Chapter 3
Modeling

Part I: Classic Models
Introduction to IR Models
Basic Concepts
The Boolean Model
Term Weighting
The Vector Model
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.
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

- Information need
- Documents
- Index terms
  - Docs terms
  - Query terms
- Match
- Ranking
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.
An **IR model** is a quadruple \([D, Q, \mathcal{F}, R(q_i, d_j)]\) where

1. \(D\) is a set of logical views for the documents in the collection
2. \(Q\) 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

![Diagram](https://via.placeholder.com/150)
A Taxonomy of IR Models

- Classic IR Models (Unstructured Text)
  - Boolean
  - Vector
  - Probabilistic

- Set Theoretic
  - Fuzzy
  - Extended Boolean
  - Set-based

- Algebraic
  - Generalized Vector
  - Latent Semantic Indexing
  - Neural Networks

- Probabilistic
  - BM25
  - Language Models
  - Divergence from Randomness
  - Bayesian Networks

- Document Property
  - Text
  - Links
  - Multimedia

- Semi-Structured Text
  - Proximal Nodes, others
  - XML-based

- Web
  - Page Rank
  - Hubs & Authorities

- Multimedia Retrieval
  - Image Retrieval
  - Audio and Music Retrieval
  - Video Retrieval
Retrieval: Ad Hoc x Filtering

Ad Hoc Retrieval:

Collection
Fixed size

Q1
Q2
Q3
Q4
Retrieval: Ad Hoc x Filtering

Filtering

user 1
profile

user 2
profile

docs filtered for user 1

docs filtered for user 2

documents stream
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

\[
V = \begin{bmatrix} k_1 & k_2 & k_3 & \ldots & k_t \end{bmatrix}
\]

**vocabulary of** \( t \) **index terms**
Basic Concepts

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

\[ V = \begin{bmatrix} k_1 & k_2 & k_3 & \ldots & k_i \\ 1 & 0 & 0 & \ldots & 0 \\ \vdots \\ 1 & 1 & 1 & \ldots & 1 \end{bmatrix} \]

- pattern that represents documents (and queries) with the term \( k_i \) and no other
- pattern that represents documents (and queries) with all index terms

Each of these patterns of term co-occurrence 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 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 be written as

$$
\begin{pmatrix}
  d_1 & d_2 \\
  k_1 & \begin{bmatrix}
    f_{1,1} & f_{1,2} \\
    f_{2,1} & f_{2,2} \\
    f_{3,1} & f_{3,2}
  \end{bmatrix}
\end{pmatrix}
$$

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
    \[ q = k_a \land (k_b \lor \neg k_c) \]
- 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 \((k_i, q)\)
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** \( q = k_a \land (k_b \lor \neg k_c) \)
- **vocabulary** \( V = \{k_a, k_b, k_c\} \)

Then

\[
q_{DNF} = (1, 1, 1) \lor (1, 1, 0) \lor (1, 0, 0)
\]

\( c(q) \): a conjunctive component for \( q \).
The Boolean Model

The three conjunctive components for the query

\[ q = k_a \land (k_b \lor \neg k_c) \]
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 \( [q = k_a \land (k_b \lor \neg k_c)] \) is represented in disjunctive normal form as

\[ q_{DNF} = (1, 1, 1, 0) \lor (1, 1, 1, 1) \lor (1, 1, 0, 0) \lor (1, 1, 0, 1) \lor (1, 0, 0, 0) \lor (1, 0, 0, 1) \]
The Boolean Model

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

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

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.
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
- $w_{i,j} \geq 0$ 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 $d_j$.
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:

- $f(do, d_1) = 2$
- $f(do, d_2) = 0$
- $f(do, d_3) = 3$
- $f(do, d_4) = 3$
- $F(do) = 8$
- $n(do) = 3$

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

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

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

Do do do, da da da da.
Let it be, let it be.
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.
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 $c_{u,v} \in C$ 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

\[
\begin{bmatrix}
d_1 & d_2 \\
k_1 & \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \end{bmatrix} \\
k_2 & \\
k_3 & \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
k_1 & k_2 & k_3 \\
d_1 & \begin{bmatrix} w_{1,1} & w_{2,1} & w_{3,1} \\ w_{1,2} & w_{2,2} & w_{3,2} \end{bmatrix} \\
d_2 & \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
k_1 & k_2 & k_3 \\
k_1 & \begin{bmatrix} w_{1,1}w_{1,1} + w_{1,2}w_{1,2} & w_{1,1}w_{2,1} + w_{1,2}w_{2,2} & w_{1,1}w_{3,1} + w_{1,2}w_{3,2} \\ w_{2,1}w_{1,1} + w_{2,2}w_{1,2} & w_{2,1}w_{2,1} + w_{2,2}w_{2,2} & w_{2,1}w_{3,1} + w_{2,2}w_{3,2} \\ w_{3,1}w_{1,1} + w_{3,2}w_{1,2} & w_{3,1}w_{2,1} + w_{3,2}w_{2,2} & w_{3,1}w_{3,1} + w_{3,2}w_{3,2} \end{bmatrix} \\
k_2 & \\
k_3 & \\
\end{bmatrix}
\]
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 Frequency (TF) Weights**

- **Luhn Assumption.** The value of $w_{i,j}$ is proportional to the term frequency $f_{i,j}$
  
  That is, the more often a term occurs in the text of the document, the higher its weight

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

- Which leads directly to the following $tf$ weight formulation:

  $$tf_{i,j} = f_{i,j}$$
A variant of $tf$ weight used in the literature is

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

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

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

$d_1$: To do is to be. To be is to do.

$d_2$: To be or not to be. I am what I am.

$d_3$: I think therefore I am. Do be do be do.

$d_4$: Do do do, da da da. Let it be, let it be.
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
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

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

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

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

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

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

<table>
<thead>
<tr>
<th>term</th>
<th>$n_i$</th>
<th>$idf_i = \log(N/n_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>to</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>do</td>
<td>3</td>
<td>0.415</td>
</tr>
<tr>
<td>is</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>be</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>or</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>not</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>I</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>am</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>what</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>think</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>therefore</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>da</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>let</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>it</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
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

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

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
<th>$d_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>to</td>
<td>3</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>do</td>
<td>0.830</td>
<td>-</td>
<td>1.073</td>
<td>1.073</td>
</tr>
<tr>
<td>3</td>
<td>is</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>be</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>or</td>
<td>-</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>not</td>
<td>-</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>I</td>
<td>-</td>
<td>2</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>am</td>
<td>-</td>
<td>2</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>what</td>
<td>-</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>think</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>therefore</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>5.170</td>
</tr>
<tr>
<td>12</td>
<td>da</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>let</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>14</td>
<td>it</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4</td>
</tr>
</tbody>
</table>

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

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

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

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

<table>
<thead>
<tr>
<th></th>
<th>tf weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary</td>
<td>{0,1}</td>
</tr>
<tr>
<td>raw frequency</td>
<td>(f_{i,j})</td>
</tr>
<tr>
<td>log normalization</td>
<td>(1 + \log f_{i,j})</td>
</tr>
<tr>
<td>double normalization 0.5</td>
<td>(0.5 + 0.5 \frac{f_{i,j}}{\max_i f_{i,j}})</td>
</tr>
<tr>
<td>double normalization K</td>
<td>(K + (1 - K) \frac{f_{i,j}}{\max_i f_{i,j}})</td>
</tr>
</tbody>
</table>
Variants of TF-IDF

Five distinct variants of idf weight

<table>
<thead>
<tr>
<th>Variant</th>
<th>idf weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>unary</td>
<td>1</td>
</tr>
<tr>
<td>inverse frequency</td>
<td>$\log \frac{N}{n_i}$</td>
</tr>
<tr>
<td>inv frequency smooth</td>
<td>$\log(1 + \frac{N}{n_i})$</td>
</tr>
<tr>
<td>inv frequency max</td>
<td>$\log(1 + \frac{\max_i n_i}{n_i})$</td>
</tr>
<tr>
<td>probabilistic inv frequency</td>
<td>$\log \frac{N - n_i}{n_i}$</td>
</tr>
</tbody>
</table>
## Variants of TF-IDF

### Recommended tf-idf weighting schemes

<table>
<thead>
<tr>
<th>weighting scheme</th>
<th>document term weight</th>
<th>query term weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$f_{i,j} \times \log \frac{N}{n_i}$</td>
<td>$(0.5 + 0.5 \frac{f_{i,q}}{\max_i f_{i,q}}) \times \log \frac{N}{n_i}$</td>
</tr>
<tr>
<td>2</td>
<td>$1 + \log f_{i,j}$</td>
<td>$\log(1 + \frac{N}{n_i})$</td>
</tr>
<tr>
<td>3</td>
<td>$(1 + \log f_{i,j}) \times \log \frac{N}{n_i}$</td>
<td>$(1 + \log f_{i,q}) \times \log \frac{N}{n_i}$</td>
</tr>
</tbody>
</table>
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:

\begin{align*}
    tf_i &= 1 + \log \sum_{j=1}^{N} f_{i,j} \\
    idf_i &= \log \frac{N}{n_i}
\end{align*}
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 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**
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 $w_{i,j} \times \vec{k}_i$. 

The document representation $\vec{d}_j$ is a vector composed of all its term vector components:

$$\vec{d}_j = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$$

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

<table>
<thead>
<tr>
<th>d₁</th>
<th>d₂</th>
<th>d₃</th>
<th>d₄</th>
</tr>
</thead>
</table>
| To do is to be.  
To be is to do. | To be or not to be.  
I am what I am. | I think therefore I am.  
Do be do do be do. | Do do do, da da da.  
Let it be, let it be. |
| 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
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 $t$-dimensional 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 $q$

\[
\cos(\theta) = \frac{\vec{d}_j \cdot \vec{q}}{|\vec{d}_j| \times |\vec{q}|}
\]

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

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

Weights in the Vector model are basically tf-idf weights

\[ w_{i,q} = (1 + \log f_{i,q}) \times \log \frac{N}{n_i} \]

\[ w_{i,j} = (1 + \log f_{i,j}) \times \log \frac{N}{n_i} \]

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)

<table>
<thead>
<tr>
<th>doc</th>
<th>rank computation</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_1$</td>
<td>$1\times 3 + 0.415 \times 0.830$</td>
<td>0.660</td>
</tr>
<tr>
<td>$d_2$</td>
<td>$1\times 2 + 0.415 \times 0$</td>
<td>0.408</td>
</tr>
<tr>
<td>$d_3$</td>
<td>$1\times 0 + 0.415 \times 1.073$</td>
<td>0.118</td>
</tr>
<tr>
<td>$d_4$</td>
<td>$1\times 0 + 0.415 \times 1.073$</td>
<td>0.058</td>
</tr>
</tbody>
</table>
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?
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,

\[
sim(d_j, q) = \frac{P(\tilde{d}_j| R, q) \times P(R, q)}{P(\tilde{d}_j| \overline{R}, q) \times P(\overline{R}, q)} \sim \frac{P(\tilde{d}_j| R, q)}{P(\tilde{d}_j| \overline{R}, q)}
\]

where

- \(P(\tilde{d}_j| R, q)\) : 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 \(q\)
- \(P(\tilde{d}_j| \overline{R}, q)\) 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:

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

where

- $P(k_i \mid R, q)$: probability that the term $k_i$ is present in a document randomly selected from the set $R$
- $P(\overline{k}_i \mid R, q)$: 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
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:

\[ \text{sim}(d_j, q) \sim \frac{(\prod_{k_i | w_{i,j} = 1} p_{iR}) \times (\prod_{k_i | w_{i,j} = 0} (1 - p_{iR}))}{(\prod_{k_i | w_{i,j} = 1} q_{iR}) \times (\prod_{k_i | w_{i,j} = 0} (1 - q_{iR}))} \]
Taking logarithms, we write

\[
sim(d_j, q) \sim \log \prod_{k_i | w_{i,j} = 1} p_{iR} + \log \prod_{k_i | w_{i,j} = 0} (1 - p_{iR})
- \log \prod_{k_i | w_{i,j} = 1} q_{iR} - \log \prod_{k_i | w_{i,j} = 0} (1 - q_{iR})
\]
The Ranking

Summing up terms that cancel each other, we obtain

\[
sim(d_j, q) \sim \log \prod_{k_i \mid w_{i,j} = 1} p_{iR} + \log \prod_{k_i \mid w_{i,j} = 0} (1 - p_{iR}) - \log \prod_{k_i \mid w_{i,j} = 1} (1 - p_{iR}) + \log \prod_{k_i \mid w_{i,j} = 1} (1 - q_{iR}) - \log \prod_{k_i \mid w_{i,j} = 0} (1 - q_{iR}) + \log \prod_{k_i \mid w_{i,j} = 1} (1 - q_{iR}) - \log \prod_{k_i \mid w_{i,j} = 1} (1 - q_{iR})
\]
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.
Further, assuming that
\[ \forall k_i \notin q, \ p_{iR} = q_{iR} \]
and converting the log products into sums of logs, we finally obtain

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

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

<table>
<thead>
<tr>
<th>docs that contain ( k_i )</th>
<th>relevant</th>
<th>non-relevant</th>
<th>all docs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_i )</td>
<td>( n_i - r_i )</td>
<td>( n_i )</td>
<td></td>
</tr>
<tr>
<td>docs that do not contain ( k_i )</td>
<td>( R - r_i )</td>
<td>( N - n_i - (R - r_i) )</td>
<td>( N - n_i )</td>
</tr>
<tr>
<td>all docs</td>
<td>( R )</td>
<td>( N - R )</td>
<td>( N )</td>
</tr>
</tbody>
</table>
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 \( k_i \in q \land k_i \in d_j \)
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 $\text{sim}(d_j, q)$ 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.
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 bootstrap the ranking equation, which leads to

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

This equation provides an idf-like ranking computation

In the absence of relevance information, this is the equation for ranking in the probabilistic model
Document ranks computed by the previous probabilistic ranking equation for the query “to do”

<table>
<thead>
<tr>
<th>doc</th>
<th>rank computation</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{d}_1</td>
<td>\log \frac{4-2+0.5}{2+0.5} + \log \frac{4-3+0.5}{3+0.5}</td>
<td>-1.222</td>
</tr>
<tr>
<td>\textbf{d}_2</td>
<td>\log \frac{4-2+0.5}{2+0.5}</td>
<td>0</td>
</tr>
<tr>
<td>\textbf{d}_3</td>
<td>\log \frac{4-3+0.5}{3+0.5}</td>
<td>-1.222</td>
</tr>
<tr>
<td>\textbf{d}_4</td>
<td>\log \frac{4-3+0.5}{3+0.5}</td>
<td>-1.222</td>
</tr>
</tbody>
</table>
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.
Using this latest formulation, we redo the ranking computation for our example collection for the query “to do” and obtain

<table>
<thead>
<tr>
<th>doc</th>
<th>rank computation</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d_1)</td>
<td>(\log \frac{4+0.5}{2+0.5} + \log \frac{4+0.5}{3+0.5})</td>
<td>1.210</td>
</tr>
<tr>
<td>(d_2)</td>
<td>(\log \frac{4+0.5}{2+0.5})</td>
<td>0.847</td>
</tr>
<tr>
<td>(d_3)</td>
<td>(\log \frac{4+0.5}{3+0.5})</td>
<td>0.362</td>
</tr>
<tr>
<td>(d_4)</td>
<td>(\log \frac{4+0.5}{3+0.5})</td>
<td>0.362</td>
</tr>
</tbody>
</table>
Estimating $r_i$ 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

\[ \text{sim}(d_j, q) \sim \sum_{k_i \in q \land k_i \in d_j} \log \left( \frac{p_i R}{1 - p_i R} \right) + \log \left( \frac{1 - q_i R}{q_i R} \right) \]

How obtain the probabilities \( p_i R \) and \( q_i R \) ?

Estimates based on assumptions:

- \( p_i R = 0.5 \)
- \( q_i R = \frac{n_i}{N} \) 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:

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

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.
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

\[ \text{sim}(d_j, q) \sim \sum_{k_i \in q \land k_i \in d_j} \log \left( \frac{N - n_i}{n_i} \right) \]

To avoid problems with \( D = 1 \) and \( D_i = 0 \):

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

Also,

\[ p_{iR} = \frac{D_i + \frac{n_i}{N}}{D + 1}; \quad q_{iR} = \frac{n_i - D_i + \frac{n_i}{N}}{N - D + 1} \]
Pluses and Minuses

Advantages:
- Docs ranked in decreasing order of probability of relevance

Disadvantages:
- Need to guess initial estimates for $p_i R$
- Method does not take into account $t f$ 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.
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 $V_S = \{S_1, S_2, \ldots, S_{2t}\}$ is the **vocabulary-set** of the collection

To illustrate, consider the document collection below

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

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

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

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

To simplify notation, let us define

\[ k_a = \text{to} \quad k_d = \text{be} \quad k_g = \text{l} \quad k_j = \text{think} \quad k_m = \text{let} \]

\[ k_b = \text{do} \quad k_e = \text{or} \quad k_h = \text{am} \quad k_k = \text{therefore} \quad k_n = \text{it} \]

\[ k_c = \text{is} \quad k_f = \text{not} \quad k_i = \text{what} \quad k_l = \text{da} \]

Further, let the letters \( a \ldots n \) refer to the index terms \( k_a \ldots k_n \), respectively.
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

<table>
<thead>
<tr>
<th>Termset</th>
<th>Set of Terms</th>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_a$</td>
<td>${a}$</td>
<td>${d_1, d_2}$</td>
</tr>
<tr>
<td>$S_b$</td>
<td>${b}$</td>
<td>${d_1, d_3, d_4}$</td>
</tr>
<tr>
<td>$S_d$</td>
<td>${d}$</td>
<td>${d_1, d_2, d_3, d_4}$</td>
</tr>
<tr>
<td>$S_n$</td>
<td>${n}$</td>
<td>${d_4}$</td>
</tr>
<tr>
<td>$S_{ab}$</td>
<td>${a, b}$</td>
<td>${d_1}$</td>
</tr>
<tr>
<td>$S_{ad}$</td>
<td>${a, d}$</td>
<td>${d_1, d_2}$</td>
</tr>
<tr>
<td>$S_{bd}$</td>
<td>${b, d}$</td>
<td>${d_1, d_3, d_4}$</td>
</tr>
<tr>
<td>$S_{bn}$</td>
<td>${b, n}$</td>
<td>${d_4}$</td>
</tr>
<tr>
<td>$S_{abd}$</td>
<td>${a, b, d}$</td>
<td>${d_1}$</td>
</tr>
<tr>
<td>$S_{bdn}$</td>
<td>${b, d, n}$</td>
<td>${d_4}$</td>
</tr>
</tbody>
</table>

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 $N_i$ be the number of documents in which $S_i$ occurs.

An $n$-termset $S_i$ is said to be **frequent** if $N_i$ 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
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.
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 $q$
- $N_i$ be the number of documents in which termset $S_i$ occurs
- $N$ be the total number of documents in the collection
- $F_{i,j}$ be the frequency of termset $S_i$ in document $d_j$

For each pair $[S_i, d_j]$ we compute a weight $W_{i,j}$ given by

$$W_{i,j} = \begin{cases} 
(1 + \log F_{i,j}) \log\left(1 + \frac{N}{N_i}\right) & \text{if } F_{i,j} > 0 \\
0 & \text{if } F_{i,j} = 0 
\end{cases}$$

We also compute a $W_{i,q}$ value for each pair $[S_i, q]$. 
### Ranking Computation

Consider query $q = \{a, b, d, n\}$

**Document $d_1 = \text{``a b c a d a d c a b''}$$''**

<table>
<thead>
<tr>
<th>Termset</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_a$</td>
<td>$\mathcal{W}_{a,1} = (1 + \log 4) \times \log(1 + 4/2) = 4.75$</td>
</tr>
<tr>
<td>$S_b$</td>
<td>$\mathcal{W}_{b,1} = (1 + \log 2) \times \log(1 + 4/3) = 2.44$</td>
</tr>
<tr>
<td>$S_d$</td>
<td>$\mathcal{W}_{d,1} = (1 + \log 2) \times \log(1 + 4/4) = 2.00$</td>
</tr>
<tr>
<td>$S_n$</td>
<td>$\mathcal{W}_{n,1} = 0 \times \log(1 + 4/1) = 0.00$</td>
</tr>
<tr>
<td>$S_{ab}$</td>
<td>$\mathcal{W}_{ab,1} = (1 + \log 2) \times \log(1 + 4/1) = 4.64$</td>
</tr>
<tr>
<td>$S_{ad}$</td>
<td>$\mathcal{W}_{ad,1} = (1 + \log 2) \times \log(1 + 4/2) = 3.17$</td>
</tr>
<tr>
<td>$S_{bd}$</td>
<td>$\mathcal{W}_{bd,1} = (1 + \log 2) \times \log(1 + 4/3) = 2.44$</td>
</tr>
<tr>
<td>$S_{dn}$</td>
<td>$\mathcal{W}_{dn,1} = 0 \times \log(1 + 4/1) = 0.00$</td>
</tr>
<tr>
<td>$S_{abd}$</td>
<td>$\mathcal{W}_{abd,1} = (1 + \log 2) \times \log(1 + 4/1) = 4.64$</td>
</tr>
<tr>
<td>$S_{bdn}$</td>
<td>$\mathcal{W}_{bdn,1} = 0 \times \log(1 + 4/1) = 0.00$</td>
</tr>
</tbody>
</table>
A document $d_j$ and a query $q$ are represented as vectors in a $2^t$-dimensional space of termsets

$$\vec{d}_j = (\mathcal{W}_{1,j}, \mathcal{W}_{2,j}, \ldots, \mathcal{W}_{2^t,j})$$

$$\vec{q} = (\mathcal{W}_{1,q}, \mathcal{W}_{2,q}, \ldots, \mathcal{W}_{2^t,q})$$

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

$$\text{sim}(d_j, q) = \frac{\vec{d}_j \cdot \vec{q}}{|\vec{d}_j| \times |\vec{q}|} = \frac{\sum_{S_i} W_{i,j} \times W_{i,q}}{|\vec{d}_j| \times |\vec{q}|}$$

For termsets that are not in the query $q$, $W_{i,q} = 0$
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

$$
|\vec{d}_1| = \sqrt{\mathcal{W}_{a,1}^2 + \mathcal{W}_{b,1}^2 + \mathcal{W}_{c,1}^2 + \mathcal{W}_{d,1}^2}
$$

$$
= \sqrt{4.75^2 + 2.44^2 + 4.64^2 + 2.00^2}
$$

$$
= 7.35
$$
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

\[
\text{sim}(d_1, q) = \frac{(W_{a,1} \cdot W_{a,q} + W_{b,1} \cdot W_{b,q} + W_{d,1} \cdot W_{d,q} + \\
W_{ab,1} \cdot W_{ab,q} + W_{ad,1} \cdot W_{ad,q} + W_{bd,1} \cdot W_{bd,q} + \\
W_{abd,1} \cdot W_{abd,q}) / |\vec{d}_1|}{\sqrt{\sum_i W_i}}
\]

\[
= \frac{(4.75 \cdot 1.58 + 2.44 \cdot 1.22 + 2.00 \cdot 1.00 + \\
4.64 \cdot 2.32 + 3.17 \cdot 1.58 + 2.44 \cdot 1.22 + \\
4.64 \cdot 2.32)}{7.35}
\]

\[
= 5.71
\]
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

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

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

<table>
<thead>
<tr>
<th>frequency($S_i$)</th>
<th>frequent termset</th>
<th>closed termset</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>3</td>
<td>b, bd</td>
<td>bd</td>
</tr>
<tr>
<td>2</td>
<td>a, ad</td>
<td>ad</td>
</tr>
<tr>
<td>2</td>
<td>g, h, gh, ghd</td>
<td>ghd</td>
</tr>
</tbody>
</table>
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

$$
sim(d_1, q) = \left( \mathcal{W}_{d,1} \times \mathcal{W}_{d,q} + \mathcal{W}_{ab,1} \times \mathcal{W}_{ab,q} + \mathcal{W}_{ad,1} \times \mathcal{W}_{ad,q} + \mathcal{W}_{bd,1} \times \mathcal{W}_{bd,q} + \mathcal{W}_{abd,1} \times \mathcal{W}_{abd,q} \right) / |d_1|$$

$$
= \left( 2.00 \times 1.00 + 4.64 \times 2.32 + 3.17 \times 1.58 + 2.44 \times 1.22 + 4.64 \times 2.32 \right) / 7.35
$$

$$
= 4.28
$$
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

\[ q = k_x \land k_y \]

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 $w_{y,j}$.
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 $q_{or} = k_x \lor k_y$, 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 $q_{\text{and}} = k_x \land k_y$, 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.

\[
sim(q_{\text{and}}, d) = 1 - \sqrt{\frac{(1 - x)^2 + (1 - y)^2}{2}}
\]
The Idea

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

\[ \text{sim}(q_{and}, d) = 1 - \sqrt{\frac{(1 - x)^2 + (1 - y)^2}{2}} \]
Generalizing the Idea

We can extend the previous model to consider Euclidean distances in a \( t \)-dimensional space.

This can be done using \( p \)-\textit{norms} which extend the notion of distance to include \( p \)-distances, where \( 1 \leq p \leq \infty \).

A generalized conjunctive query is given by

\[
q_{\text{and}} = k_1 \land^p k_2 \land^p \ldots \land^p k_m
\]

A generalized disjunctive query is given by

\[
q_{\text{or}} = k_1 \lor^p k_2 \lor^p \ldots \lor^p k_m
\]
Generalizing the Idea

The query-document similarities are now given by

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

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

where each \( x_i \) stands for a weight \( w_{i,d} \)

- If \( p = 1 \) then (vector-like)
  \[ \text{sim}(q_{or}, d_j) = \text{sim}(q_{and}, d_j) = \frac{x_1 + \ldots + x_m}{m} \]

- If \( p = \infty \) then (Fuzzy like)
  \[ \text{sim}(q_{or}, d_j) = \max(x_i) \]
  \[ \text{sim}(q_{and}, d_j) = \min(x_i) \]
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 $q = (k_1 \land^p k_2) \lor^p k_3$.
  - $k_1$ 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

$$sim(q, d) = \left( \frac{1}{2} \cdot \left( \frac{1 - \left( \frac{(1-x_1)^p + (1-x_2)^p}{2} \right)}{2} \right)^p + x_3^p \right)^{\frac{1}{p}}$$
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:
  \[ q_1 = (k_1 \lor k_2) \land k_3 \]
  \[ q_2 = (k_1 \land k_3) \lor (k_2 \land k_3) \]
  \[ \text{sim}(q_1, d_j) \neq \text{sim}(q_2, d_j) \]
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 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
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,

\[
\mu_{\overline{A}}(u) = 1 - \mu_A(u)
\]
\[
\mu_{A \cup B}(u) = \max(\mu_A(u), \mu_B(u))
\]
\[
\mu_{A \cap B}(u) = \min(\mu_A(u), \mu_B(u))
\]
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 $k_\ell$.

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

where

- $n_i$: number of docs which contain $k_i$
- $n_\ell$: number of docs which contain $k_\ell$
- $n_{i,\ell}$: number of docs which contain both $k_i$ and $k_\ell$
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

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

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 $k_i$. 
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 $d_j$.
Consider the query \( q = k_a \land (k_b \lor \neg k_c) \)

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.
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,

\[
\begin{align*}
cc_1 &= \mu_{a,j} \mu_{b,j} \mu_{c,j} \\
cc_2 &= \mu_{a,j} \mu_{b,j} (1 - \mu_{c,j}) \\
cc_3 &= \mu_{a,j} (1 - \mu_{b,j}) (1 - \mu_{c,j})
\end{align*}
\]
Fuzzy IR: An Example

\[ D_q = cc_1 + cc_2 + cc_3 \]

\[ \mu_{q,j} = \mu_{cc_1 + cc_2 + cc_3,j} \]

\[ = 1 - \prod_{i=1}^{3} (1 - \mu_{cc_i,j}) \]

\[ = 1 - \left( 1 - \mu_{a,j} \mu_{b,j} \mu_{c,j} \right) \times \left( 1 - \mu_{a,j} \mu_{b,j} (1 - \mu_{c,j}) \right) \times \left( 1 - \mu_{a,j} (1 - \mu_{b,j})(1 - \mu_{c,j}) \right) \]
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 \( \forall i, j \Rightarrow \vec{k}_i \cdot \vec{k}_j = 0 \)
- In the generalized vector space model, two index term vectors might be non-orthogonal
As before, let $w_{i,j}$ be the weight associated with $[k_i, d_j]$ and $V = \{k_1, k_2, \ldots, k_t\}$ 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:

$$m_1 = (0, 0, 0, \ldots, 0)$$
$$m_2 = (1, 0, 0, \ldots, 0)$$
$$m_3 = (0, 1, 0, \ldots, 0)$$
$$m_4 = (1, 1, 0, \ldots, 0)$$
$$\vdots$$
$$m_{2^t} = (1, 1, 1, \ldots, 1)$$

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 $\vec{m}_r$,

$$\vec{m}_1 = (1, 0, \ldots, 0)$$

$$\vec{m}_2 = (0, 1, \ldots, 0)$$

$$\vdots$$

$$\vec{m}_{2^t} = (0, 0, \ldots, 1)$$

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

1, 2, \ldots, 2^t
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:

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

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

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:

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

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

<table>
<thead>
<tr>
<th></th>
<th>$K_1$</th>
<th>$K_2$</th>
<th>$K_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_1$</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$d_2$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$d_3$</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>$d_4$</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$d_5$</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$d_6$</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$d_7$</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>$q$</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Computation of $c_{i,r}$

<table>
<thead>
<tr>
<th></th>
<th>$K_1$</th>
<th>$K_2$</th>
<th>$K_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_1$</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$d_2$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$d_3$</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>$d_4$</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$d_5$</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>$d_6$</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$d_7$</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>$q$</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

| $d_1 = m_6$ | 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 |

| $m_1$   | 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 $\vec{k}_i$

\[
\vec{k}_1 = \frac{(3\vec{m}_2 + 2\vec{m}_6 + \vec{m}_8)}{\sqrt{3^2 + 2^2 + 1^2}}
\]

\[
\vec{k}_2 = \frac{(5\vec{m}_3 + 3\vec{m}_7 + 2\vec{m}_8)}{\sqrt{5 + 3 + 2}}
\]

\[
\vec{k}_3 = \frac{(1\vec{m}_6 + 5\vec{m}_7 + 4\vec{m}_8)}{\sqrt{1 + 5 + 4}}
\]

<table>
<thead>
<tr>
<th></th>
<th>$c_{1,r}$</th>
<th>$c_{2,r}$</th>
<th>$c_{3,r}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$m_2$</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$m_3$</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>$m_4$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$m_5$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$m_6$</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$m_7$</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>$m_8$</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
Computation of Document Vectors

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

<table>
<thead>
<tr>
<th></th>
<th>( K_1 )</th>
<th>( K_2 )</th>
<th>( K_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_1 )</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( d_2 )</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( d_3 )</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>( d_4 )</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( d_5 )</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>( d_6 )</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>( d_7 )</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>( q )</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
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
- \(M = [m_{ij}]\): term-document matrix \(t \times N\)

To each element of \(M\) 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 $M = [m_{ij}]$ can be decomposed into three components using singular value decomposition

$$M = K \cdot S \cdot D^T$$

where

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

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

<table>
<thead>
<tr>
<th></th>
<th>$K_1$</th>
<th>$K_2$</th>
<th>$K_3$</th>
<th>$q \cdot d_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_1$</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>$d_2$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$d_3$</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>$d_4$</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$d_5$</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>17</td>
</tr>
<tr>
<td>$d_6$</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>$d_7$</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>$q$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Compute the matrices $K$, $S$, and $D^t$
Latent Semantic Indexing

In the matrix $S$, consider that only the $s$ largest singular values are selected

Keep the corresponding columns in $K$ and $D^T$

The resultant matrix is called $M_s$ and is given by

$$M_s = K_s \cdot S_s \cdot D_s^T$$

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
The relationship between any two documents in $s$ can be obtained from the $M_s^T \cdot M_s$ matrix given by

\[
M_s^T \cdot M_s = (K_s \cdot S_s \cdot D_s^T)^T \cdot K_s \cdot S_s \cdot D_s^T = D_s \cdot S_s \cdot (K_s \cdot S_s)^T \\
= D_s \cdot S_s \cdot D_s^T
\]

In the above matrix, the $(i, j)$ element quantifies the relationship between documents $d_i$ and $d_j$. 
The user query can be modelled as a pseudo-document in the original $M$ matrix.

Assume the query is modelled as the document numbered $0$ in the $M$ matrix.

The matrix $M^T_s \cdot M_s$ 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.
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

query nodes

document term nodes (vocabulary)
document nodes

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

- Weight associated with an edge from a document term node $k_i$ to a document node $d_j$:
  
  $$
  \overline{w}_{i,j} = \frac{w_{i,j}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2}}
  $$
Quantifying Signal Propagation

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

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

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
Modern Information Retrieval

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

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

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

$$F_{i,j} = S_1 \times \frac{f_{i,j}}{K_1 + f_{i,j}}$$

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.
The next step was to modify the $F_{i,j}$ factor by adding document length normalization to it, as follows:

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

where

- $\text{len}(d_j)$ is the length of document $d_j$ (computed, for instance, as the number of terms in the document)
- $\text{avg\_doclen}$ is the average document length for the collection
Next, a correction factor $G_{j,q}$ dependent on the document and query lengths was added

$$G_{j,q} = K_2 \times \text{len}(q) \times \frac{\text{avg}_\text{doclen} - \text{len}(d_j)}{\text{avg}_\text{doclen} + \text{len}(d_j)}$$

where

- $\text{len}(q)$ is the query length (number of terms in the query)
- $K_2$ is a constant
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:

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

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

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

where \(k_i[q,d_j]\) is a short notation for \(k_i \in q \land k_i \in d_j\)
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 $F_{i,q}$ 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

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

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

\[ sim_{BM11}(d_j, q) \sim \sum_{k_i[q,d_j]} \frac{(K_1 + 1)f_{i,j}}{K_1 \frac{\text{len}(d_j)}{\text{avg\_doclen}} + f_{i,j}} \times \log \left( \frac{N - n_i + 0.5}{n_i + 0.5} \right) \]
BM25 Ranking Formula

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

\[ B_{i,j} = \frac{(K_1 + 1)f_{i,j}}{K_1 \left[ (1 - b) + b \frac{\text{len}(d_j)}{\text{avg}_\text{doclen}} \right] + f_{i,j}} \]

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

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

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 tuned for particular collections through proper experimentation
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
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
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
Ranking in a language model is provided by estimating $P(q|M_j)$.

Several researches 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:

$$P(q|M_j) = \prod_{k_i \in q} P(k_i|M_j)$$
By taking logs on both sides

\[
\log P(q|M_j) = \sum_{k_i \in q} \log P(k_i|M_j) \\
= \sum_{k_i \in q \land d_j} \log P_{\in}(k_i|M_j) + \sum_{k_i \in q \land \neg d_j} \log P_{\notin}(k_i|M_j) \\
= \sum_{k_i \in q \land d_j} \log \left( \frac{P_{\in}(k_i|M_j)}{P_{\notin}(k_i|M_j)} \right) + \sum_{k_i \in q} \log P_{\notin}(k_i|M_j)
\]

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.
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 $F_i = \sum_j f_{i,j}$
Thus, we obtain

\[
\log P(q|M_j) = \sum_{k_i \in q \land d_j} \log \left( \frac{P(\epsilon(k_i|M_j))}{\alpha_j P(k_i|\mathcal{C})} \right) + n_q \log \alpha_j + \sum_{k_i \in q} \log P(k_i|\mathcal{C})
\]

\[
\sim \sum_{k_i \in q \land d_j} \log \left( \frac{P(\epsilon(k_i|M_j))}{\alpha_j P(k_i|\mathcal{C})} \right) + n_q \log \alpha_j
\]

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:

$$\log \left( \frac{P(\mathbf{k}_i|\mathbf{M}_j)}{\alpha_j P(\mathbf{k}_i|\mathbf{C})} \right)$$

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.
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 $P(\notin k_i | M_j)$ 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:

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

where $P_s(k_i | M_j)$ is the smoothed distribution for terms in document $d_j$. 

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

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

That is,

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

Under the above assumptions, the smoothing parameter $\alpha_j$ is also a function of $P_s(k_i|M_j)$.

As a result, distinct smoothing methods can be obtained through distinct specifications of $P_s(k_i|M_j)$.

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:

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

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

\[
P_s(k_i|M_j, \lambda) = \frac{f_{i,j} + \lambda \sum_i F_i}{\sum_i f_{i,j} + \lambda}
\]

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.
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 $P_s^s(k_i|M_j)$ 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 $\alpha_j = \frac{1-\sum_{k_i \in d_j} P_s^s(k_i|M_j)}{1-\sum_{k_i \in d_j} P(k_i|C)}$
- compute the ranking using the formula

$$\log P(q|M_j) = \sum_{k_i \in q \land d_j} \log \left( \frac{P_s^s(k_i|M_j)}{\alpha_j P(k_i|C)} \right) + n_q \log \alpha_j$$
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.
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
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.
To fix this, let us define the average frequency $\bar{f}_{i,j}$ of term $k_i$ in document $d_j$ as

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

$$- \log P(k_i|C)$$

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

\[
1 - P(k_i|d_j)
\]
Given these assumptions, the weight $w_{i,j}$ of a term $k_i$ in a document $d_j$ is defined as

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

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

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

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

\[
- \log P(k_i | C) = - \log \left( \frac{e^{-\lambda_i} \lambda_i^{f_{i,j}}}{f_{i,j}!} \right)
\]

\[
\approx - f_{i,j} \log \lambda_i + \lambda_i \log e + \log(f_{i,j}!)
\]

\[
\approx f_{i,j} \log \left( \frac{f_{i,j}}{\lambda_i} \right) + \left( \lambda_i + \frac{1}{12f_{i,j} + 1} - f_{i,j} \right) \log e
\]

\[
+ \frac{1}{2} \log(2\pi f_{i,j})
\]

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

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

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

or

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

where \( \text{avg\_doclen} \) is the average document length in the collection and \( \text{len}(d_j) \) 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
  
  \[
  \sum_{\forall x_i} F_i(x_i, \Gamma_{x_i}) = 1 \\
  0 \leq F_i(x_i, \Gamma_{x_i}) \leq 1
  \]

  where \( x_i \) also refers to the states of the random variable associated to the node \( x_i \)
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 \((q, q_1, \text{and } q_2)\)
- 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
- $q_1 = (k_1 \land k_2) \lor k_i$
- $I = (q \lor q_1)$
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
The ranking is computed as $P(q \land d_j)$ 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 $\forall_{l \neq j} d_l = 0$, because we observe one document at a time.

\[
P(q \land d_j) = \sum_{\forall \vec{k}} P(q \land d_j \mid \vec{k}) \times P(\vec{k})
\]

\[
= \sum_{\forall \vec{k}} P(q \land d_j \land \vec{k})
\]

\[
= \sum_{\forall \vec{k}} P(q \mid d_j \land \vec{k}) \times P(d_j \land \vec{k})
\]

\[
= \sum_{\forall \vec{k}} P(q \mid \vec{k}) \times P(\vec{k} \mid d_j) \times P(d_j)
\]

\[
P(q \land d_j) = 1 - P(q \land d_j)
\]
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

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

$$\left( \prod_{\forall i|\text{on}(i, \vec{k})=1} P(k_i|d_j) \times \prod_{\forall i|\text{on}(i, \vec{k})=0} P(\bar{k}_i|d_j) \right)$$

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

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

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

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

where $|\overrightarrow{d}_j|$ stands for the norm of the vector $\overrightarrow{d}_j$
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:

\[
P(d_j) = \frac{1}{N} \quad P(\overline{d}_j) = 1 - \frac{1}{N}
\]

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

\[
P(k_i|d_j) = \begin{cases} 
1 & \text{if } k_i \in d_j \\
0 & \text{otherwise}
\end{cases}
\]

\[
P(\overline{k}_i|d_j) = 1 - P(k_i|d_j)
\]
We can use $P(k_i|d_j)$ and $P(q|\vec{k})$ factors to compute the evidential support the index terms provide to $q$:

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

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

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

By using these definitions in $P(q \land d_j)$ and $P(\neg q \land d_j)$ equations, we obtain the Boolean form of retrieval.
For a tf-idf ranking strategy

Prior probability $P(d_j)$ reflects the importance of document normalization

$$P(d_j) = \frac{1}{|\vec{d}_j|} \quad P(\overline{d}_j) = 1 - P(d_j)$$
For the document-term beliefs, we write:

\[
P(k_i | d_j) = \alpha + (1 - \alpha) \times \frac{f_{i,j}}{\max_i f_{i,j}} \times idf_i
\]

\[
P(\overline{k}_i | d_j) = 1 - P(k_i | d_j)
\]

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**:

\[
\overline{f}_{i,j} = \frac{f_{i,j}}{\max_i f_{i,j}}
\]

\[
idf_i = \frac{\log \frac{N}{n_i}}{\log N}
\]
For the term-query beliefs, we write:

\[
P(q|\vec{k}) = \sum_{k_i \in q} f_{i,j} \times w_q
\]

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

where \(w_q\) is a parameter used to set the maximum belief achievable at the query node.
By substituting these definitions into $P(q \land d_j)$ and $P(q \land d_j)$ 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.
Let $I = q \lor q_1$

In this case, the ranking provided by the inference network is computed as

$$P(I \land d_j) = \sum_{\vec{k}} P(I|\vec{k}) \times P(\vec{k}|d_j) \times P(d_j)$$

$$= \sum_{\vec{k}} (1 - P(q|\vec{k}) \times P(q_1|\vec{k})) \times P(\vec{k}|d_j) \times P(d_j)$$

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
By applying Bayes’ rule, we can write

\[
P(d_j | q) = \frac{P(d_j \land q)}{P(q)}
\]

\[
P(d_j | q) \sim \sum_{\forall \vec{k}} P(d_j \land q | \vec{k}) \times P(\vec{k})
\]

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

$$\vec{k}_i = \vec{k} \mid \text{on}(i, \vec{k}) = 1 \wedge \forall j \neq i \text{ on}(i, \vec{k}) = 0$$

The motivation is that tf-idf ranking strategies sum up the individual contributions of index terms.

We proceed as follows

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

$$P(\vec{q}|\vec{k}) = 1 - P(q|\vec{k})$$
Further, define

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

\[ P(\overline{d}_j|\vec{k}) = 1 - P(d_j|\vec{k}) \]

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