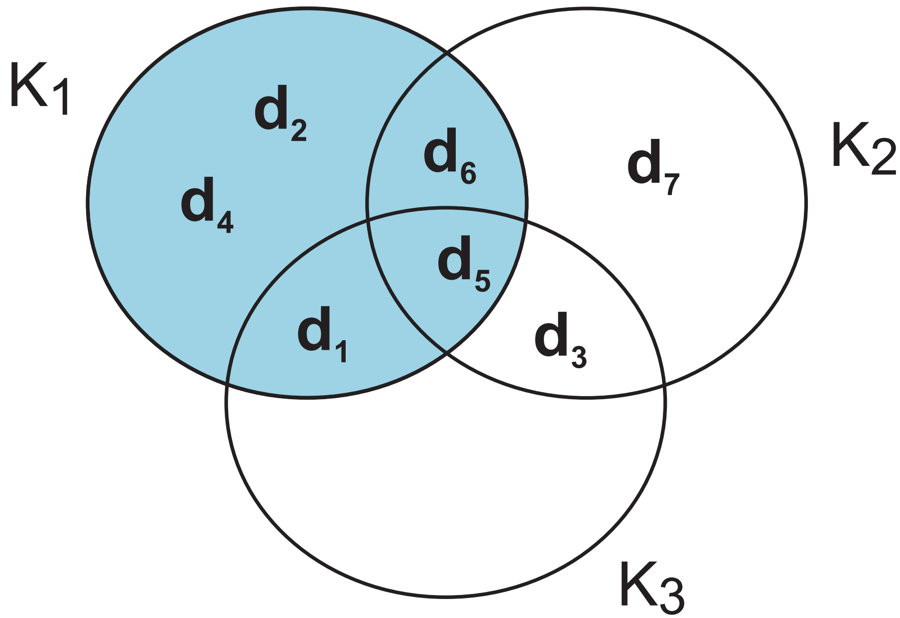
- A degree of correlation between the terms $k_i$ and $k_j$ can now be computed as:

$$\vec{k}_i \bullet \vec{k}_j = \sum_{\forall r} on(i, m_r) \times c_{i,r} \times on(j, m_r) \times c_{j,r}$$

- This degree of correlation sums up the dependencies between $k_i$ and $k_j$ induced by the docs in the collection

# The Generalized Vector Model

- An Example




| | $K_1$ | $K_2$ | $K_3$ |
|---|---|---|---|
| $d_1$ | 2 | 0 | 1 |
| $d_2$ | 1 | 0 | 0 |
| $d_3$ | 0 | 1 | 3 |
| $d_4$ | 2 | 0 | 0 |
| $d_5$ | 1 | 2 | 4 |
| $d_6$ | 1 | 2 | 0 |
| $d_7$ | 0 | 5 | 0 |
| $q$ | 1 | 2 | 3 |

# Computation of $c_{i,r}$

| | $K_1$ | $K_2$ | $K_3$ |
|---|---|---|---|
| $d_1$ | 2 | 0 | 1 |
| $d_2$ | 1 | 0 | 0 |
| $d_3$ | 0 | 1 | 3 |
| $d_4$ | 2 | 0 | 0 |
| $d_5$ | 1 | 2 | 4 |
| $d_6$ | 0 | 2 | 2 |
| $d_7$ | 0 | 5 | 0 |
| $q$ | 1 | 2 | 3 |

| | $K_1$ | $K_2$ | $K_3$ |
|---|---|---|---|
| $d_1 = m_6$ | 1 | 0 | 1 |
| $d_2 = m_2$ | 1 | 0 | 0 |
| $d_3 = m_7$ | 0 | 1 | 1 |
| $d_4 = m_2$ | 1 | 0 | 0 |
| $d_5 = m_8$ | 1 | 1 | 1 |
| $d_6 = m_7$ | 0 | 1 | 1 |
| $d_7 = m_3$ | 0 | 1 | 0 |
| $q = m_8$ | 1 | 1 | 1 |

| | $c_{1,r}$ | $c_{2,r}$ | $c_{3,r}$ |
|---|---|---|---|
| $m_1$ | 0 | 0 | 0 |
| $m_2$ | 3 | 0 | 0 |
| $m_3$ | 0 | 5 | 0 |
| $m_4$ | 0 | 0 | 0 |
| $m_5$ | 0 | 0 | 0 |
| $m_6$ | 2 | 0 | 1 |
| $m_7$ | 0 | 3 | 5 |
| $m_8$ | 1 | 2 | 4 |

# Computation of $\overrightarrow{k_i}$

- $\overrightarrow{k_1} = \frac{(3\vec{m}_2+2\vec{m}_6+\vec{m}_8)}{\sqrt{3^2+2^2+1^2}}$
- $\overrightarrow{k_2} = \frac{(5\vec{m}_3+3\vec{m}_7+2\vec{m}_8)}{\sqrt{5+3+2}}$
- $\overrightarrow{k_3} = \frac{(1\vec{m}_6+5\vec{m}_7+4\vec{m}_8)}{\sqrt{1+5+4}}$

| | $c_{1,r}$ | $c_{2,r}$ | $c_{3,r}$ |
|---|---|---|---|
| $m_1$ | 0 | 0 | 0 |
| $m_2$ | 3 | 0 | 0 |
| $m_3$ | 0 | 5 | 0 |
| $m_4$ | 0 | 0 | 0 |
| $m_5$ | 0 | 0 | 0 |
| $m_6$ | 2 | 0 | 1 |
| $m_7$ | 0 | 3 | 5 |
| $m_8$ | 1 | 2 | 4 |

# Computation of Document Vectors

- $\overrightarrow{d_1} = 2\overrightarrow{k_1} + \overrightarrow{k_3}$
- $\overrightarrow{d_2} = \overrightarrow{k_1}$
- $\overrightarrow{d_3} = \overrightarrow{k_2} + 3\overrightarrow{k_3}$
- $\overrightarrow{d_4} = 2\overrightarrow{k_1}$
- $\overrightarrow{d_5} = \overrightarrow{k_1} + 2\overrightarrow{k_2} + 4\overrightarrow{k_3}$
- $\overrightarrow{d_6} = 2\overrightarrow{k_2} + 2\overrightarrow{k_3}$
- $\overrightarrow{d_7} = 5\overrightarrow{k_2}$
- $\overrightarrow{q} = \overrightarrow{k_1} + 2\overrightarrow{k_2} + 3\overrightarrow{k_3}$

| | $K_1$ | $K_2$ | $K_3$ |
|---|---|---|---|
| $d_1$ | 2 | 0 | 1 |
| $d_2$ | 1 | 0 | 0 |
| $d_3$ | 0 | 1 | 3 |
| $d_4$ | 2 | 0 | 0 |
| $d_5$ | 1 | 2 | 4 |
| $d_6$ | 0 | 2 | 2 |
| $d_7$ | 0 | 5 | 0 |
| $q$ | 1 | 2 | 3 |

# Conclusions

- Model considers correlations among index terms
- Not clear in which situations it is superior to the standard Vector model
- Computation costs are higher
- Model does introduce interesting new ideas

# Latent Semantic Indexing

# Latent Semantic Indexing

- Classic IR might lead to poor retrieval due to:
  - unrelated documents might be included in the answer set
  - relevant documents that do not contain at least one index term are not retrieved
  - **Reasoning**: retrieval based on index terms is vague and noisy
- The user information need is more related to concepts and ideas than to index terms
- A document that shares concepts with another document known to be relevant might be of interest

# Latent Semantic Indexing

- The idea here is to map documents and queries into a dimensional space composed of concepts
- Let
  - $t$: total number of index terms
  - $N$: number of documents
  - $\mathbf{M} = [m_{ij}]$: term-document matrix $t \times N$
- To each element of $\mathbf{M}$ is assigned a weight $w_{i,j}$ associated with the term-document pair $[k_i, d_j]$
  - The weight $w_{i,j}$ can be based on a *tf-idf* weighting scheme

# Latent Semantic Indexing

- The matrix $\mathbf{M} = [m_{ij}]$ can be decomposed into three components using singular value decomposition

$$\mathbf{M} = \mathbf{K} \cdot \mathbf{S} \cdot \mathbf{D}^T$$

- were
  - $\mathbf{K}$ is the matrix of eigenvectors derived from $\mathbf{C} = \mathbf{M} \cdot \mathbf{M}^T$
  - $\mathbf{D}^T$ is the matrix of eigenvectors derived from $\mathbf{M}^T \cdot \mathbf{M}$
  - $\mathbf{S}$ is an $r \times r$ diagonal matrix of singular values where $r = \min(t, N)$ is the rank of $\mathbf{M}$

# Computing an Example

- Let $\mathbf{M}^T = [m_{ij}]$ be given by

| | $K_1$ | $K_2$ | $K_3$ | $q \bullet d_j$ |
|---|---|---|---|---|
| $d_1$ | 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 | |

- Compute the matrices $\mathbf{K}$, $\mathbf{S}$, and $\mathbf{D}^t$

# Latent Semantic Indexing

- In the matrix $\mathbf{S}$, consider that only the $s$ largest singular values are selected
- Keep the corresponding columns in $\mathbf{K}$ and $\mathbf{D}^T$
- The resultant matrix is called $\mathbf{M}_s$ and is given by

$$\mathbf{M}_s = \mathbf{K}_s \cdot \mathbf{S}_s \cdot \mathbf{D}_s^T$$

- where $s$, $s < r$, is the dimensionality of a reduced concept space
- The parameter $s$ should be
  - large enough to allow fitting the characteristics of the data
  - small enough to filter out the non-relevant representational details

# Latent Ranking

- The relationship between any two documents in $s$ can be obtained from the $\mathrm{M}_s^T \cdot \mathrm{M}_s$ matrix given by

$$\begin{aligned}
\mathbf{M}_s^T \cdot \mathbf{M}_s &= (\mathbf{K}_s \cdot \mathbf{S}_s \cdot \mathbf{D}_s^T)^T \cdot \mathbf{K}_s \cdot \mathbf{S}_s \cdot \mathbf{D}_s^T \\
&= \mathbf{D}_s \cdot \mathbf{S}_s \cdot \mathbf{K}_s^T \cdot \mathbf{K}_s \cdot \mathbf{S}_s \cdot \mathbf{D}_s^T \\
&= \mathbf{D}_s \cdot \mathbf{S}_s \cdot \mathbf{S}_s \cdot \mathbf{D}_s^T \\
&= (\mathbf{D}_s \cdot \mathbf{S}_s) \cdot (\mathbf{D}_s \cdot \mathbf{S}_s)^T
\end{aligned}$$

- In the above matrix, the $(i, j)$ element quantifies the relationship between documents $d_i$ and $d_j$

# Latent Ranking

- The user query can be modelled as a pseudo-document in the original $\mathbf{M}$ matrix
- Assume the query is modelled as the document numbered $0$ in the $\mathbf{M}$ matrix
- The matrix $\mathbf{M}_s^T \cdot \mathbf{M}_s$ quantifies the relationship between any two documents in the reduced concept space
- The first row of this matrix provides the rank of all the documents with regard to the user query

# Conclusions

- Latent semantic indexing provides an interesting conceptualization of the IR problem
- Thus, it has its value as a new theoretical framework
- From a practical point of view, the latent semantic indexing model has not yielded encouraging results

# Neural Network Model

# Neural Network Model

- Classic IR:
  - Terms are used to index documents and queries
  - Retrieval is based on index term matching
- Motivation:
  - Neural networks are known to be good pattern matchers

# Neural Network Model

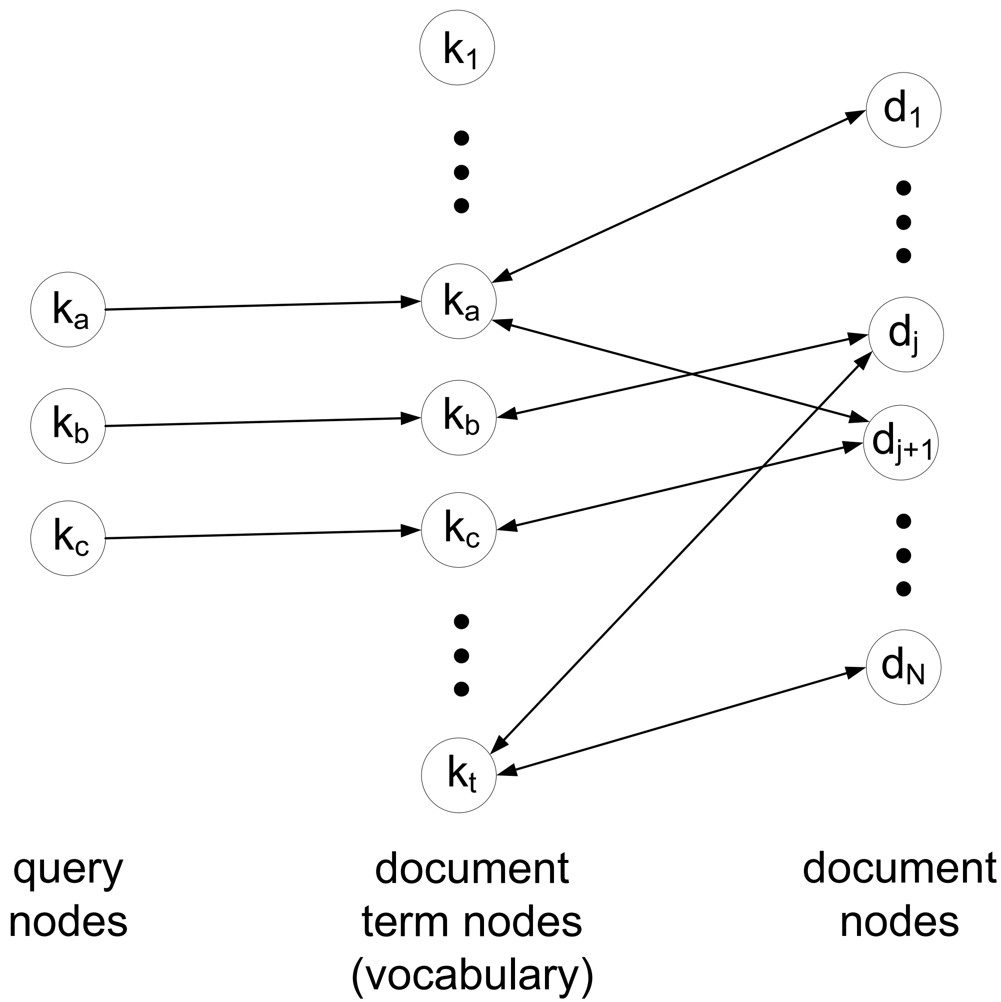
- The human brain is composed of billions of neurons
- Each neuron can be viewed as a small processing unit
- A neuron is stimulated by input signals and emits output signals in reaction
- A chain reaction of propagating signals is called a **spread activation process**
- As a result of spread activation, the brain might command the body to take physical reactions

# Neural Network Model

- A neural network is an oversimplified representation of the neuron interconnections in the human brain:
  - nodes are processing units
  - edges are synaptic connections
  - the strength of a propagating signal is modelled by a weight assigned to each edge
  - the state of a node is defined by its **activation level**
  - depending on its activation level, a node might issue an output signal

# Neural Network for IR

- A neural network model for information retrieval



# Neural Network for IR

- Three layers network: one for the query terms, one for the document terms, and a third one for the documents
- Signals propagate across the network
- First level of propagation:
  - Query terms issue the first signals
  - These signals propagate across the network to reach the document nodes
- Second level of propagation:
  - Document nodes might themselves generate new signals which affect the document term nodes
  - Document term nodes might respond with new signals of their own

# Quantifying Signal Propagation

- Normalize signal strength (MAX = 1)
- Query terms emit initial signal equal to 1
- Weight associated with an edge from a query term node $k_i$ to a document term node $k_i$:

$$\overline{w}_{i,q} = \frac{w_{i,q}}{\sqrt{\sum_{i=1}^{t} \quad w_{i,q}^2}}$$

- Weight associated with an edge from a document term node $k_i$ to a document node $d_j$:

$$\overline{w}_{i,j} = \frac{w_{i,j}}{\sqrt{\sum_{i=1}^{t} \quad w_{i,j}^2}}$$

# Quantifying Signal Propagation

- After the first level of signal propagation, the activation level of a document node $d_j$ is given by:

$$\sum_{i=1}^{t} \overline{w}_{i,q}\, \overline{w}_{i,j} = \frac{\sum_{i=1}^{t} \; w_{i,q}\, w_{i,j}}{\sqrt{\sum_{i=1}^{t} \; w_{i,q}^2} \times \sqrt{\sum_{i=1}^{t} \; w_{i,j}^2}}$$

  which is exactly the ranking of the Vector model

- New signals might be exchanged among document term nodes and document nodes
- A minimum threshold should be enforced to avoid spurious signal generation

# Conclusions

- Model provides an interesting formulation of the IR problem
- Model has not been tested extensively
- It is not clear the improvements that the model might provide

# Chapter 3

# Modeling

**Part III: Alternative Probabilistic Models**

- BM25
- Language Models
- Divergence from Randomness
- Belief Network Models
- Other Models

# BM25 (Best Match 25)

# BM25 (Best Match 25)

- BM25 was created as the result of a series of experiments on variations of the probabilistic model
- A good term weighting is based on three principles
  - inverse document frequency
  - term frequency
  - document length normalization
- The classic probabilistic model covers only the first of these principles
- This reasoning led to a series of experiments with the Okapi system, which led to the BM25 ranking formula

# BM1, BM11 and BM15 Formulas

- At first, the Okapi system used the Equation below as ranking formula

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

  which is the equation used in the probabilistic model, when no relevance information is provided

- It was referred to as the BM1 formula (*Best Match 1*)

# BM1, BM11 and BM15 Formulas

- The first idea for improving the ranking was to introduce a **term-frequency** factor $\mathcal{F}_{i,j}$ in the BM1 formula
- This factor, after some changes, evolved to become

$$\mathcal{F}_{i,j} = S_1 \times \frac{f_{i,j}}{K_1 + f_{i,j}}$$

  where

  - $f_{i,j}$ is the frequency of term $k_i$ within document $d_j$
  - $K_1$ is a constant setup experimentally for each collection
  - $S_1$ is a scaling constant, normally set to $S_1 = (K_1 + 1)$

- If $K_1 = 0$, this whole factor becomes equal to 1 and bears no effect in the ranking

# BM1, BM11 and BM15 Formulas

- The next step was to modify the $\mathcal{F}_{i,j}$ factor by adding **document length normalization** to it, as follows:

$$\mathcal{F}'_{i,j} = S_1 \times \frac{f_{i,j}}{\frac{K_1 \times len(d_j)}{avg\_doclen} + f_{i,j}}$$

where

  - $len(d_j)$ is the length of document $d_j$ (computed, for instance, as the number of terms in the document)
  - $avg\_doclen$ is the average document length for the collection

# BM1, BM11 and BM15 Formulas

- Next, a correction factor $G_{j,q}$ dependent on the document and query lengths was added

$$\mathcal{G}_{j,q} = K_2 \times len(q) \times \frac{avg\_doclen - len(d_j)}{avg\_doclen + len(d_j)}$$

where

  - $len(q)$ is the query length (number of terms in the query)
  - $K_2$ is a constant

# BM1, BM11 and BM15 Formulas

- A third additional factor, aimed at taking into account term frequencies within queries, was defined as

$$\mathcal{F}_{i,q} = S_3 \times \frac{f_{i,q}}{K_3 + f_{i,q}}$$

where

  - $f_{i,q}$ is the frequency of term $k_i$ within query $q$
  - $K_3$ is a constant
  - $S_3$ is an scaling constant related to $K_3$, normally set to $S_3 = (K_3 + 1)$

# BM1, BM11 and BM15 Formulas

- Introduction of these three factors led to various BM (Best Matching) formulas, as follows:

$$sim_{BM1}(d_j, q) \sim \sum_{k_i[q,d_j]} \log \left( \frac{N - n_i + 0.5}{n_i + 0.5} \right)$$

$$sim_{BM15}(d_j, q) \sim \mathcal{G}_{j,q} + \sum_{k_i[q,d_j]} \mathcal{F}_{i,j} \times \mathcal{F}_{i,q} \times \log \left( \frac{N - n_i + 0.5}{n_i + 0.5} \right)$$

$$sim_{BM11}(d_j, q) \sim \mathcal{G}_{j,q} + \sum_{k_i[q,d_j]} \mathcal{F}'_{i,j} \times \mathcal{F}_{i,q} \times \log \left( \frac{N - n_i + 0.5}{n_i + 0.5} \right)$$

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

# BM1, BM11 and BM15 Formulas

- Experiments using TREC data have shown that BM11 outperforms BM15
- Further, empirical considerations can be used to simplify the previous equations, as follows:
  - Empirical evidence suggests that a best value of $K_2$ is $0$, which eliminates the $G_{j,q}$ factor from these equations
  - Further, good estimates for the scaling constants $S_1$ and $S_3$ are $K_1 + 1$ and $K_3 + 1$, respectively
  - Empirical evidence also suggests that making $K_3$ very large is better. As a result, the $\mathcal{F}_{i,q}$ factor is reduced simply to $f_{i,q}$
  - For short queries, we can assume that $f_{i,q}$ is 1 for all terms

# BM1, BM11 and BM15 Formulas

- These considerations lead to simpler equations as follows

$$sim_{BM1}(d_j, q) \sim \sum_{k_i[q,d_j]} \log\left(\frac{N - n_i + 0.5}{n_i + 0.5}\right)$$

$$sim_{BM15}(d_j, q) \sim \sum_{k_i[q,d_j]} \frac{(K_1 + 1)f_{i,j}}{(K_1 + f_{i,j})} \times \log\left(\frac{N - n_i + 0.5}{n_i + 0.5}\right)$$

$$sim_{BM11}(d_j, q) \sim \sum_{k_i[q,d_j]} \frac{(K_1 + 1)f_{i,j}}{\frac{K_1\ len(d_j)}{avg\_doclen} + f_{i,j}} \times \log\left(\frac{N - n_i + 0.5}{n_i + 0.5}\right)$$

# BM25 Ranking Formula

- BM25: combination of the BM11 and BM15
- The motivation was to combine the BM11 and BM25 term frequency factors as follows

$$\mathcal{B}_{i,j} = \frac{(K_1 + 1) f_{i,j}}{K_1 \left[ (1 - b) + b \frac{len(d_j)}{avg\_doclen} \right] + f_{i,j}}$$

where $b$ is a constant with values in the interval $[0, 1]$

  - If $b = 0$, it reduces to the BM15 term frequency factor
  - If $b = 1$, it reduces to the BM11 term frequency factor
  - For values of $b$ between 0 and 1, the equation provides a combination of BM11 with BM15

# BM25 Ranking Formula

- The ranking equation for the BM25 model can then be written as

$$sim_{BM25}(d_j, q) \sim \sum_{k_i[q,d_j]} \mathcal{B}_{i,j} \times \log \left( \frac{N - n_i + 0.5}{n_i + 0.5} \right)$$

  where $K_1$ and $b$ are empirical constants

  - $K_1 = 1$ works well with real collections
  - $b$ should be kept closer to 1 to emphasize the document length normalization effect present in the BM11 formula
  - For instance, $b = 0.75$ is a reasonable assumption
  - Constants values can be fine tunned for particular collections through proper experimentation

# BM25 Ranking Formula

- Unlike the probabilistic model, the BM25 formula can be computed without relevance information
- There is consensus that BM25 outperforms the classic vector model for general collections
- Thus, it has been used as a baseline for evaluating new ranking functions, in substitution to the classic vector model

# Language Models

# Language Models

- Language models are used in many natural language processing applications
  - Ex: part-of-speech tagging, speech recognition, machine translation, and information retrieval
- To illustrate, the regularities in spoken language can be modeled by probability distributions
- These distributions can be used to predict the likelihood that the next token in the sequence is a given word
- These probability distributions are called **language models**

# Language Models

- A **language model** for IR is composed of the following components
  - A set of document language models, one per document $d_j$ of the collection
  - A probability distribution function that allows estimating the likelihood that a document language model $M_j$ *generates* each of the query terms
  - A ranking function that combines these generating probabilities for the query terms into a rank of document $d_j$ with regard to the query

# Statistical Foundation

- Let $S$ be a sequence of $r$ consecutive terms that occur in a document of the collection:

$$S = k_1, k_2, \ldots, k_r$$

- An $n$-gram language model uses a Markov process to assign a probability of occurrence to $S$:

$$P_n(S) = \prod_{i=1}^{r} P(k_i | k_{i-1}, k_{i-2}, \ldots, k_{i-(n-1)})$$

where $n$ is the order of the Markov process

- The occurrence of a term depends on observing the $n - 1$ terms that precede it in the text

# Statistical Foundation

- **Unigram language model** $(n = 1)$: the estimatives are based on the occurrence of individual words
- **Bigram language model** $(n = 2)$: the estimatives are based on the co-occurrence of pairs of words
- Higher order models such as **Trigram language models ($n = 3$)** are usually adopted for speech recognition
- **Term independence assumption**: in the case of IR, the impact of word order is less clear
  - As a result, Unigram models have been used extensively in IR

# Multinomial Process

- Ranking in a language model is provided by estimating $P(q|M_j)$
- Several researchs have proposed the adoption of a multinomial process to generate the query
- According to this process, if we assume that the query terms are independent among themselves (unigram model), we can write:

$$P(q|M_j) = \prod_{k_i \in q} P(k_i|M_j)$$

# Multinomial Process

- By taking logs on both sides

$$
\begin{aligned}
\log P(q|M_j) &= \sum_{k_i \in q} \log P(k_i|M_j) \\
&= \sum_{k_i \in q \wedge d_j} \log P_{\in}(k_i|M_j) + \sum_{k_i \in q \wedge \neg d_j} \log P_{\notin}(k_i|M_j) \\
&= \sum_{k_i \in q \wedge d_j} \log \left( \frac{P_{\in}(k_i|M_j)}{P_{\notin}(k_i|M_j)} \right) + \sum_{k_i \in q} \log P_{\notin}(k_i|M_j)
\end{aligned}
$$

where $P_{\in}$ and $P_{\notin}$ are two distinct probability distributions:

- The first is a distribution for the query terms in the document
- The second is a distribution for the query terms not in the document

# Multinomial Process

- For the second distribution, statistics are derived from all the document collection
- Thus, we can write

$$P_{\not\in}(k_i|M_j) = \alpha_j P(k_i|C)$$

where $\alpha_j$ is a parameter associated with document $d_j$ and $P(k_i|C)$ is a collection $C$ language model

# Multinomial Process

- $P(k_i|C)$ can be estimated in different ways
- For instance, Hiemstra suggests an idf-like estimative:

$$P(k_i|C) = \frac{n_i}{\sum_i n_i}$$

where $n_i$ is the number of docs in which $k_i$ occurs

- Miller, Leek, and Schwartz suggest

$$P(k_i|C) = \frac{F_i}{\sum_i F_i}$$

where $F_i = \sum_j f_{i,j}$

# Multinomial Process

- Thus, we obtain

$$
\begin{aligned}
\log P(q|M_j) &= \sum_{k_i \in q \wedge d_j} \log \left( \frac{P_{\in}(k_i|M_j)}{\alpha_j P(k_i|C)} \right) + n_q \log \alpha_j + \sum_{k_i \in q} \log P(k_i|C) \\
&\sim \sum_{k_i \in q \wedge d_j} \log \left( \frac{P_{\in}(k_i|M_j)}{\alpha_j P(k_i|C)} \right) + n_q \log \alpha_j
\end{aligned}
$$

where $n_q$ stands for the query length and the last sum was dropped because it is constant for all documents

# Multinomial Process

- The ranking function is now composed of two separate parts
- The **first part** assigns weights to each query term that appears in the document, according to the expression

$$\log \left( \frac{P_{\in}(k_i|M_j)}{\alpha_j P(k_i|C)} \right)$$

- This term weight plays a role analogous to the tf plus idf weight components in the vector model
- Further, the parameter $\alpha_j$ can be used for document length normalization

# Multinomial Process

- The **second part** assigns a fraction of probability mass to the query terms that are not in the document—a process called **smoothing**
- The combination of a multinomial process with smoothing leads to a ranking formula that naturally includes $tf$, $idf$, and document length normalization
- That is, smoothing plays a key role in modern language modeling, as we now discuss

# Smoothing

- In our discussion, we estimated $P_{\not\in}(k_i|M_j)$ using $P(k_i|C)$ to avoid assigning zero probability to query terms not in document $d_j$
- This process, called **smoothing**, allows fine tuning the ranking to improve the results.
- One popular smoothing technique is to move some mass probability from the terms in the document to the terms not in the document, as follows:

$$P(k_i|M_j) = \begin{cases} P^s_{\in}(k_i|M_j) & \text{if } k_i \in d_j \\ \alpha_j P(k_i|C) & \text{otherwise} \end{cases}$$

where $P^s_{\in}(k_i|M_j)$ is the **smoothed distribution** for terms in document $d_j$

# Smoothing

- Since $\sum_i P(k_i|M_j) = 1$, we can write

$$\sum_{k_i \in d_j} P^s_{\in}(k_i|M_j) + \sum_{k_i \notin d_j} \alpha_j P(k_i|C) = 1$$

- That is,

$$\alpha_j = \frac{1 - \sum_{k_i \in d_j} P^s_{\in}(k_i|M_j)}{1 - \sum_{k_i \in d_j} P(k_i|C)}$$

# Smoothing

- Under the above assumptions, the smoothing parameter $\alpha_j$ is also a function of $P^s_\in(k_i|M_j)$
- As a result, distinct smoothing methods can be obtained through distinct specifications of $P^s_\in(k_i|M_j)$
- Examples of smoothing methods:
  - Jelinek-Mercer Method
  - Bayesian Smoothing using Dirichlet Priors

# Jelinek-Mercer Method

- The idea is to do a linear interpolation between the document frequency and the collection frequency distributions:

$$P^s_{\in}(k_i|M_j, \lambda) = (1-\lambda)\frac{f_{i,j}}{\sum_i f_{i,j}} + \lambda\frac{F_i}{\sum_i F_i}$$

  where $0 \leq \lambda \leq 1$

- It can be shown that

$$\alpha_j = \lambda$$

- Thus, the larger the values of $\lambda$, the larger is the effect of smoothing

# Dirichlet smoothing

- In this method, the language model is a multinomial distribution in which the conjugate prior probabilities are given by the Dirichlet distribution
- This leads to

$$P_{\in}^{s}(k_i|M_j,\lambda) = \frac{f_{i,j} + \lambda \frac{F_i}{\sum_i F_i}}{\sum_i f_{i,j} + \lambda}$$

- As before, closer is $\lambda$ to 0, higher is the influence of the term document frequency. As $\lambda$ moves towards 1, the influence of the term collection frequency increases

# Dirichlet smoothing

- Contrary to the Jelinek-Mercer method, this influence is always partially mixed with the document frequency
- It can be shown that

$$\alpha_j = \frac{\lambda}{\sum_i f_{i,j} + \lambda}$$

- As before, the larger the values of $\lambda$, the larger is the effect of smoothing

# Smoothing Computation

- In both smoothing methods above, computation can be carried out efficiently
- All frequency counts can be obtained directly from the index
- The values of $\alpha_j$ can be precomputed for each document
- Thus, the complexity is analogous to the computation of a vector space ranking using tf-idf weights

# Applying Smoothing to Ranking

- The IR ranking in a multinomial language model is computed as follows:
  - compute $P^s_\in(k_i|M_j)$ using a smoothing method
  - compute $P(k_i|C)$ using $\frac{n_i}{\sum_i n_i}$ or $\frac{F_i}{\sum_i F_i}$
  - compute $\alpha_j$ from the Equation $\alpha_j = \frac{1 - \sum_{k_i \in d_j} P^s_\in(k_i|M_j)}{1 - \sum_{k_i \in d_j} P(k_i|C)}$
  - compute the ranking using the formula

$$\log P(q|M_j) = \sum_{k_i \in q \wedge d_j} \log \left( \frac{P^s_\in(k_i|M_j)}{\alpha_j P(k_i|C)} \right) + n_q \log \alpha_j$$

# Bernoulli Process

- The first application of languages models to IR was due to Ponte & Croft. They proposed a Bernoulli process for generating the query, as we now discuss
- Given a document $d_j$, let $M_j$ be a reference to a language model for that document
- If we assume independence of index terms, we can compute $P(q|M_j)$ using a multivariate Bernoulli process:

$$P(q|M_j) = \prod_{k_i \in q} P(k_i|M_j) \ \times \ \prod_{k_i \notin q} [1 - P(k_i|M_j)]$$

  where $P(k_i|M_j)$ are term probabilities
- This is analogous to the expression for ranking computation in the classic probabilistic model

# Bernoulli process

- A simple estimate of the term probabilities is

$$P(k_i|M_j) = \frac{f_{i,j}}{\sum_\ell f_{\ell,j}}$$

  which computes the probability that term $k_i$ will be produced by a random draw (taken from $d_j$)

- However, the probability will become zero if $k_i$ does not occur in the document
- Thus, we assume that a non-occurring term is related to $d_j$ with the probability $P(k_i|C)$ of observing $k_i$ in the whole collection $C$

# Bernoulli process

- $P(k_i|C)$ can be estimated in different ways
- For instance, Hiemstra suggests an idf-like estimative:

$$P(k_i|C) = \frac{n_i}{\sum_\ell n_\ell}$$

where $n_i$ is the number of docs in which $k_i$ occurs

- Miller, Leek, and Schwartz suggest

$$P(k_i|C) = \frac{F_i}{\sum_\ell F_\ell} \quad \text{where} \quad F_i = \sum_j f_{i,j}$$

- This last equation for $P(k_i|C)$ is adopted here

# Bernoulli process

- As a result, we redefine $P(k_i|M_j)$ as follows:

$$P(k_i|M_j) = \begin{cases} \frac{f_{i,j}}{\sum_i f_{i,j}} & \text{if } f_{i,j} > 0 \\ \\ \frac{F_i}{\sum_i F_i} & \text{if } f_{i,j} = 0 \end{cases}$$

- In this expression, $P(k_i|M_j)$ estimation is based only on the document $d_j$ when $f_{i,j} > 0$
- This is clearly undesirable because it leads to instability in the model

# Bernoulli process

- This drawback can be accomplished through an average computation as follows

$$P(k_i) = \frac{\sum_{j|k_i \in d_j} P(k_i|M_j)}{n_i}$$

- That is, $P(k_i)$ is an estimate based on the language models of all documents that contain term $k_i$
- However, it is the same for all documents that contain term $k_i$
- That is, using $P(k_i)$ to predict the generation of term $k_i$ by the $M_j$ involves a risk

# Bernoulli process

- To fix this, let us define the average frequency $\overline{f}_{i,j}$ of term $k_i$ in document $d_j$ as

$$\overline{f}_{i,j} = P(k_i) \times \sum_i f_{i,j}$$

# Bernoulli process

- The risk $R_{i,j}$ associated with using $\overline{f}_{i,j}$ can be quantified by a geometric distribution:

$$R_{i,j} = \left( \frac{1}{1 + \overline{f}_{i,j}} \right) \times \left( \frac{\overline{f}_{i,j}}{1 + \overline{f}_{i,j}} \right)^{f_{i,j}}$$

- For terms that occur very frequently in the collection, $\overline{f}_{i,j} \gg 0$ and $R_{i,j} \sim 0$
- For terms that are rare both in the document and in the collection, $f_{i,j} \sim 1$, $\overline{f}_{i,j} \sim 1$, and $R_{i,j} \sim 0.25$

# Bernoulli process

- Let us refer the probability of observing term $k_i$ according to the language model $M_j$ as $P_R(k_i|M_j)$
- We then use the risk factor $R_{i,j}$ to compute $P_R(k_i|M_j)$, as follows

$$P_R(k_i|M_j) = \begin{cases} P(k_i|M_j)^{(1-R_{i,j})} \times P(k_i)^{R_{i,j}} & \text{if } f_{i,j} > 0 \\ \\ \frac{F_i}{\sum_i F_i} & \text{otherwise} \end{cases}$$

- In this formulation, if $R_{i,j} \sim 0$ then $P_R(k_i|M_j)$ is basically a function of $P(k_i|M_j)$
- Otherwise, it is a mix of $P(k_i)$ and $P(k_i|M_j)$

# Bernoulli process

- Substituting into original $P(q|M_j)$ Equation, we obtain

$$P(q|M_j) = \prod_{k_i \in q} P_R(k_i|M_j) \times \prod_{k_i \notin q} [1 - P_R(k_i|M_j)]$$

  which computes the probability of generating the query from the language (document) model

- This is the basic formula for ranking computation in a language model based on a Bernoulli process for generating the query

# Divergence from Randomness

# Divergence from Randomness

- A distinct probabilistic model has been proposed by Amati and Rijsbergen
- The idea is to compute term weights by measuring the divergence between a term distribution produced by a random process and the actual term distribution
- Thus, the name **divergence from randomness**
- The model is based on two fundamental assumptions, as follows

# Divergence from Randomness

- First assumption:
  - Not all words are equally important for describing the content of the documents
  - Words that carry little information are assumed to be **randomly distributed** over the whole document collection $C$
  - Given a term $k_i$, its probability distribution over the whole collection is referred to as $P(k_i|C)$
  - The amount of information associated with this distribution is given by

$$\boxed{-\log \; P(k_i|C)}$$

  - By modifying this probability function, we can implement distinct **notions of term randomness**

# Divergence from Randomness

- Second assumption:
  - A complementary term distribution can be obtained by considering just the subset of documents that contain term $k_i$
  - This subset is referred to as the **elite set**
  - The corresponding probability distribution, computed with regard to document $d_j$, is referred to as $P(k_i|d_j)$
  - Smaller the probability of observing a term $k_i$ in a document $d_j$, more rare and important is the term considered to be
  - Thus, the amount of information associated with the term in the elite set is defined as

$$\boxed{1 - P(k_i|d_j)}$$

# Divergence from Randomness

- Given these assumptions, the weight $w_{i,j}$ of a term $k_i$ in a document $d_j$ is defined as

$$w_{i,j} = [-\log\ P(k_i|C)] \ \times \ [1 - P(k_i|d_j)]$$

- Two term distributions are considered: in the collection and in the subset of docs in which it occurs
- The rank $R(d_j, q)$ of a document $d_j$ with regard to a query $q$ is then computed as

$$\boxed{R(d_j, q) = \sum_{k_i \in q} f_{i,q} \times w_{i,j}}$$

where $f_{i,q}$ is the frequency of term $k_i$ in the query

# Random Distribution

- To compute the distribution of terms in the collection, distinct probability models can be considered
- For instance, consider that Bernoulli trials are used to model the occurrences of a term in the collection
- To illustrate, consider a collection with 1,000 documents and a term $k_i$ that occurs 10 times in the collection
- Then, the probability of observing 4 occurrences of term $k_i$ in a document is given by

$$P(k_i|C) = \binom{10}{4}\left(\frac{1}{1000}\right)^4\left(1 - \frac{1}{1000}\right)^6$$

which is a standard binomial distribution

# Random Distribution

- In general, let $p = 1/N$ be the probability of observing a term in a document, where $N$ is the number of docs
- The probability of observing $f_{i,j}$ occurrences of term $k_i$ in document $d_j$ is described by a binomial distribution:

$$P(k_i|C) = \binom{F_i}{f_{i,j}} p^{f_{i,j}} \times (1-p)^{F_i - f_{i,j}}$$

- Define

$$\lambda_i = p \times F_i$$

and assume that $p \to 0$ when $N \to \infty$, but that $\lambda_i = p \times F_i$ remains constant

# Random Distribution

- Under these conditions, we can aproximate the binomial distribution by a Poisson process, which yields

$$P(k_i|C) = \frac{e^{-\lambda_i} \ \lambda_i^{f_{i,j}}}{f_{i,j}!}$$

# Random Distribution

- The amount of information associated with term $k_i$ in the collection can then be computed as

$$
\begin{aligned}
-\log\ P(k_i|C) &= -\log\left(\frac{e^{-\lambda_i}\ \lambda_i^{f_{i,j}}}{f_{i,j}!}\right) \\
&\approx -f_{i,j}\log\lambda_i + \lambda_i\log e + \log(f_{i,j}!) \\
&\approx f_{i,j}\log\left(\frac{f_{i,j}}{\lambda_i}\right) + \left(\lambda_i + \frac{1}{12f_{i,j}+1} - f_{i,j}\right)\log e \\
&\quad + \frac{1}{2}\ \log(2\pi f_{i,j})
\end{aligned}
$$

in which the logarithms are in base 2 and the factorial term $f_{i,j}!$ was approximated by the **Stirling's formula**

$$
f_{i,j}! \approx \sqrt{2\pi}\ f_{i,j}^{(f_{i,j}+0.5)}\ e^{-f_{i,j}}\ e^{(12f_{i,j}+1)^{-1}}
$$

# Random Distribution

- Another approach is to use a Bose-Einstein distribution and approximate it by a geometric distribution:

$$P(k_i|C) \approx p \ \times \ p^{f_{i,j}}$$

  where $p = 1/(1+\lambda_i)$

- The amount of information associated with term $k_i$ in the collection can then be computed as

$$-\log \ P(k_i|C) \approx -\log \left( \frac{1}{1+\lambda_i} \right) - f_{i,j} \times \log \left( \frac{\lambda_i}{1+\lambda_i} \right)$$

  which provides a second form of computing the term distribution over the whole collection

# Distribution over the Elite Set

- The amount of information associated with term distribution in elite docs can be computed by using Laplace's law of succession

$$1 - P(k_i|d_j) = \frac{1}{f_{i,j} + 1}$$

- Another possibility is to adopt the ratio of two Bernoulli processes, which yields

$$1 - P(k_i|d_j) = \frac{F_i + 1}{n_i \times (f_{i,j} + 1)}$$

where $n_i$ is the number of documents in which the term occurs, as before

# Normalization

- These formulations do not take into account the length of the document $d_j$. This can be done by normalizing the term frequency $f_{i,j}$
- Distinct normalizations can be used, such as

$$f'_{i,j} = f_{i,j} \times \frac{avg\_doclen}{len(d_j)}$$

or

$$f'_{i,j} = f_{i,j} \times \log\left(1 + \frac{avg\_doclen}{len(d_j)}\right)$$

where $avg\_doclen$ is the average document length in the collection and $len(d_j)$ is the length of document $d_j$

# Normalization

- To compute $w_{i,j}$ weights using normalized term frequencies, just substitute the factor $f_{i,j}$ by $f'_{i,j}$
- In here we consider that a same normalization is applied for computing $P(k_i|C)$ and $P(k_i|d_j)$
- By combining different forms of computing $P(k_i|C)$ and $P(k_i|d_j)$ with different normalizations, various ranking formulas can be produced

# Bayesian Network Models

# Bayesian Inference

- One approach for developing probabilistic models of IR is to use **Bayesian belief networks**
- Belief networks provide a clean formalism for combining distinct sources of evidence
  - Types of evidences: past queries, past feedback cycles, distinct query formulations, etc.
- In here we discuss two models:
  - **Inference network**, proposed by Turtle and Croft
  - **Belief network model**, proposed by Ribeiro-Neto and Muntz
- Before proceeding, we briefly introduce **Bayesian networks**

# Bayesian Networks

- Bayesian networks are **directed acyclic graphs (DAGs)** in which
  - **the nodes** represent random variables
  - **the arcs** portray causal relationships between these variables
  - **the strengths** of these causal influences are expressed by conditional probabilities
- The **parents** of a node are those judged to be direct causes for it
- This **causal relationship** is represented by a link directed from each parent node to the child node
- The **roots** of the network are the nodes without parents

# Bayesian Networks

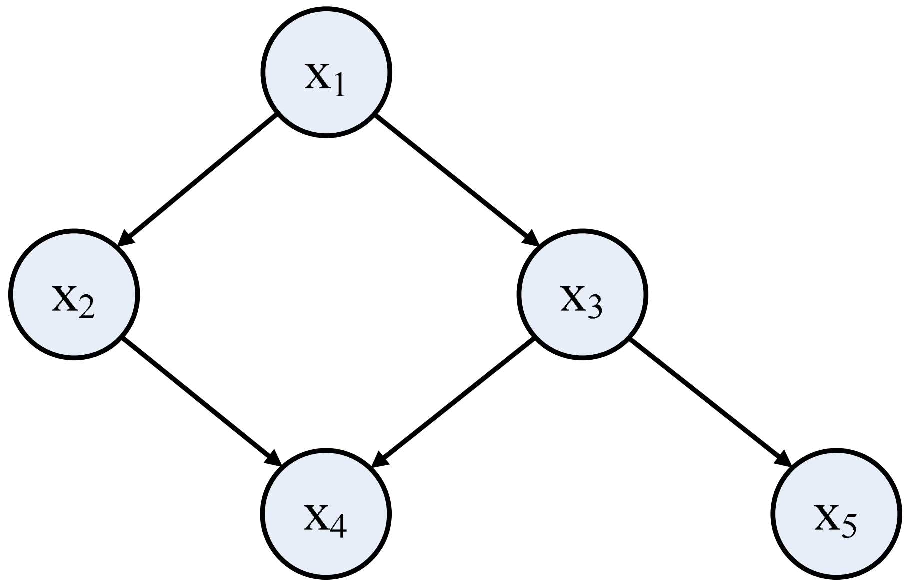
- Let
  - $x_i$ be a node in a Bayesian network $G$
  - $\Gamma_{x_i}$ be the set of parent nodes of $x_i$
- The influence of $\Gamma_{x_i}$ on $x_i$ can be specified by any set of functions $F_i(x_i, \Gamma_{x_i})$ that satisfy

$$\begin{aligned} \sum_{\forall x_i} F_i(x_i, \Gamma_{x_i}) &= 1 \\ 0 \leq F_i(x_i, \Gamma_{x_i}) &\leq 1 \end{aligned}$$

where $x_i$ also refers to the states of the random variable associated to the node $x_i$

# Bayesian Networks

- A Bayesian network for a joint probability distribution $P(x_1, x_2, x_3, x_4, x_5)$

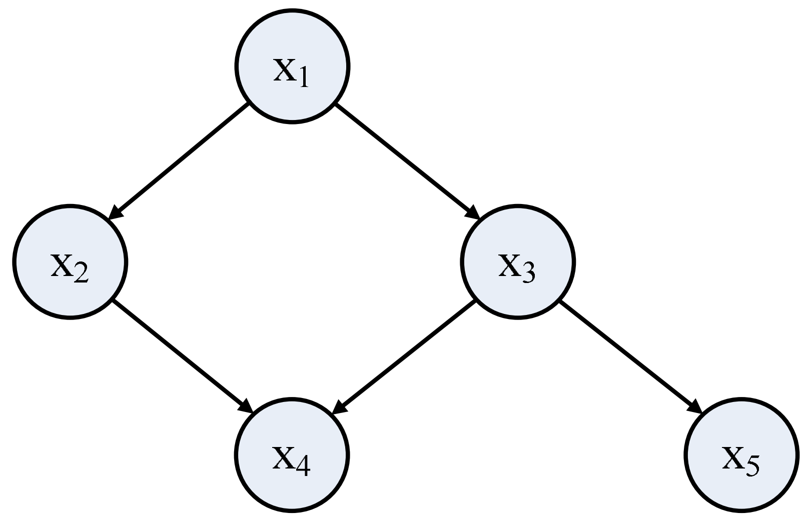


# Bayesian Networks

- The dependencies declared in the network allow the natural expression of the joint probability distribution

$$P(x_1, x_2, x_3, x_4, x_5) = P(x_1)P(x_2|x_1)P(x_3|x_1)P(x_4|x_2, x_3)P(x_5|x_3)$$

- The probability $P(x_1)$ is called the **prior** probability for the network
- It can be used to model previous knowledge about the semantics of the application



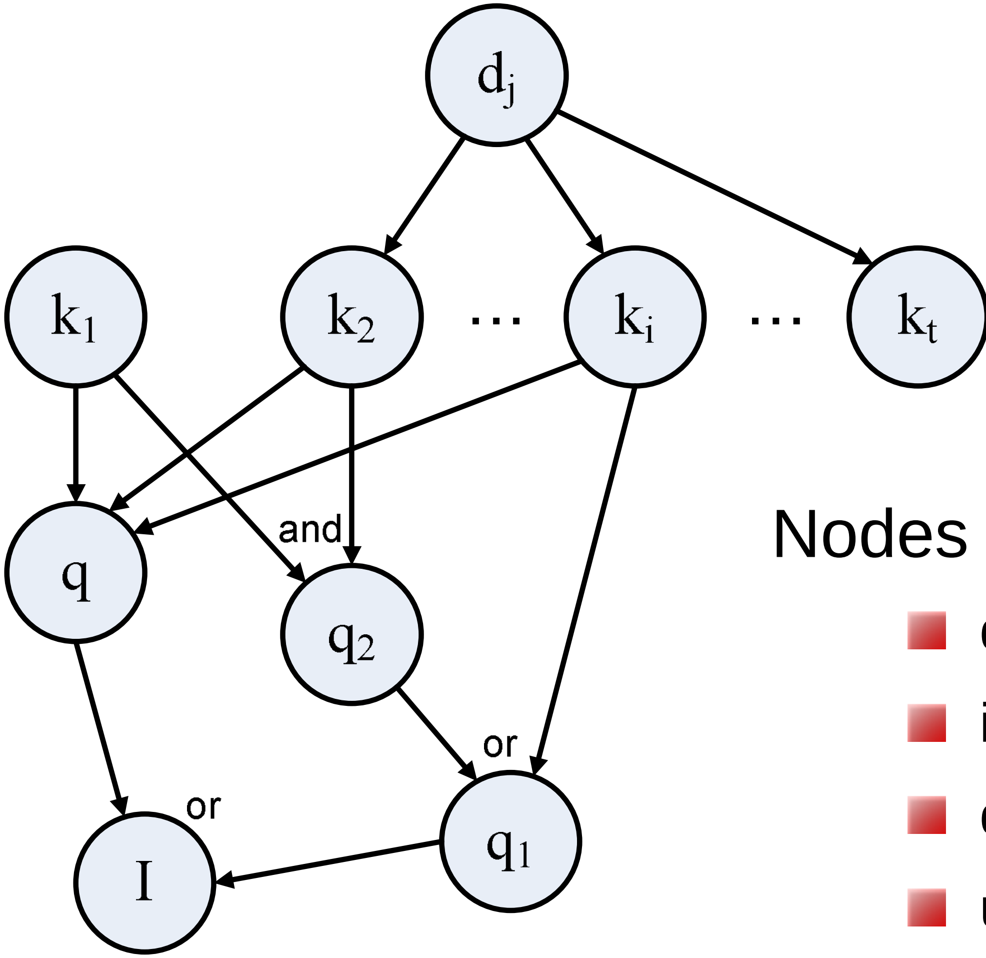
# Inference Network Model

# Inference Network Model

- An **epistemological** view of the information retrieval problem
- Random variables associated with documents, index terms and queries
- A random variable associated with a document $d_j$ represents the event of observing that document
- The observation of $d_j$ asserts a belief upon the random variables associated with its index terms

# Inference Network Model

- An **inference network** for information retrieval

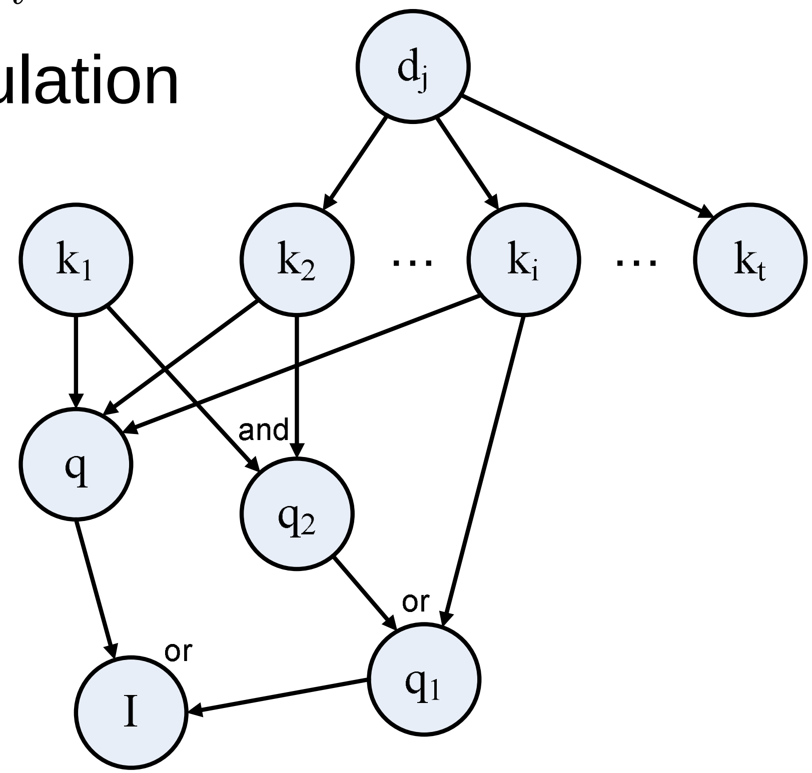



Nodes of the network

- documents $(d_j)$
- index terms $(k_i)$
- queries $(q, q_1, and\ q_2)$
- user information need $(I)$

# Inference Network Model

- The edges from $d_j$ to the nodes $k_i$ indicate that the observation of $d_j$ increase the belief in the variables $k_i$
- $d_j$ has index terms $k_2$, $k_i$, and $k_t$
- $q$ has index terms $k_1$, $k_2$, and $k_i$
- $q_1$ and $q_2$ model boolean formulation
- $q_1 = (k_1 \wedge k_2) \vee k_i)$
- $I = (q \vee q_1)$



# Inference Network Model

- Let $\vec{k} = (k_1, k_2, \ldots, k_t)$ a t-dimensional vector
- $k_i \in \{0, 1\}$, then $k$ has $2^t$ possible states
- Define

$$on(i, \vec{k}) = \begin{cases} 1 & \text{if } k_i = 1 \text{ according to } \vec{k} \\ 0 & \text{otherwise} \end{cases}$$

- Let $d_j \in \{0, 1\}$ and $q \in \{0, 1\}$
- The ranking of $d_j$ is a measure of how much evidential support the observation of $d_j$ provides to the query

# Inference Network Model

- The ranking is computed as $P(q \wedge d_j)$ where $q$ and $d_j$ are short representations for $q = 1$ and $d_j = 1$, respectively
- $d_j$ stands for a state where $d_j = 1$ and $\forall_{l \neq j} \Rightarrow d_l = 0$, because we observe one document at a time

$$
\begin{aligned}
P(q \wedge d_j) &= \sum_{\forall \vec{k}} P(q \wedge d_j | \vec{k}) \times P(\vec{k}) \\
&= \sum_{\forall \vec{k}} P(q \wedge d_j \wedge \vec{k}) \\
&= \sum_{\forall \vec{k}} P(q | d_j \wedge \vec{k}) \times P(d_j \wedge \vec{k}) \\
&= \sum_{\forall \vec{k}} P(q | \vec{k}) \times P(\vec{k} | d_j) \times P(d_j) \\
P(\overline{q \wedge d_j}) &= 1 - P(q \wedge d_j)
\end{aligned}
$$

# Inference Network Model

- The observation of $d_j$ separates its children index term nodes making them mutually independent
- This implies that $P(\vec{k}|d_j)$ can be computed in product form which yields

$$
\begin{aligned}
P(q \wedge d_j) \quad = \quad & \sum_{\forall \vec{k}} P(q|\vec{k}) \times P(d_j) \quad \times \\
& \left( \prod_{\forall i | on(i,\vec{k})=1} P(k_i|d_j) \quad \times \prod_{\forall i | on(i,\vec{k})=0} P(\overline{k}_i|d_j) \right)
\end{aligned}
$$

where $P(\overline{k}_i|d_j) = 1 - P(k_i|d_j)$

# Prior Probabilities

- The **prior probability** $P(d_j)$ reflects the probability of observing a given document $d_j$
- In Turtle and Croft this probability is set to $1/N$, where $N$ is the total number of documents in the system:

$$P(d_j) = \frac{1}{N} \qquad P(\overline{d}_j) = 1 - \frac{1}{N}$$

- To include document length normalization in the model, we could also write $P(d_j)$ as follows:

$$P(d_j) = \frac{1}{|\vec{d_j}|} \qquad P(\overline{d}_j) = 1 - P(d_j)$$

where $|\vec{d_j}|$ stands for the norm of the vector $\vec{d_j}$

# Network for Boolean Model

- How an inference network can be tuned to subsume the Boolean model?
- First, for the Boolean model, the prior probabilities are given by:

$$P(d_j) = \frac{1}{N} \qquad P(\overline{d}_j) = 1 - \frac{1}{N}$$

- Regarding the conditional probabilities $P(k_i|d_j)$ and $P(q|\vec{k})$, the specification is as follows

$$\begin{aligned} P(k_i|d_j) &= \begin{cases} 1 & \text{if } k_i \in d_j \\ 0 & \text{otherwise} \end{cases} \\ P(\overline{k}_i|d_j) &= 1 - P(k_i|d_j) \end{aligned}$$

# Network for Boolean Model

- We can use $P(k_i|d_j)$ and $P(q|\vec{k})$ factors to compute the evidential support the index terms provide to $q$:

$$P(q|\vec{k}) = \begin{cases} 1 & \text{if } c(q) = c(\vec{k}) \\ 0 & \text{otherwise} \end{cases}$$

$$P(\overline{q}|\vec{k}) = 1 - P(q|\vec{k})$$

  where $c(q)$ and $c(\vec{k})$ are the conjunctive components associated with $q$ and $\vec{k}$, respectively

- By using these definitions in $P(q \wedge d_j)$ and $P(\overline{q \wedge d_j})$ equations, we obtain the Boolean form of retrieval

# Network for TF-IDF Strategies

- For a tf-idf ranking strategy
- Prior probability $P(d_j)$ reflects the importance of **document normalization**

$$P(d_j) = \frac{1}{|\vec{d_j}|} \qquad P(\overline{d}_j) = 1 - P(d_j)$$

# Network for TF-IDF Strategies

- For the document-term beliefs, we write:

$$
\begin{aligned}
P(k_i|d_j) &= \alpha + (1-\alpha) \times \overline{f}_{i,j} \times \overline{idf}_i \\
P(\overline{k}_i|d_j) &= 1 - P(k_i|d_j)
\end{aligned}
$$

  where $\alpha$ varies from $0$ to $1$, and empirical evidence suggests that $\alpha = 0.4$ is a good default value

- Normalized **term frequency** and **inverse document frequency**:

$$
\overline{f}_{i,j} = \frac{f_{i,j}}{max_i f_{i,j}} \qquad \overline{idf}_i = \frac{\log \frac{N}{n_i}}{\log N}
$$

# Network for TF-IDF Strategies

- For the term-query beliefs, we write:

$$
\begin{aligned}
P(q|\vec{k}) &= \sum_{k_i \in q} \overline{f}_{i,j} \times w_q \\
P(\overline{q}|\vec{k}) &= 1 - P(q|\vec{k})
\end{aligned}
$$

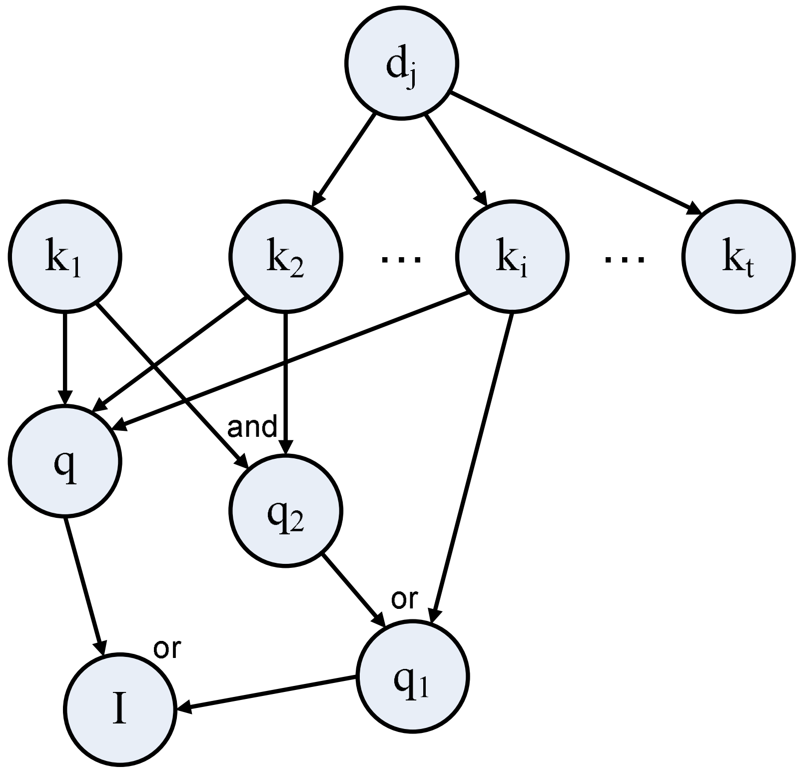
where $w_q$ is a parameter used to set the maximum belief achievable at the query node

# Network for TF-IDF Strategies

- By substituting these definitions into $P(q \wedge d_j)$ and $P(\overline{q \wedge d_j})$ equations, we obtain a tf-idf form of ranking
- We notice that the ranking computed by the inference network is distinct from that for the vector model
- However, an inference network is able to provide good retrieval performance with general collections

# Combining Evidential Sources

- In Figure below, the node $q$ is the standard keyword-based query formulation for $I$
- The second query $q_1$ is a Boolean-like query formulation for the same information need



# Combining Evidential Sources

- Let $I = q \vee q_1$
- In this case, the ranking provided by the inference network is computed as

$$
\begin{aligned}
P(I \wedge d_j) &= \sum_{\vec{k}} P(I|\vec{k}) \times P(\vec{k}|d_j) \times P(d_j) \\
&= \sum_{\vec{k}} (1 - P(\overline{q}|\vec{k})\ P(\overline{q}_1|\vec{k})) \times P(\vec{k}|d_j) \times P(d_j)
\end{aligned}
$$

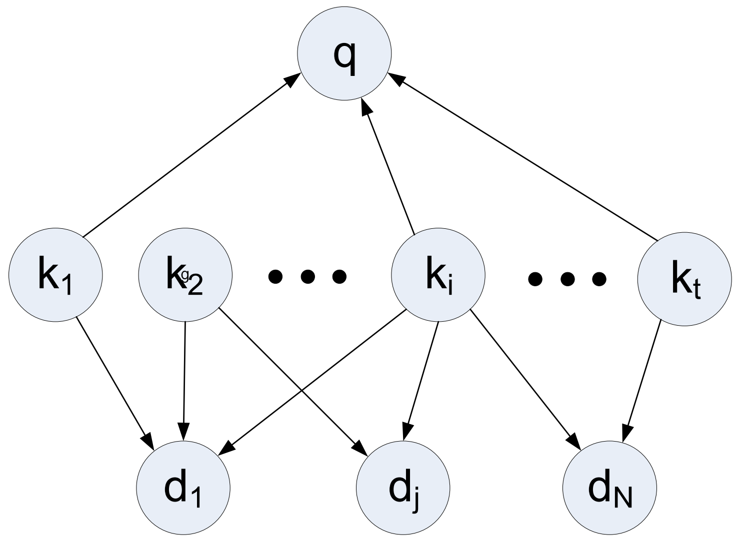
which might yield a retrieval performance which surpasses that of the query nodes in isolation (Turtle and Croft)

# Belief Network Model

# Belief Network Model

- The belief network model is a variant of the inference network model with a slightly different network topology
- As the Inference Network Model
  - Epistemological view of the IR problem
  - Random variables associated with documents, index terms and queries
- Contrary to the Inference Network Model
  - Clearly defined sample space
  - Set-theoretic view

# Belief Network Model




$P(d_j \mid q)$: rank of $d_j$ with regard to q

By applying Bayes' rule, we can write

$$P(d_j|q) = P(d_j \wedge q)/P(q)$$

$$P(d_j|q) \sim \sum_{\forall \vec{k}} P(d_j \wedge q|\vec{k}) \times P(\vec{k})$$

because $P(q)$ is a constant for all documents in the collection

# Belief Network Model

- Instantiation of the index term variables separates the nodes $q$ and $d$ making them mutually independent:

$$P(d_j|q) \sim \sum_{\forall \vec{k}} P(d_j|\vec{k}) \times P(q|\vec{k}) \times P(\vec{k})$$

- To complete the belief network we need to specify the conditional probabilities $P(q|\vec{k})$ and $P(d_j|\vec{k})$
- Distinct specifications of these probabilities allow the modeling of different ranking strategies

# Belief Network Model

- For the vector model, for instance, we define a vector $\vec{k}_i$ given by

$$\vec{k}_i = \vec{k} \mid on(i, \vec{k}) = 1 \land \forall_{j \neq i}\ on(i, \vec{k}) = 0$$

- The motivation is that tf-idf ranking strategies sum up the individual contributions of index terms
- We proceed as follows

$$P(q|\vec{k}) = \begin{cases} \frac{w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,q}^2}} & \text{if } \vec{k} = \vec{k}_i \ \land \ on(i, \vec{q}) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$P(\overline{q}|\vec{k}) = 1 - P(q|\vec{k})$$

# Belief Network Model

- Further, define

$$P(d_j|\vec{k}) = \begin{cases} \frac{w_{i,j}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2}} & \text{if } \vec{k} = \vec{k_i} \ \wedge \ on(i, \vec{d_j}) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$P(\overline{d_j}|\vec{k}) = 1 - P(d_j|\vec{k})$$

- Then, the ranking of the retrieved documents coincides with the ranking ordering generated by the vector model

# Computational Costs

- In the inference network model only the states which have a single document active node are considered
- Thus, the cost of computing the ranking is linear on the number of documents in the collection
- However, the ranking computation is restricted to the documents which have terms in common with the query
- The networks do not impose additional costs because the networks do not include cycles

# Other Models

- Hypertext Model
- Web-based Models
- Structured Text Retrieval
- Multimedia Retrieval
- Enterprise and Vertical Search

# Hypertext Model

# The Hypertext Model

- **Hypertexts** provided the basis for the design of the hypertext markup language (HTML)
- Written text is usually conceived to be read **sequentially**
- Sometimes, however, we are looking for information that cannot be easily captured through sequential reading
  - For instance, while glancing at a book about the history of the wars, we might be interested in wars in Europe
- In such a situation, a different organization of the text is desired

# The Hypertext Model

- The solution is to define a new organizational structure besides the one already in existence
- One way to accomplish this is through **hypertexts**, that are high level interactive navigational structures
- A hypertext consists basically of nodes that are correlated by directed links in a graph structure

# The Hypertext Model

- Two nodes $A$ and $B$ might be connected by a **directed link** $l_{AB}$ which correlates the texts of these two nodes
- In this case, the reader might move to the node $B$ while reading the text associated with node $A$
- When the hypertext is large, the user might lose track of the organizational structure of the hypertext
- To avoid this problem, it is desirable that the hypertext include a hypertext **map**
    - In its simplest form, this map is a directed graph which displays the current node being visited

# The Hypertext Model

- Definition of the structure of the hypertext should be accomplished in a domain **modeling phase**
- After the modeling of the domain, a user **interface design** should be concluded prior to implementation
- Only then, can we say that we have a proper hypertext structure for the application at hand

# Web-based Models

# Web-based Models

- The first Web search engines were fundamentally IR engines based on the models we have discussed here
- The key differences were:
  - the collections were composed of Web pages (not documents)
  - the pages had to be crawled
  - the collections were much larger
- This third difference also meant that each query word retrieved too many documents
- As a result, results produced by these engines were frequently dissatisfying

# Web-based Models

- A key piece of innovation was missing—the use of link information present in Web pages to modify the ranking
- There are two fundamental approaches to do this namely, PageRank and Hubs-Authorities
  - Such approaches are covered in Chapter 11 of the book (Web Retrieval)

# Structured Text Retrieval

# Structured Text Retrieval

- All the IR models discussed here treat the text as a string with no particular structure
- However, information on the structure might be important to the user for particular searches
  - Ex: retrieve a book that contains a figure of the Eiffel tower in a section whose title contains the term “France”
- The solution to this problem is to take advantage of the text structure of the documents to improve retrieval
- Structured text retrieval are discussed in Chapter 13 of the book

# Multimedia Retrieval

# Multimedia Retrieval

- Multimedia data, in the form of images, audio, and video, frequently lack text associated with them
- The retrieval strategies that have to be applied are quite distinct from text retrieval strategies
- However, multimedia data are an integral part of the Web
- Multimedia retrieval methods are discussed in great detail in Chapter 14 of the book

# Enterprise and Vertical Search

# Enterprise and Vertical Search

- Enterprise search is the task of searching for information of interest in corporate document collections
- Many issues not present in the Web, such as privacy, ownership, permissions, are important in enterprise search
- In Chapter 15 of the book we discuss in detail some enterprise search solutions

# Enterprise and Vertical Search

- A *vertical collection* is a repository of documents specialized in a given domain of knowledge
  - To illustrate, Lexis-Nexis offers full-text search focused on the area of business and in the area of legal
- Vertical collections present specific challenges with regard to search and retrieval