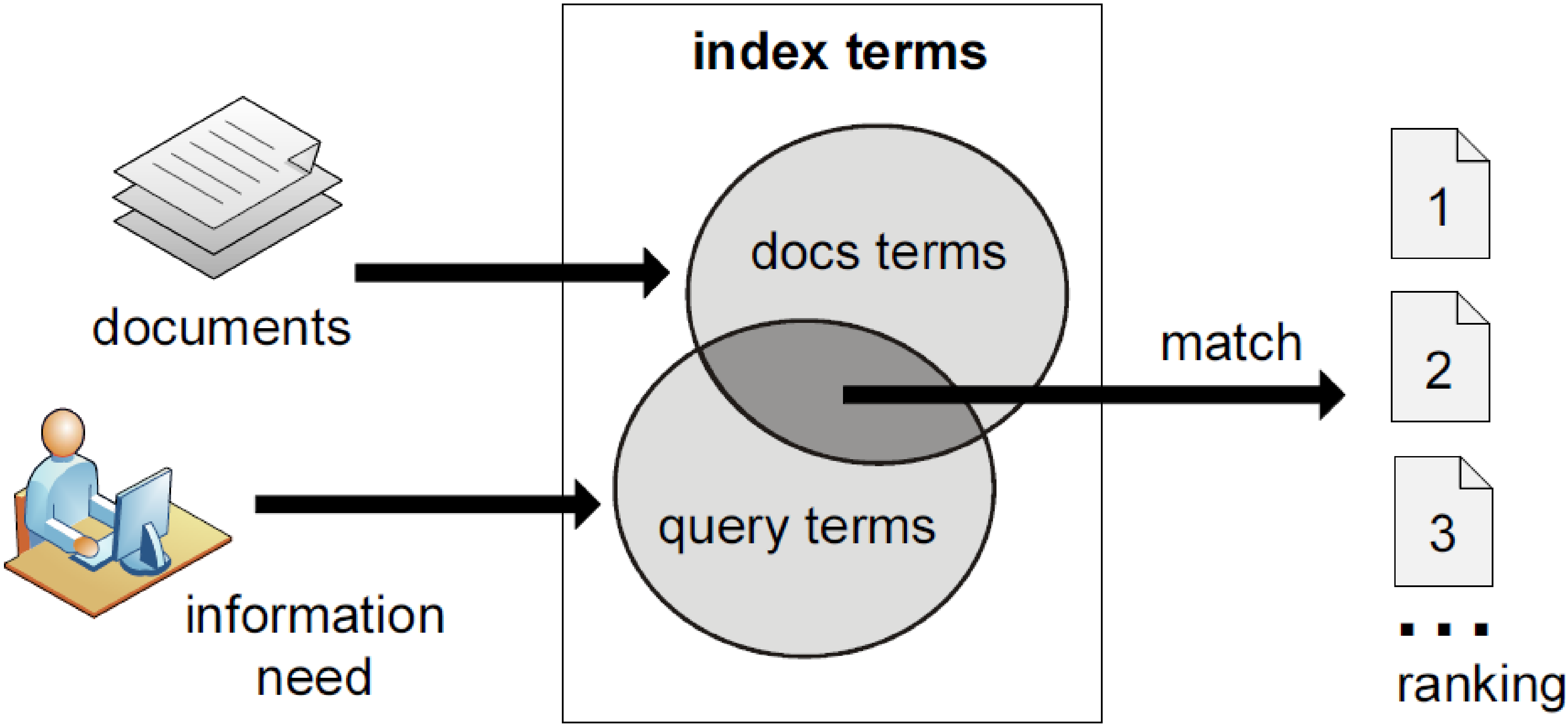
Introduction

- IR systems usually adopt index terms to process queries
- Index term:
  – a keyword or group of selected words
  – any word (more general)
- Stemming might be used:
  – connect: connecting, connection, connections
- An **inverted file** is built for the chosen index terms

Introduction

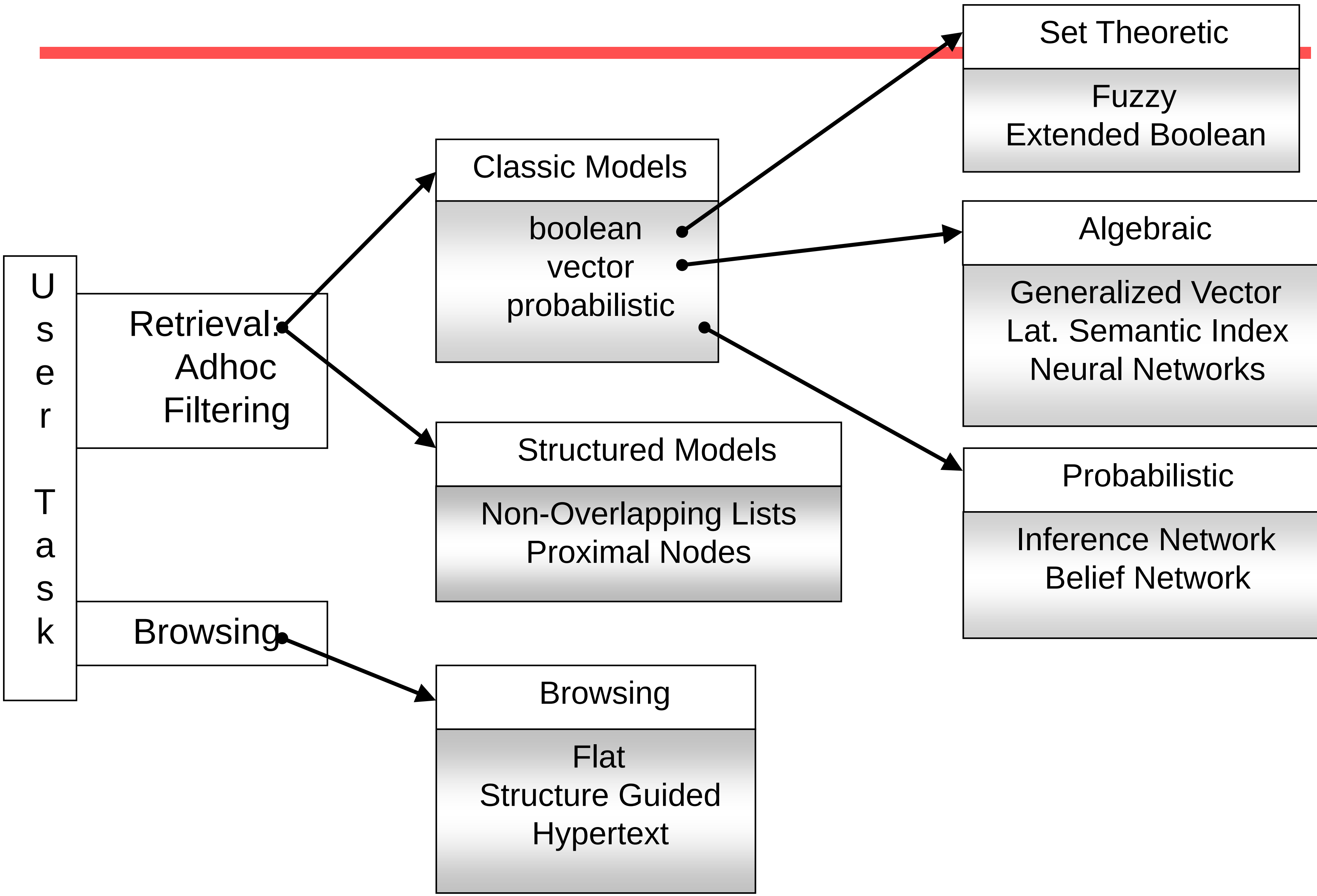


Introduction

- A ***ranking*** is an ordering of the documents retrieved that (hopefully) reflects the relevance of the documents to the user query
- A ranking is based on fundamental premisses regarding the notion of relevance, such as:
  - common sets of index terms
  - sharing of weighted terms
  - likelihood of relevance
- Each set of premisses leads to a distinct *IR model*

Tree, **Tree**
Tree, **Plant**
Tree, **Agriculture**

IR Models



User Task
Retrieval:
Adhoc
Filtering
Browsing
Classic Models
boolean
vector
probabilistic
Structured Models
Non-Overlapping Lists
Proximal Nodes
Browsing
Flat
Structure Guided
Hypertext
Set Theoretic
Fuzzy
Extended Boolean
Algebraic
Generalized Vector
Lat. Semantic Index
Neural Networks
Probabilistic
Inference Network
Belief Network

IR Models

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

LOGICAL VIEW OF DOCUMENTS

USER TASK

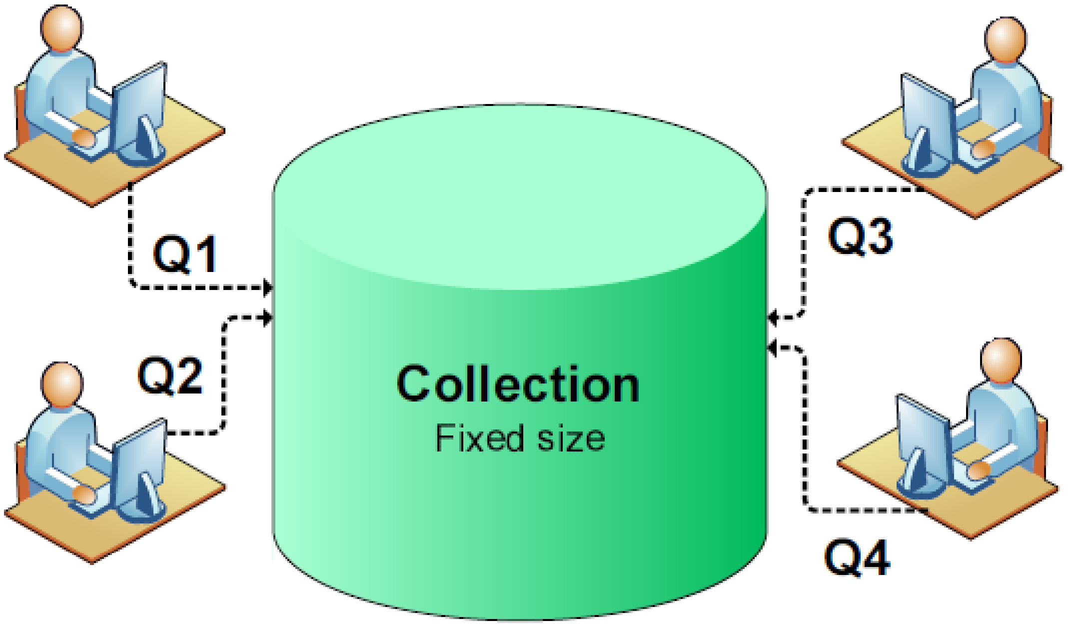
	Index Terms	**Full Text**	**Full Text + Structure**
Retrieval	Classic Set Theoretic Algebraic Probabilistic	Classic Set Theoretic Algebraic Probabilistic	Structured
Browsing	Flat	Flat Hypertext	Structure Guided Hypertext

Retrieval : Ad hoc and Filtering

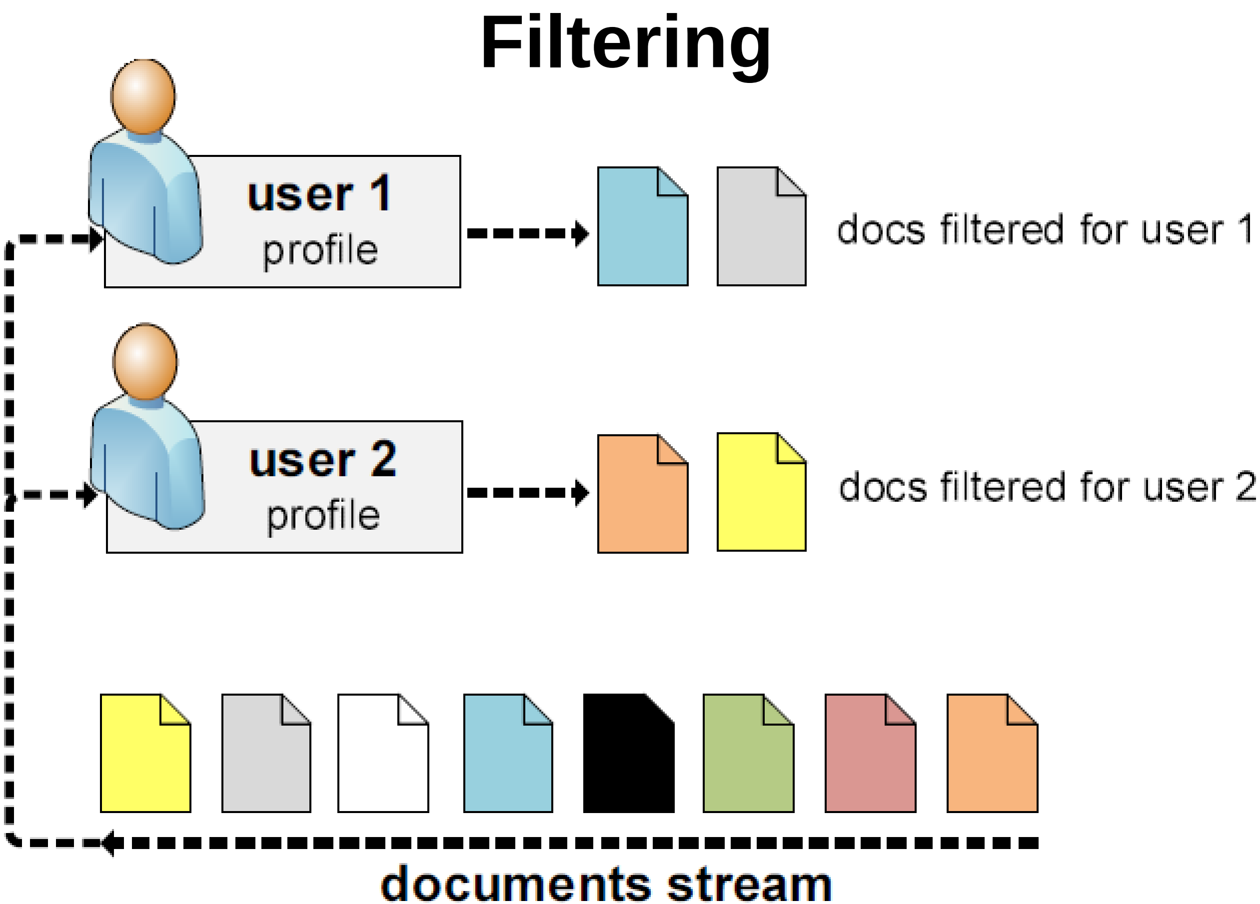
- **Ad hoc (Search)**: The documents in the collection remain relatively static while new queries are submitted to the system.
- **Routing (Filtering)**: The queries remain relatively static while new documents come into the system

Retrieval: Ad Hoc x Filtering

Ad hoc retrieval



Retrieval: Ad Hoc x Filtering



Classic IR Models - Basic Concepts

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

Classic IR Models - Basic Concepts

- Not all terms are equally useful for representing the document contents: ***less frequent terms allow identifying a narrower set of documents***
- The *importance* of the index terms is represented by weights associated to them
- Let
  - k_i be an index term
  - d_j be a document
  - w_{ij} is a weight associated with (k_i,d_j)
- The weight w_{ij} quantifies the importance of the index term for describing the document contents

Classic IR Models - Basic Concepts

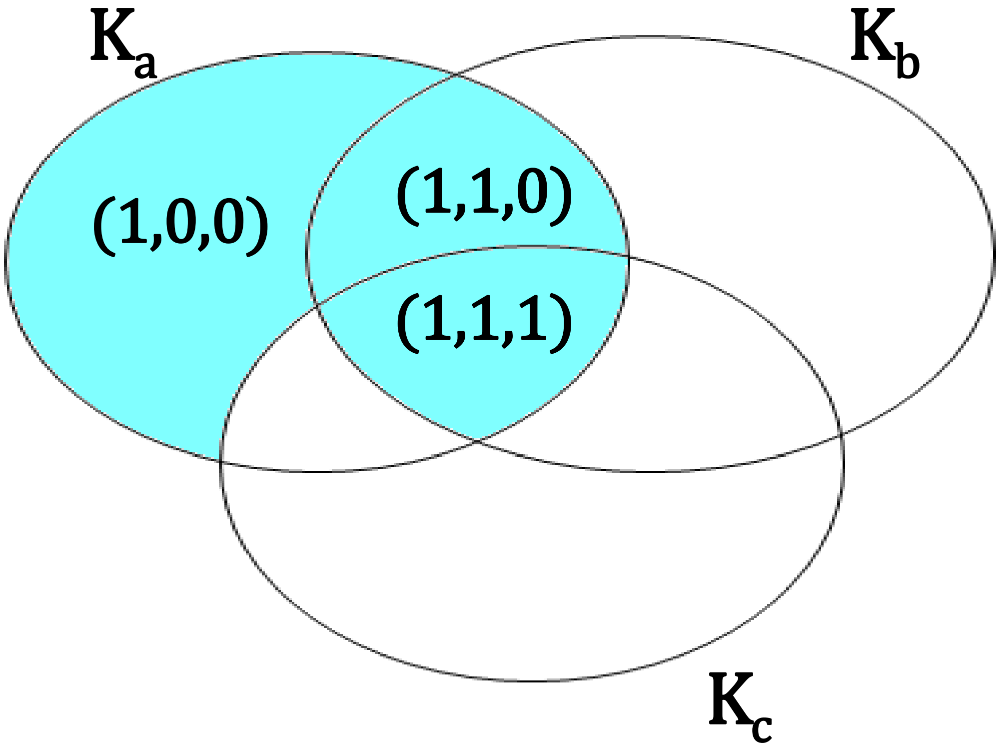
- ***K_i*** is an index term
- ***d_j*** is a document
- ***t*** is the total number of index terms
- ***$K = (k_1, k_2, \ldots, k_t)$*** is the set of all index terms
- ***$w_{ij} >= 0$*** is a weight associated with *(k_i,d_j)*
- ***$w_{ij} = 0$*** indicates that term does not belong to doc
- ***$vec(d_j) = (w_{1j}, w_{2j}, \ldots, w_{tj})$*** is a weighted vector associated with the document *d_j*
- ***$gi(vec(d_j)) = w_{ij}$*** is a function which returns the weight associated with pair ***(k_i,d_j)***

The Boolean Model

- Simple model based on set theory
- Queries specified as boolean expressions
  - precise semantics
  - neat formalism
  - $\boldsymbol{q = k_a \wedge (k_b \vee \neg k_c)}$
- Terms are either **present or absent**. Thus, $w_{ij}\ \varepsilon\ \{0,1\}$
- Consider
  - $\boldsymbol{q = k_a \wedge (k_b \vee \neg k_c)}$
  - $vec(qdnf) = (1,1,1) \vee (1,1,0) \vee (1,0,0)$
  - $vec(qcc) = (1,1,0)$ is a conjunctive component

The Boolean Model

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



The Boolean Model Example

เอกสารในระบบ

$D_j = \{K_{dog}, K_{cat}, K_{tiger}\}$

$D_1 = \{1,1,0\}$

$D_2 = \{1,0,0\}$

$D_3 = \{1,0,1\}$

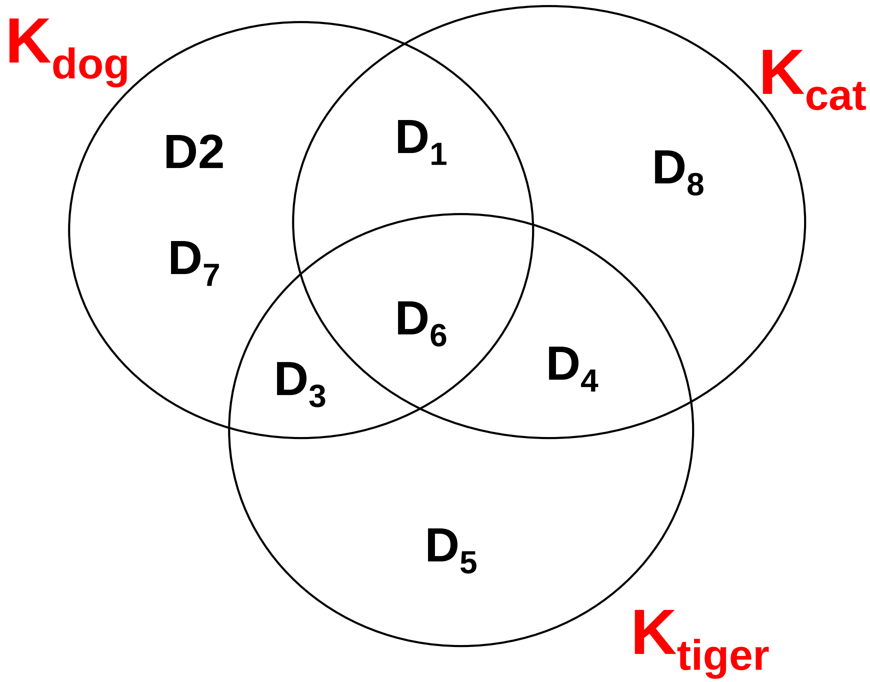
$D_4 = \{0,1,1\}$

$D_5 = \{0,0,1\}$

$D_6 = \{1,1,1\}$

$D_7 = \{1,0,0\}$

$D_8 = \{0,1,0\}$

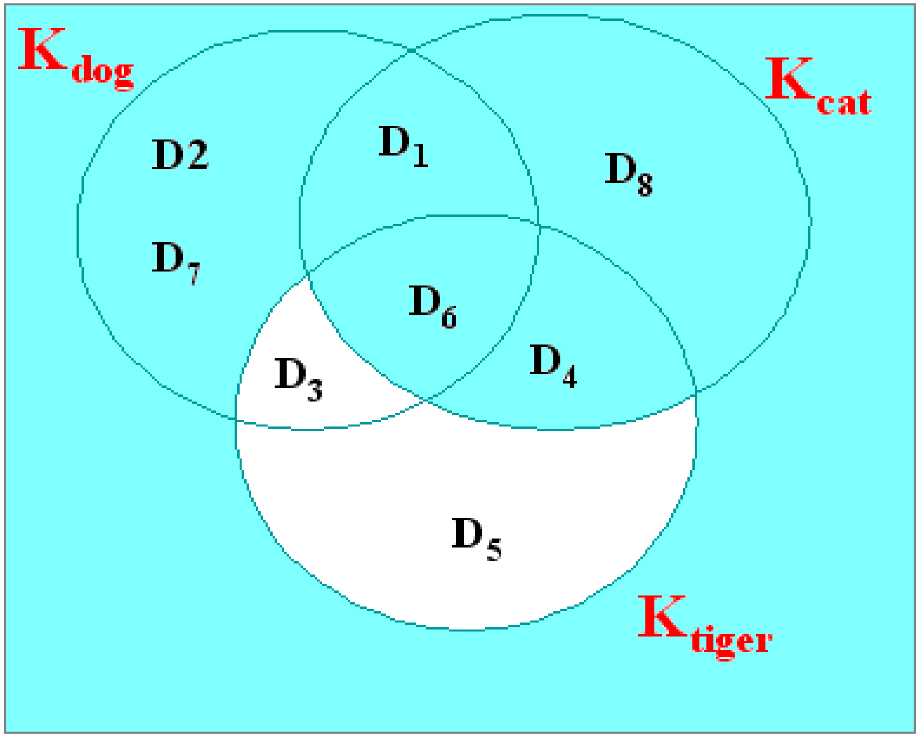


$$q = k_{dog} \wedge (k_{cat} \vee \neg k_{tiger})$$

The Boolean Model Example

$$q = k_{dog} \wedge (k_{cat} \vee \neg k_{tiger})$$

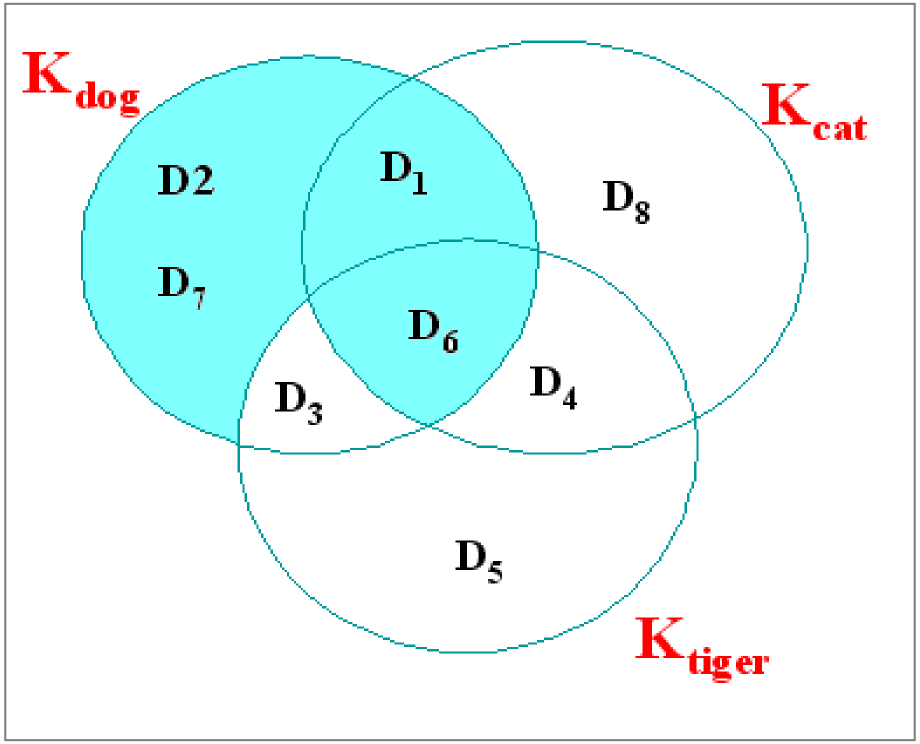
$(k_{cat} \vee \neg k_{tiger})$



The Boolean Model Example

$$q = k_{dog} \wedge (k_{cat} \vee \neg k_{tiger})$$

$$k_{dog} \wedge (k_{cat} \vee \neg k_{tiger})$$



Answer : **D1,D2,D6,D7**

The Boolean Model Example

Terms: $K_1, \ldots, K_8$

$D_1 = \{K_1, K_2, K_3, K_4, K_5\}$
$D_2 = \{K_1, K_2, K_3, K_4\}$
$D_3 = \{K_2, K_4, K_6, K_8\}$
$D_4 = \{K_1, K_3, K_5, K_7\}$
$D_5 = \{K_4, K_5, K_6, K_7, K_8\}$
$D_6 = \{K_1, K_2, K_3, K_4\}$

Query: $K_1 \wedge (K_2 \vee \neg K_3)$

Answer **:** $\{D_1, D_2, D_4, D_6\} \wedge (\{D_1, D_2, D_3, D_6\} \vee \{D_3, D_5\})$
$= \{D_1, D_2, D_6\}$

Boolean queries (cont.)

Example:

D1 = (A, B, C)

D2 = (A, C, D)

Query: (A **and** B) **or** (C **and** D)

	A and B	**C and D**	**Relevance**
D1	1	0	1
D2	0	1	1

Boolean queries (cont.)

Example:

D1 = (A, B, C)

D2 = (A, B, C, D)

Query: (A **and** ⌐D)

	A and ⌐D	**Relevance**
D1	1	1
D2	0	0

Drawbacks of the Boolean Model

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

Boolean query extensions

Example:

D1 = (A, B, C)

D2 = (A, C, D)

Query: (A **and** B) **or** (C **and** D)

	A and B	**C and D**	**Relevance**
D1	min(**1*1**,**1*1**) = **1**	min(**1*1**,**1*0**)= **0**	max(1,0)= **1**
D2	min(**1*1**,**1*0**)= **0**	min(**1*1**,**1*1**) = **1**	max(0,1) = **1**

Boolean query extensions

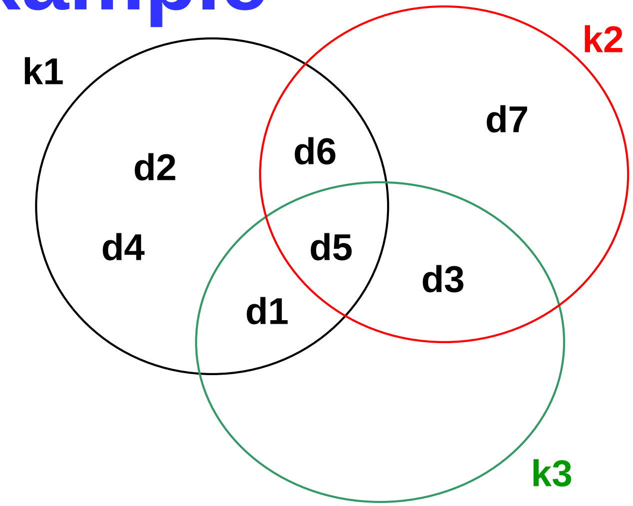
D1 = (0.8A, 0.5B, 0.6C)
D2 = (0.4A, 0.4B, 0.1C, 0.8D)
Query: (0.5A **and** 0.2B) **Or** (**not** D Or 0.3C)

	A and B	Not D	Relevance
D1	min(**0.5*0.8**,**0.2*0.5**) =0.10	max(**1**,**0.3*0.6**) = 1	max(0.1,1) = **1**
D2	min(**0.5*0.4**,**0.2*0.4**) =0.08	max(**0.2**,**0.3*0.1**) = 0.2	max(0.08,0.2)= **0.2**

The Vector Model

- Use of binary weights is too limiting
- Non-binary weights provide consideration for **partial matches**
- These term weights are used to compute a ***degree of similarity*** between a query and each document
- Ranked set of documents provides for better matching

The Vector Model: Example



	k1	k2	k3	q • dj
d1	2	0	1	5
d2	1	0	0	1
d3	0	1	3	11
d4	2	0	0	2
d5	1	2	4	17
d6	1	2	0	5
d7	0	5	0	10
q	1	2	3	

Document Collection

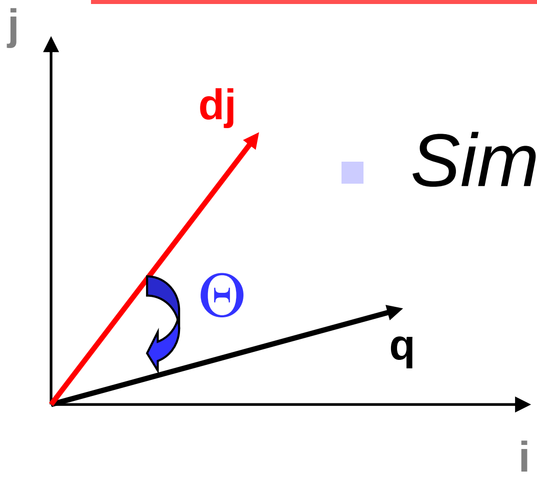
- A collection of *n* documents can be represented in the vector space model by a term-document matrix.
- An entry in the matrix corresponds to the "weight" of a term in the document; zero means the term has no significance in the document or it simply doesn't exist in the document.

$$
\begin{array}{ccccc}
 & T_1 & T_2 & \ldots & T_t \\
D_1 & w_{11} & w_{21} & \ldots & w_{t1} \\
D_2 & w_{12} & w_{22} & \ldots & w_{t2} \\
: & : & : & & : \\
: & : & : & & : \\
D_n & w_{1n} & w_{2n} & \ldots & w_{tn}
\end{array}
$$

The Vector Model

- Define:
  - $w_{ij} > 0$ whenever $k_i \in d_j$
  - $w_{iq} >= 0$ associated with the pair (k_i, q)
  - $vec(d_j) = (w_{1j}, w_{2j}, \ldots, w_{tj})$
    $vec(q) = (w_{1q}, w_{2q}, \ldots, w_{tq})$
  - To each term k_i is associated a unitary vector *vec(i)*
  - The unitary vectors *vec(i)* and *vec(j)* are assumed to be orthonormal (i.e., index terms are assumed to occur independently within the documents)
- The *t* unitary vectors *vec(i)* form an orthonormal basis for a t-dimensional space
- In this space, queries and documents are represented as weighted vectors

The Vector Model



- $Sim(q,d_j) = cos(\Theta)$

$$= \frac{(vec(d_j) * vec(q))}{|d_j| * |q|} = \frac{\left(\sum w_{ij} * w_{iq}\right)}{|d_j| * |q|}$$

- Since $w_{ij} > 0$ and $w_{iq} > 0$,

  $\mathbf{0 <= sim(q,d_j) <= 1}$

- A document is retrieved even if it matches the query terms only partially

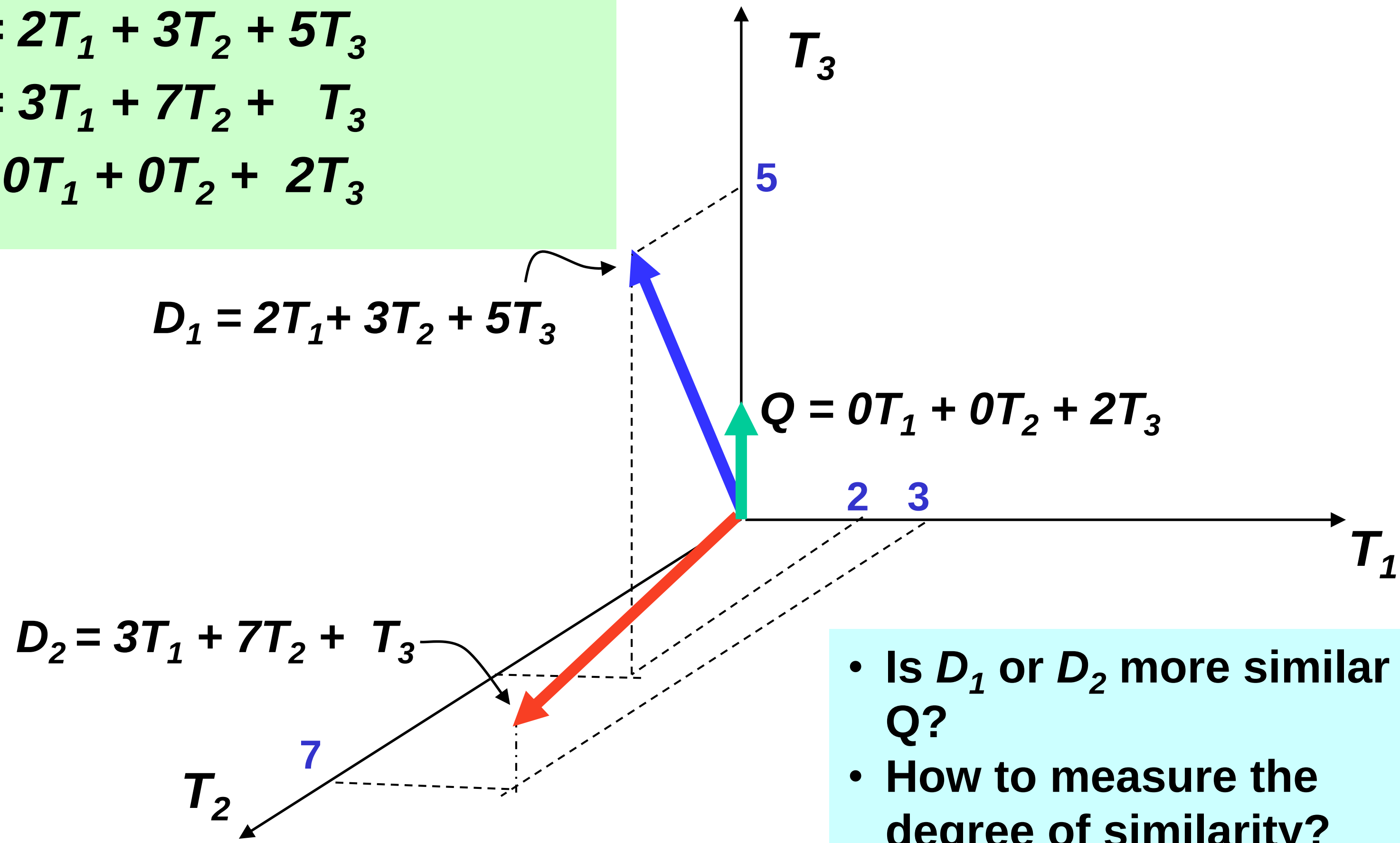
Graphic Representation

Example:

$D_1 = 2T_1 + 3T_2 + 5T_3$

$D_2 = 3T_1 + 7T_2 + T_3$

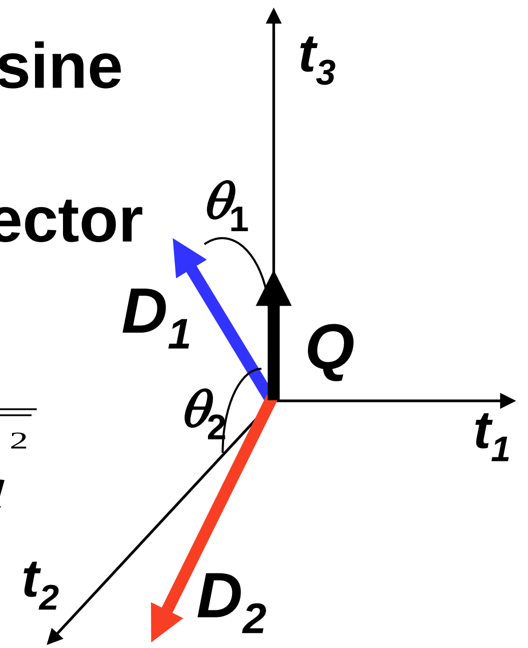
$Q = 0T_1 + 0T_2 + 2T_3$



- **Is D_1 or D_2 more similar to Q?**
- **How to measure the degree of similarity? Distance? Angle? Projection?**

Cosine Similarity Measure

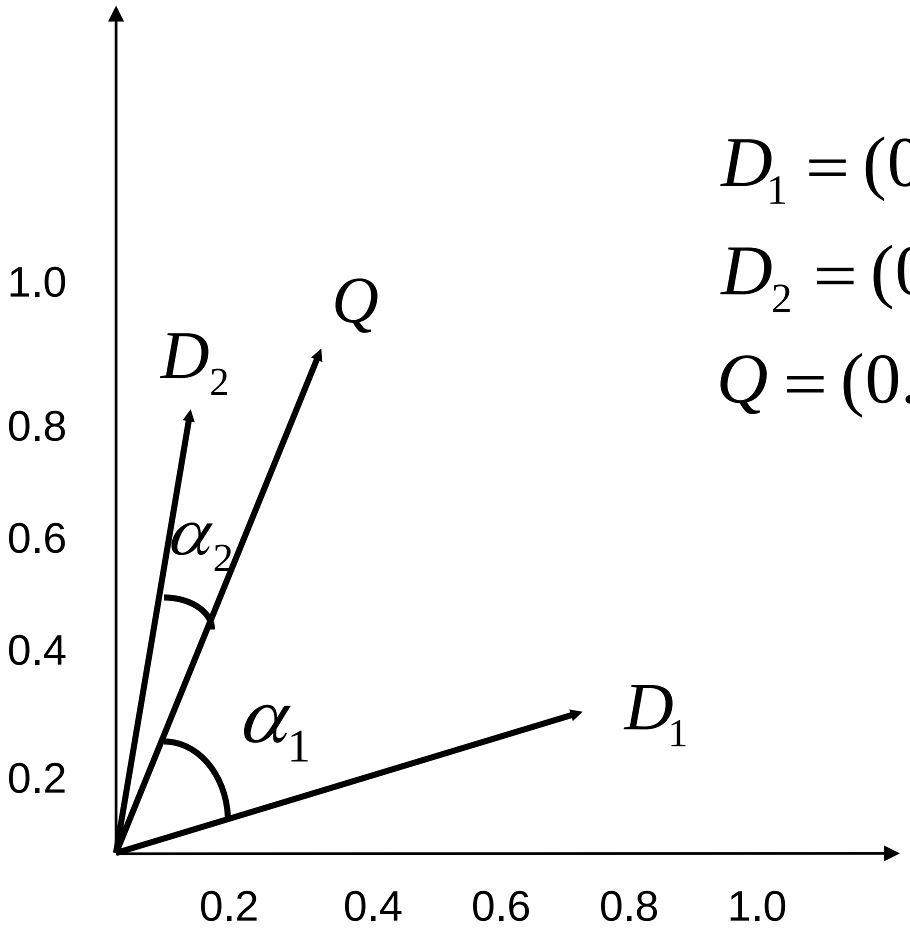
- **Cosine similarity measures the cosine of the angle between two vectors.**
- **Inner product normalized by the vector lengths.**



$$\text{CosSim}(d_j, q) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \cdot |\vec{q}|} = \frac{\sum_{i=1}^{t}(w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$

$D_1 = 2T_1 + 3T_2 + 5T_3$ CosSim(D_1 , Q) = $10 / \sqrt{(4+9+25)(0+0+4)} = 0.81$
$D_2 = 3T_1 + 7T_2 + 1T_3$ CosSim(D_2 , Q) = $2 / \sqrt{(9+49+1)(0+0+4)} = 0.13$
$Q = 0T_1 + 0T_2 + 2T_3$

Computing Similarity Scores



$$D_1 = (0.8, 0.3)$$

$$D_2 = (0.2, 0.7)$$

$$Q = (0.4, 0.8)$$

Computing a similarity score

Say we have query vector $Q = (0.4, 0.8)$

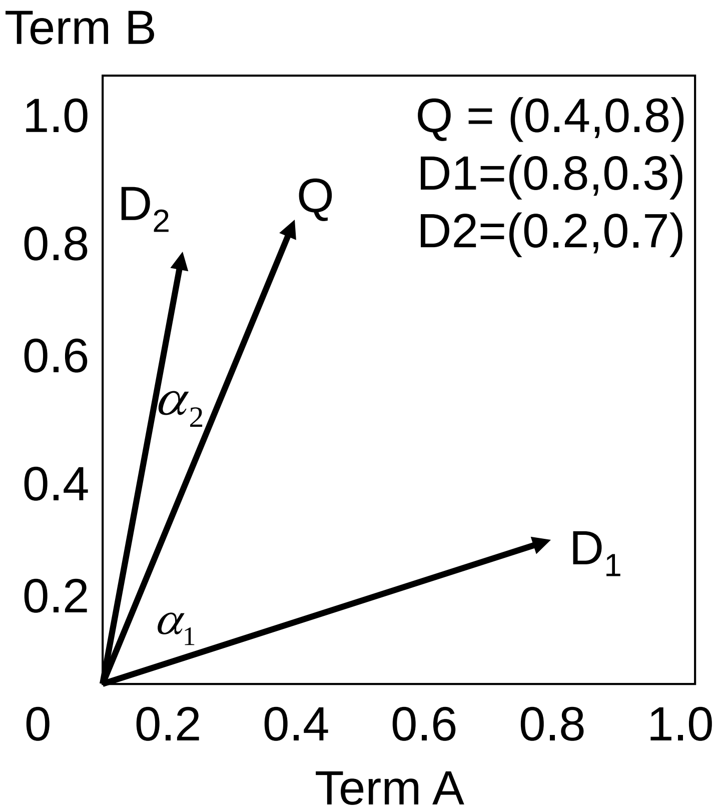
Also, document $D_2 = (0.2, 0.7)$

What does their similarity comparison yield?

$$sim(Q, D_2) = \frac{(0.4 * 0.2) + (0.8 * 0.7)}{\sqrt{[(0.4)^2 + (0.8)^2] * [(0.2)^2 + (0.7)^2]}}$$

$$= \frac{0.64}{\sqrt{0.42}} = 0.98$$

Vector Space with Term Weights and Cosine Matching



$D_i = (d_{i1}, w_{di1}; d_{i2}, w_{di2}; \ldots; d_{it}, w_{dit})$

$Q = (q_{i1}, w_{qi1}; q_{i2}, w_{qi2}; \ldots; q_{it}, w_{qit})$

$$sim(Q, D_i) = \frac{\sum_{j=1}^{t} w_{q_j} w_{d_{ij}}}{\sqrt{\sum_{j=1}^{t} (w_{q_j})^2 \sum_{j=1}^{t} (w_{d_{ij}})^2}}$$

$$sim(Q, D2) = \frac{(0.4 \cdot 0.2) + (0.8 \cdot 0.7)}{\sqrt{[(0.4)^2 + (0.8)^2] \cdot [(0.2)^2 + (0.7)^2]}}$$

$$= \frac{0.64}{\sqrt{0.42}} = 0.98$$

$$sim(Q, D_1) = \frac{0.56}{\sqrt{0.58}} = 0.74$$

The Vector Model

- *Sim(q,dj)* = $\dfrac{\left(\sum w_{ij} * w_{iq}\right)}{|d_j| * |q|}$
- How to compute the weights w_{ij} and w_{iq} ?
- A good weight must take into account two effects:
  - quantification of ***intra-document*** contents (similarity)
    - *tf* factor, the ***term frequency*** within **a document**
  - quantification of ***inter-documents*** separation (dissi-milarity)
    - *idf* factor, the ***inverse document frequency***
  - $w_{ij} = tf(i,j) * idf(i)$

TF-IDF Weighting

- A typical combined term importance indicator is *tf-idf weighting*:

$$W_{ij} = tf_{ij} * idf_i = tf_{ij} * \log(\frac{N}{n_i})$$

- A term occurring frequently in the document but rarely in the rest of the collection is given ***high weight***.
- Many other ways of determining term weights have been proposed.
- Experimentally, *tf-idf* has been found to work well.

Computing TF-IDF -- An Example

Given **a document** containing terms with given frequencies: ***(intra-document***)

A(3), B(2), C(1)

Assume **collection contains** 10,000 documents and document frequencies of these terms are: ***(inter-documents)***

A(50), B(1300), C(250)

Then:

A: tf = 3/3; idf = log(10000/50) = 2.30; tf*idf = 2.30

B: tf = 2/3; idf = log(10000/1300) = 0.89; tf*idf = 0.59

C: tf = 1/3; idf = log(10000/250) = 1.61; tf*idf = 0.53

Example

1) Once upon a **midnight** dreary, while I pondered, weak and weary.
2) Over many a quaint and curious **volume** of forgotten **lore**.
3) While I nodded, nearly napping, suddenly there came a **tap**ping.
4) As of some one gently rapping **door**, rapping at my **chamber door**.
5) "Tis some **visitor**," I muttered, "tapping at my **chamber door" .**
6) Only this, and **nothing** more.

N=6

Key1 "**chamber**" *frequency* = 2

Key2 "**door**" *frequency* = 3

Key3 "**lore**" *frequency* = 1

Key4 "**midnight**" *frequency* = 1

Key5 "**nothing**" *frequency* = 1

Key6 "**tap**" *frequency* = 1

Key7 "**visitor**" *frequency* = 1

Key8 "**volumn**" *frequency* = 1

Example

	chamber	door	lore	midnight	nothing	tap	visitor	volume
Doc1	0	0	0	1	0	0	0	0
Doc2	0	0	1	0	0	0	0	1
Doc3	0	0	0	0	0	1	0	0
Doc4	1	2	0	0	0	0	0	0
Doc5	1	1	0	0	0	0	1	0
Doc6	0	0	0	0	1	0	0	0

Computing TF-IDF (Only Doc4)

-Given a document containing terms with given frequencies:

Chamber(1), Door(2) --- > max = 2

- collection contains 6 documents

Chamber(2), Door(2)

Then:

Chamber:

tf = 1/2; idf = log(6/2) = 0.48; tf*idf = 0.24

Door:

tf = 2/2; idf = log(6/2) = 0.48; tf*idf = 0.48

Example

	chamber	door	lore	midnight	nothing	tap	visitor	volume
Doc1	0	0	0	0.78	0	0	0	0
Doc2	0	0	0.78	0	0	0	0	0.78
Doc3	0	0	0	0	0	0.78	0	0
Doc4	0.24	0.48	0	0	0	0	0	0
Doc5	0.48	0.48	0	0	0	0	0.78	0
Doc6	0	0	0	0	0.78	0	0	0

The Vector Model

- Let,
  - N be the total number of docs in the collection
  - n_i be the number of docs which contain *ki*
  - *freq(i,j)* raw frequency of k_i within d_j
- A normalized *tf* factor is given by
  - ***f(i,j) = freq(i,j) / max(freq(l,j))***
  - where ***the maximum is computed over all terms which occur within the document*** d_j
- The *idf* factor is computed as
  - ***idf(i) = log(N/n**$_i$**)***
  - the *log* is used to make the values of *tf* and *idf* comparable. It can also be interpreted as the *amount of information* associated with the term k_i.

The Vector Model

- The best term-weighting schemes use weights which are give by
  - *wij = f(i,j) * log(N/ni)*
  - the strategy is called a *tf-idf* weighting scheme
- For the query term weights, a suggestion is

$$W_{i,q} = \left(0.5 + \frac{0.5 * freq_{i,q}}{Max(freq_{I,q})} \right) * \log\left(\frac{N}{n_i} \right)$$

- The vector model with *tf-idf* weights is a good ranking strategy with general collections
- The vector model is usually as good as the known ranking alternatives. It is also simple and fast to compute.

The Vector Model

Query: ″Visitor **at your door or my door**″

Translation to known keyterms: **{ ″visitor″,″door″,″door″ }**

$$W_{i,q} = (0.5 + \frac{0.5 * freq_{i,q}}{Max(freq_{i,q})}) * \log(\frac{N}{n_i})$$

Door $\rightarrow W_{2,q} = (0.5 + \frac{0.5*2}{2}) * \log(\frac{6}{2}) = 0.477$

Visitor $\rightarrow W_{7,q} = (0.5 + \frac{0.5*1}{2}) * \log(\frac{6}{1}) = 0.584$

Resulting query vector = (0,0.477,0,0,0,0,0.584,0)

Example

	chamber	door	lore	midnight	nothing	tap	visitor	volume
Doc1	0	0	0	0.78	0	0	0	0
Doc2	0	0	0.78	0	0	0	0	0.78
Doc3	0	0	0	0	0	0.78	0	0
Doc4	0.24	0.48	0	0	0	0	0	0
Doc5	0.48	0.48	0	0	0	0	0.78	0
Doc6	0	0	0	0	0.78	0	0	0
q	**0**	**0.477**	**0**	**0**	**0**	**0**	**0.584**	**0**

$$sim(Q, D_i) = \frac{\sum_{j=1}^{t} w_{q_j} w_{d_{ij}}}{\sqrt{\sum_{j=1}^{t} (w_{q_j})^2 \sum_{j=1}^{t} (w_{d_{ij}})^2}}$$

$$sim(Q, D_4) = \frac{0.48 * 0.477}{\sqrt{[(0.477)^2 + (0.584)^2] * [(0.24)^2 + (0.48)^2]}}$$

$$= \frac{0.229}{0.405} = 0.566$$

$$sim(Q, D_1) = sim(Q, D_2)$$
$$= sim(Q, D_3)$$
$$= sim(Q, D_6)$$
$$= 0$$

$$sim(Q, D_5) = \frac{(0.48 * 0.477) + (0.78 * 0.584)}{\sqrt{[(0.477)^2 + (0.584)^2] * [(0.48)^2 + (0.48)^2 + (0.78)^2]}}$$

$$= \frac{0.684}{0.779} = 0.879$$

New Rank → **D_5,D_4,D_1,D_2,D_3,D_6**

TF-IDF Smoothing

$$tf_{i,j} = \frac{f_{i,j}}{max_i f_{i,j}}$$

	Cat	Dog	Tiger
D1	2	5	1
D2	1	4	0
D3	5	2	7
D4	2	1	0

$$\boldsymbol{tf_{dog,1} = \frac{5}{5} = 1}$$

$$\boldsymbol{tf_{dog,2} = \frac{4}{4} = 1}$$

TF-IDF Smoothing

$$tf_{i,j} = \frac{f_{i,j}}{max_i f_{i,j}}$$

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

TF-IDF Smoothing

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

	Cat	Dog	Tiger
D1	2	5	1
D2	1	4	0
D3	5	2	7
D4	2	1	0

$$\boldsymbol{idf_{cat} = \log(\frac{4}{4}) = 0}$$

$$\boldsymbol{w_{cat,j} = x * 0 = 0}$$

$$\boldsymbol{idf_{dog} = \log(\frac{4}{4}) = 0}$$

$$\boldsymbol{w_{dog,j} = x * 0 = 0}$$

TF-IDF Smoothing

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

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

Modeling
(Structured Text Models)

Introduction

- Keyword-based query answering considers that *the documents are **flat*** i.e., a word in the title has the same weight as a word in the body of the document
- But, the document structure is one additional piece of information which can be taken advantage of
- For instance, words appearing in the title or in sub-titles within the document could receive higher

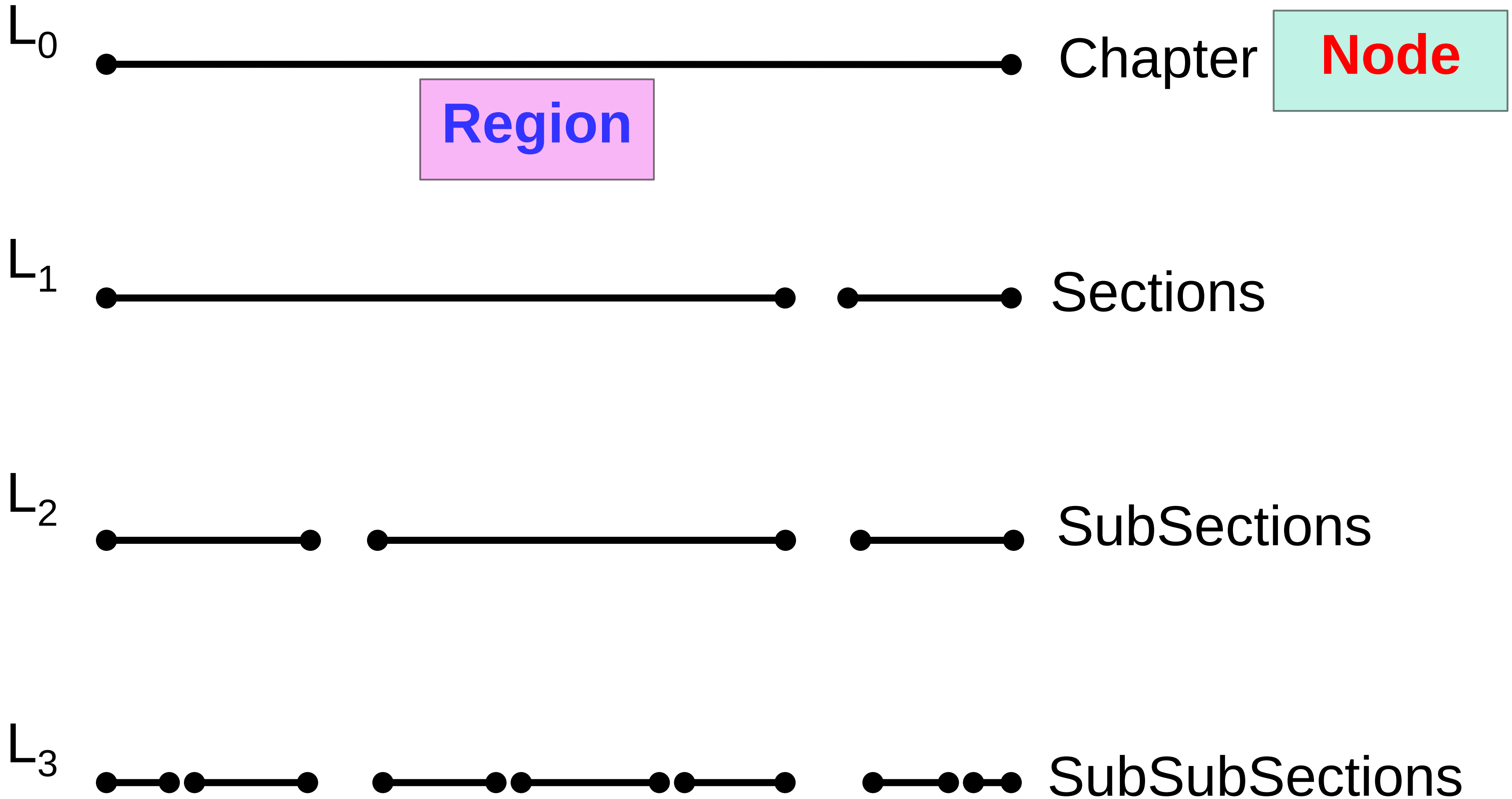
Basic Definitions

- ***Match point:*** the position in the text of a sequence of words that match the query
  - Query: “atomic holocaust in Hiroshima”
  - Doc d_j: contains 3 lines with this string
  - Then, doc d_j contains 3 match points
- ***Region*:** a contiguous portion of the text
- ***Node*:** a structural component of the text such as a chapter, a section, etc

Non-Overlapping Lists

- Due to Burkowski, 1992.
- Idea: divide the text in *non-overlapping* regions which are collected in a *list*
- Multiple ways to divide the text in non-overlapping parts yield multiple lists:
  - a list for chapters
  - a list for sections
  - a list for subsections
- Text regions from distinct lists might overlap

Non-Overlapping Lists



Non-Overlapping Lists

Vocabulary	Occurrences (a list of text regions)
Component A	(70, 200), (1330, 1420), ...
Component B	(415, 580), (5500, 5720), ...
Component C	(100, 130), ...
.	.
....	....

a structure component (chapter, section, …)

A inverted-file structure for non-overlapping lists

Conclusions

- The non-overlapping lists model is simple and allows efficient implementation
- But, types of queries that can be asked are limited "**Database in Chapter Name and Section Name**"
- Also, model does not include any provision for ranking the documents *by* ***degree of similarity*** to the query
- What does structural similarity mean?

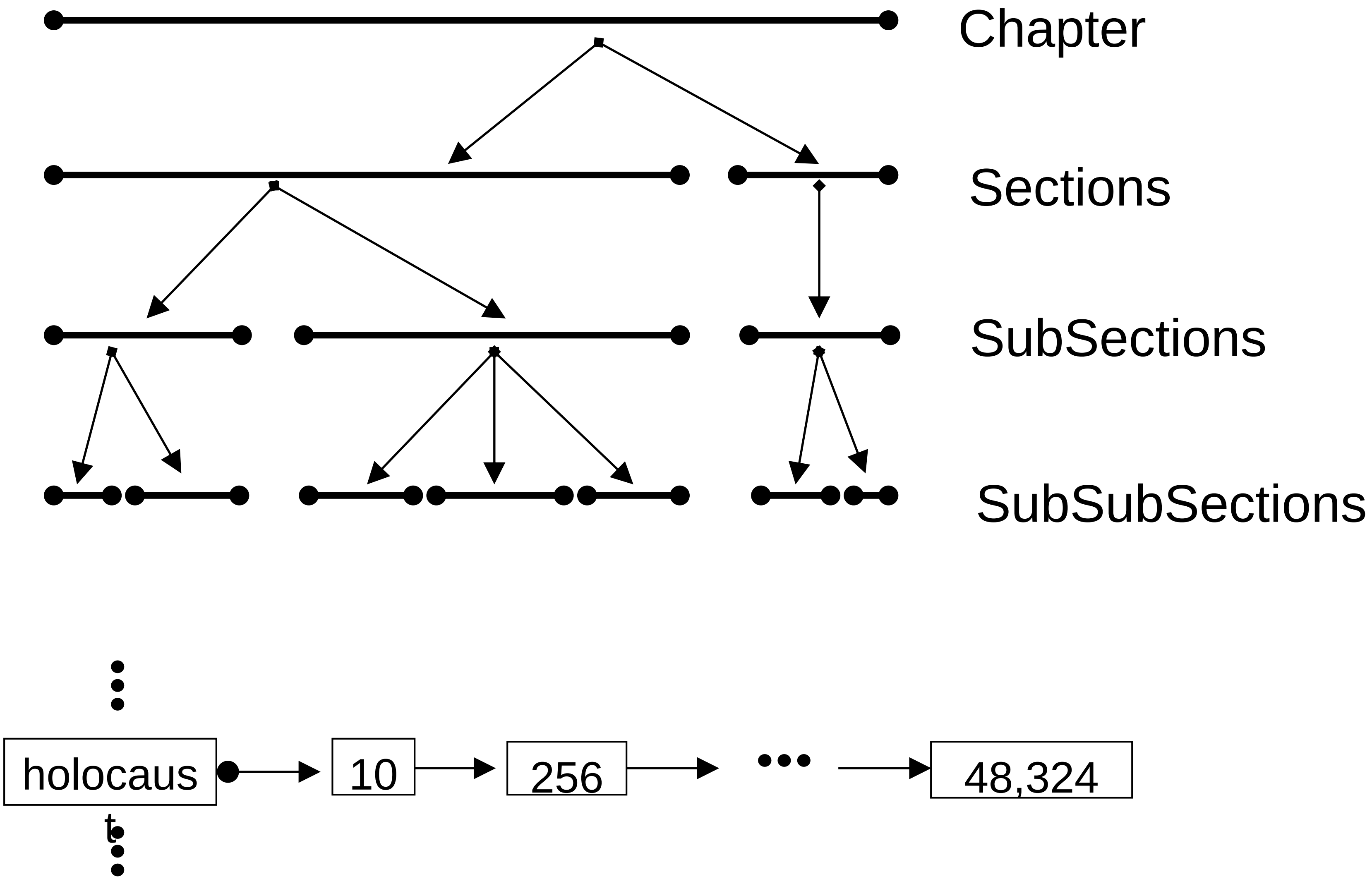
Proximal Nodes

- Due to Navarro and Baeza-Yates, 1997
- Idea: define a strict **hierarchical index over the text**. This enriches the previous model that used flat lists.
- **Multiple index hierarchies** might be defined
- Two distinct index hierarchies might refer to text regions that overlap

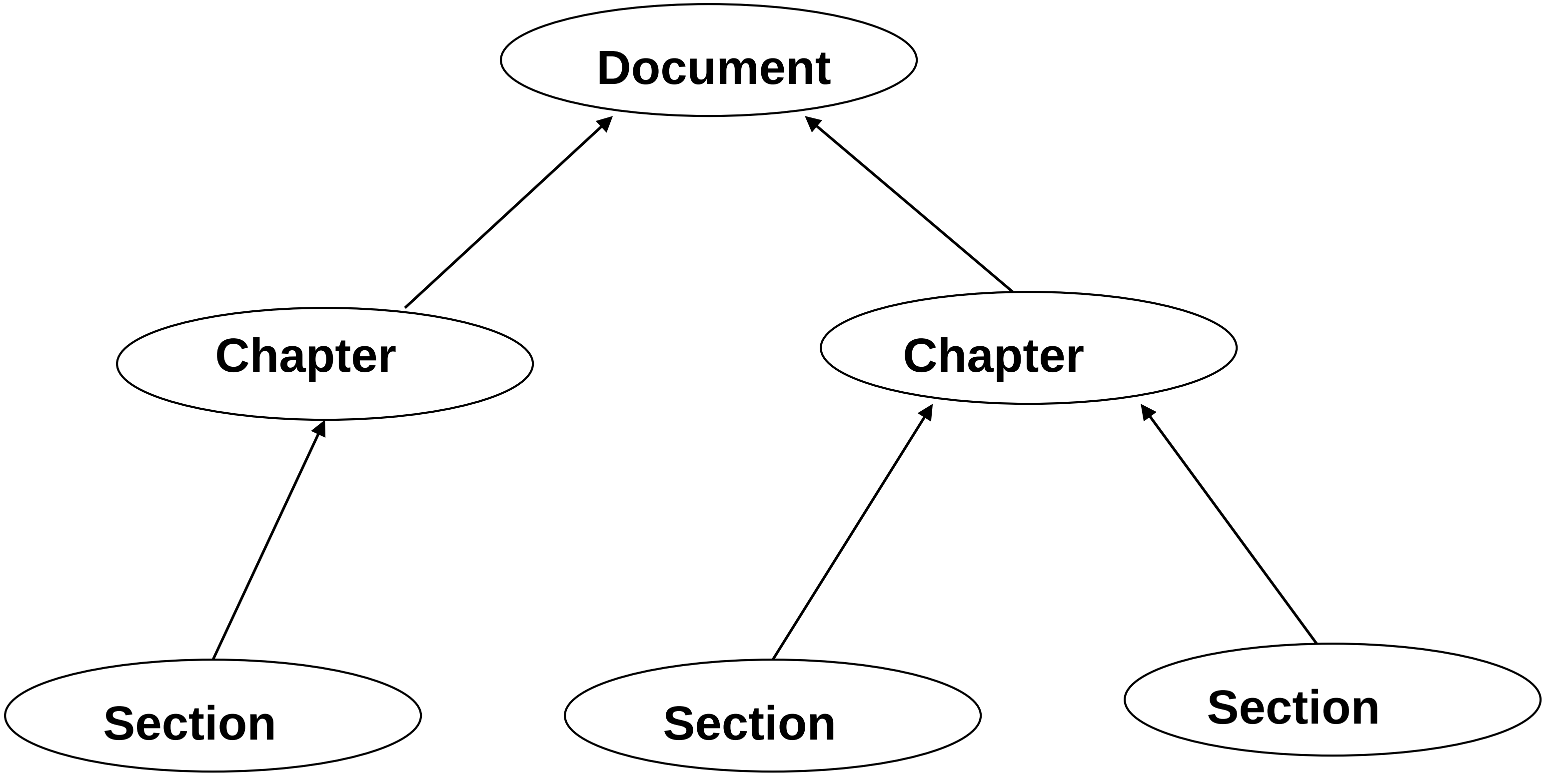
Definitions

- Each indexing structure is a strict hierarchy composed of
  - chapters
  - sections
  - subsections
  - paragraphs
  - lines
- Each of these components is called a ***node***
- To each node is associated a text region

Proximal Nodes



Proximal Nodes



Proximal Nodes

- Key points:
  - In the hierarchical index, one node might be contained within another node
  - But, two nodes of a same hierarchy cannot overlap
  - The inverted list for keywords complements the hierarchical index
  - The implementation here is more complex than that for non-overlapping lists

Conclusions

- Model allows formulating queries that are more sophisticated than those allowed by non-overlapping lists
- To speed up query processing, nearby nodes are inspected
- Types of queries that can be asked are somewhat limited (all nodes in the answer must come from a same index hierarchy!)
- Model is a compromise between efficiency and expressiveness

Model for Browsing

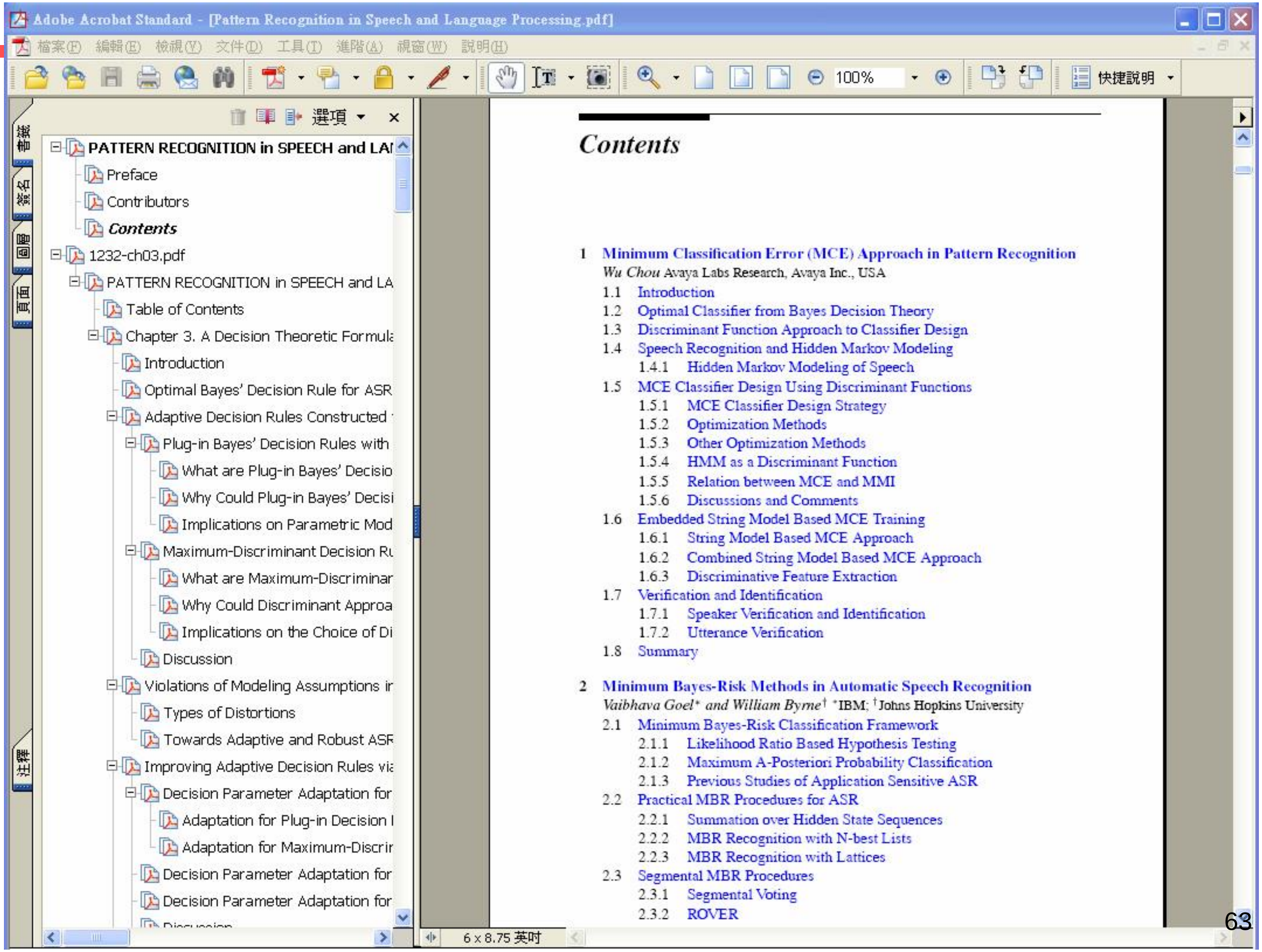
- Flat Browsing
- Structure Guided Browsing
- The Hypertext Model

Flat Browsing

-**Documents represented as dots in**
  - A two-dimensional plane
  - A one-dimensional plane (list)

-**User not know indicator,subject,page ...**

Structure Guided Browsing



Hypertext browsing

Information Retrieval in Hypertext

Home | Hypertext | IR Issues | Auto Link Generation | My System | Bibliography

Navigation or browsing is effective for small hypertext systems. For large hypertext databases, information retrieval (IR) through queries becomes crucial, although some well-structured hypertext systems, such as Victorian Web, can be navigated smoothly even without the help of information retrieval. However, Information retrieval systems serve the purpose of finding data items that are relevant to the users query request. The World Wide Web, as the largest hypertext system, is a tool that has become very popular as a means to easily access information from other sites. It is almost impossible to explore such a huge collection of various hypertext documents. **Thus information retrieval plays an extremely critical role in hypertext systems.** Of course, conversely, I argue here **hyperlinks can greatly reinforce the usage of information retrieval systems**.

Conklin had suggested that search and query mechanisms can present information at a manageable level of complexity and detail [Conklin, 1987]. Halasz's view was that "*navigational access itself is not sufficient. Effective access to information stored in a hypermedia network requires query-based access to complement navigation.........search and query needs to be elevated to a primary access mechanism on par with navigation.*" [Halasz, 1988].

Information retrieval is a large research area, mostly concerned with finding information in textual material [Bärtschi85], [Salton89]. The simplest form of information retrieval is the full text search, which finds occurrences of words or phrases specified by the user, combined by boolean operators and weighting of words based on their statistical properties. When a hyperdocument is simply regarded as a text database (ignoring the links) this type of information retrieval is the same as for other textual databases, like dictionaries, encyclopedia, on-line library catalogs, etc.

Finding information is a three-step process:

1. **finding documents**: in a centralized hypertext this is no problem; but on the Internet (World Wide Web) it may be difficult to get access to all potentially interesting documents (because there are millions of them, distributed over the whole world).
2. **formulating queries**: the user needs to express exactly what kind of information (s)he is looking for.
3. **determining relevance**: the system must determine whether a document contains the information the user is looking for.

Traditional information retrieval research and development has concentrated on the second and third step. The distributed nature of the Internet, as well as the size of large hypertexts on CD-ROM, requires shifting the focus towards the first step.

When a text database is large, but centralized, special indexing mechanisms can be employed to speed up the search. For relatively static documents, a popular indexing mechanism is the use of inverted files. Recently a new indexing mechanism, called Glimpse is becoming popular, because it requires much less space overhead than inverted files and other indexing techniques.

Bruza [Bruza-90] proposed a two-level hypertext architecture for hyperdocuments, containing a hyperindex used for information retrieval. First the index term describing the required information would be searched, followed by a "beam down" operation to the hyperdocument itself, to evaluate the selected nodes from the hyperdocument. Bruza proposed measures to determine the effectiveness of index expressions in the hyperindex.

The result of a search may be either a pointer to the first match found, or a scored list of matches. Information retrieval is inherently uncertain: a very general query (like asking for one keyword) may yield too many answers, while a very specific query may result in no answers at all.

-Not fully understand, Random read, Nonsequence browse, Miscommunication for reader and writer