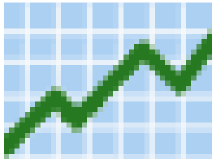
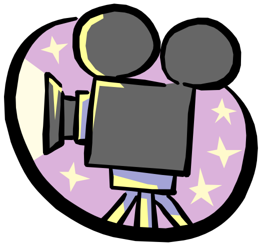
Multimedia and Text Indexing

Multimedia Data Management

• The need to query and analyze vast amounts of multimedia data (i.e., images, sound tracks, video tracks) has increased in the recent years.

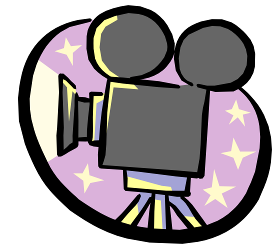
• Joint Research from Database Management, Computer Vision, Signal Processing and Pattern Recognition aims to solve problems related to multimedia data management.





Multimedia Data





- There are four major types of multimedia data: images, video sequences, sound tracks, and text.
- From the above, the easiest type to manage is text, since we can order, index, and search text using string management techniques, etc.
- Management of simple sounds is also possible by representing audio as signal sequences over different channels.
- Image retrieval has received a lot of attention in the last decade (CV and DBs). The main techniques can be extended and applied also for video retrieval.

Content-based Image Retrieval

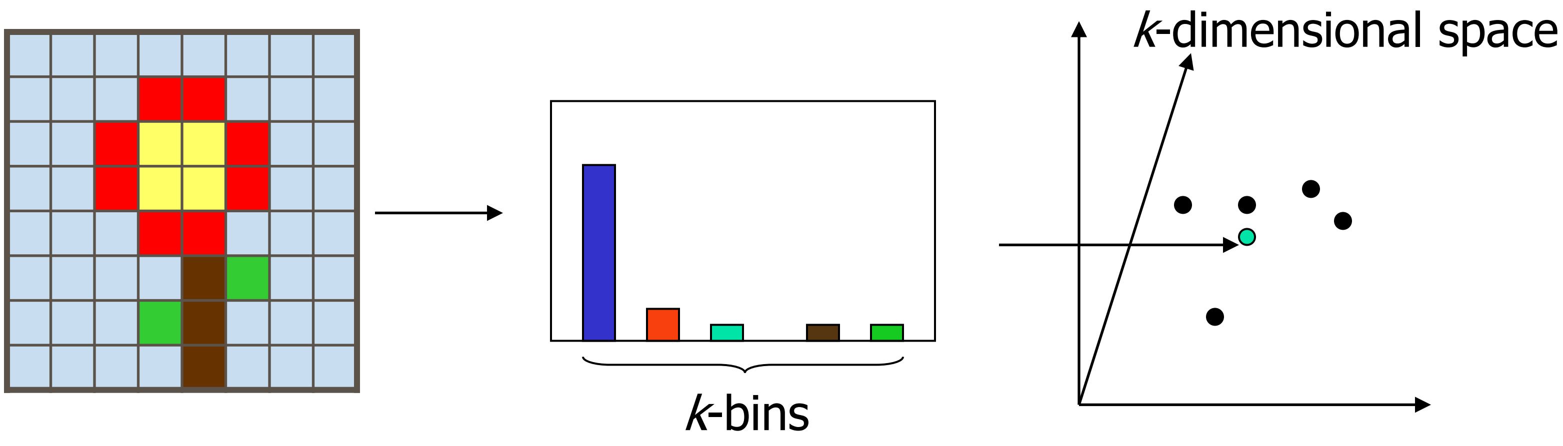
- Images were traditionally managed by first annotating their contents and then using text-retrieval techniques to index them.
- However, with the increase of information in digital image format some drawbacks of this technique were revealed:
  - Manual annotation requires vast amount of labor
  - Different people may perceive differently the contents of an image; thus no objective keywords for search are defined
- A new research field was born in the 90's: *Content-based Image Retrieval* aims at indexing and retrieving images based on their *visual contents*.

Feature Extraction

- The basis of Content-based Image Retrieval is to extract and index some *visual features* of the images.
- There are general features (e.g., color, texture, shape, etc.) and domain-specific features (e.g., objects contained in the image).
  - Domain-specific feature extraction can vary with the application domain and is based on pattern recognition
  - On the other hand, general features can be used independently from the image domain.

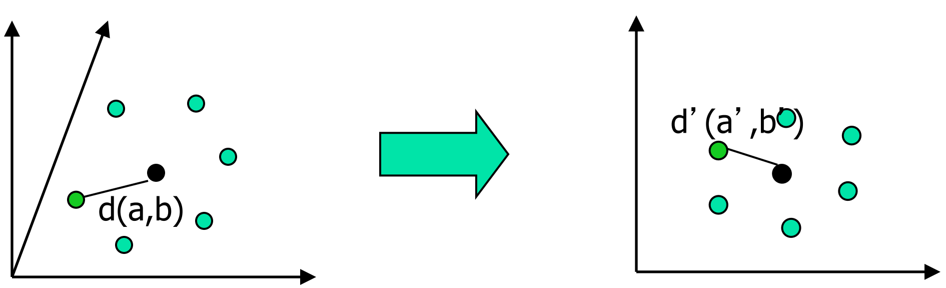
Color Features

- To represent the color of an image compactly, a *color histogram* is used. Colors are partitioned to *k* groups according to their similarity and the *percentage* of each group in the image is measured.
- Images are transformed to *k*-dimensional points and a distance metric (e.g., Euclidean distance) is used to measure the similarity between them.



Using Transformations to Reduce Dimensionality

- In many cases the *embedded* dimensionality of a search problem is much lower than the actual dimensionality
- Some methods apply transformations on the data and approximate them with low-dimensional vectors
- The aim is to reduce dimensionality and at the same time maintain the data characteristics
- If d(a,b) is the distance between two objects a, b in real (high-dimensional) and d'(a',b') is their distance in the transformed low-dimensional space, we want d'(a',b')≤d(a,b).



Multimedia Indexing

Every indexing model must follow a retrieval semantics. The multimedia indexing model must support the similarity queries.

Two types similarity queries:

- range queries
  return documents similar more than a given threshold
- *k* nearest neighbour queries
  return the first *k* most similar documents

Metric Indexing

Feature vectors are indexed according to distances between each other.

As a dissimilarity measure, a distance function $d(O_i,O_j)$ is specified such that the metric axioms are satisfied:

$d(O_i,O_i) = 0$ reflexivity

$d(O_i,O_j) > 0$ positivity

$d(O_i,O_j) = d(O_j,O_i)$ symmetry

$d(O_i,O_k) + d(O_k,O_j) \geq d(O_i,O_j)$ triangular inequality

Metric structures:

Main memory structures: cover-tree, vp-tree, mvp-tree

Disk-based structures: M-tree, Slim-tree

The M-tree

- Inherently dynamic structure
- Disk-oriented (fixed-size nodes)
- Built in a bottom-up fashion
  - Inspired by R-trees and B-trees

- All data in *leaf nodes*
- *Internal nodes*: pointers to subtrees and additional information

M-tree: Internal Node

- Internal node consists of an entry for each subtree
- Each entry consists of:
  - Pivot: *p*
  - Covering *radius* of the sub-tree: r^c
  - Distance from *p* to *parent* pivot p^p: $d(p,p^p)$
  - Pointer to sub-tree: *ptr*

$$\boxed{\boxed{\langle p_1, r_1^c, d(p_1, p^p), ptr_1 \rangle}\ \boxed{\langle p_2, r_2^c, d(p_2, p^p), ptr_2 \rangle} \cdots \boxed{\langle p_m, r_m^c, d(p_m, p^p), ptr_m \rangle}}$$

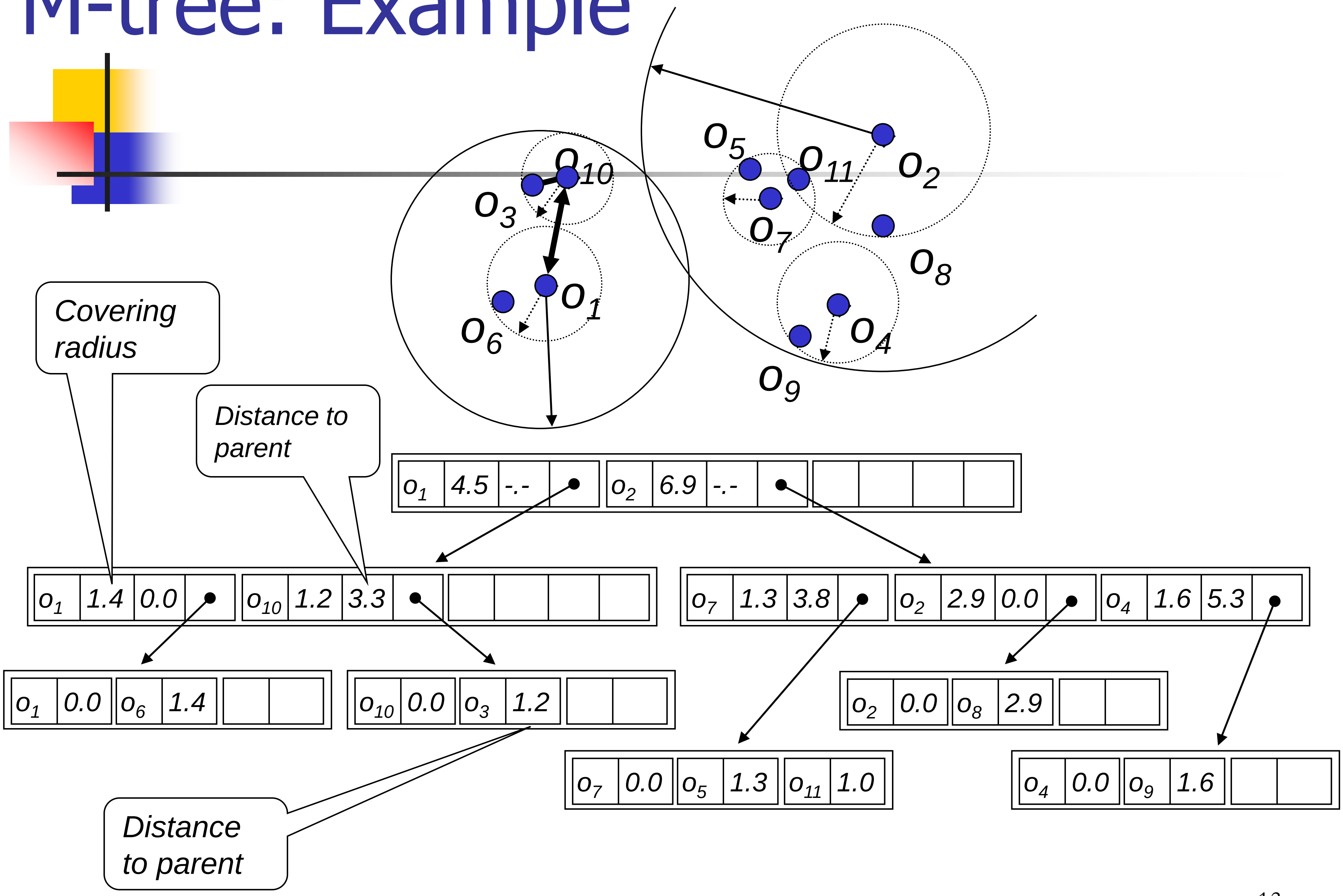
  - All objects in subtree *ptr* are within the distance r^c from *p.*

M-tree: Leaf Node

- leaf node contains **data entries**
- each entry consists of pairs:
  - object (its identifier): o
  - distance between o and its parent pivot: $d(o,o^p)$

$$\boxed{\boxed{\langle o_1, d(o_1, o^p)\rangle}\boxed{\langle o_2, d(o_2, o^p)\rangle}\cdots\boxed{\langle o_m, d(o_m, o^p)\rangle}}$$

M-tree: Example



M-tree: Insert

- Insert a new object o_N:
- recursively descend the tree to locate the *most suitable leaf* for o_N
- in each step enter the subtree with pivot p for which:
  - no enlargement of radius r^c needed, i.e., $d(o_N, p) \leq r^c$
    - in case of ties, choose one with p nearest to o_N
  - minimize the enlargement of r^c

M-tree: Insert (cont.)

- when reaching leaf node N then:
  - if N is not full then store o_N in N
  - else **Split**(N,o_N).

M-tree: Split

Split(N,o_N):

- Let S be the set containing all entries of N and o_N
- Select pivots p_1 and p_2 from S
- Partition S to S_1 and S_2 according to p_1 and p_2
- Store S_1 in N and S_2 in a new allocated node N'
- If N is root
  - Allocate a new root and store entries for p_1, p_2 there
- else (*let N^p and p^p be the parent node and parent pivot of N*)
  - Replace entry p^p with p_1
  - If N^p is full, then **Split**(N^p,p_2)
  - else store p_2 in node N^p

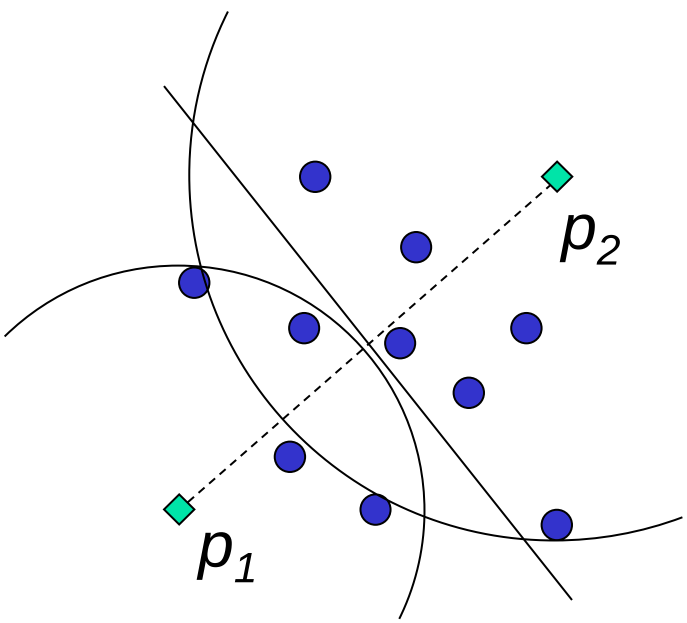
M-tree: Pivot Selection

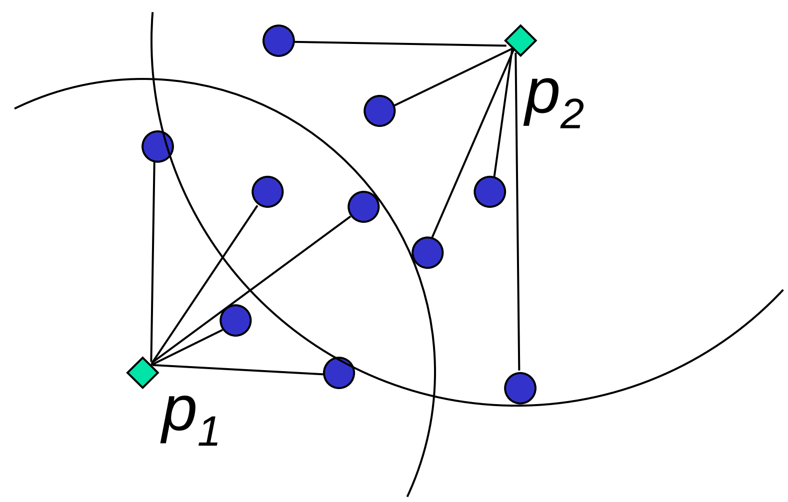
- Several pivots selection policies
  - ***RANDOM*** – select pivots p_1, p_2 randomly
  - ***m_RAD*** – select p_1, p_2 with minimum $(r_1^c + r_2^c)$
  - ***mM_RAD*** – select p_1, p_2 with minimum $max(r_1^c, r_2^c)$
  - ***M_LB_DIST*** – let $p_1 = p^p$ and $p_2 = o_i$ / $\max_i \{ d(o_i,p^p) \}$
    - Uses the pre-computed distances only
- Two versions (for most of the policies):
  - **Confirmed** – reuse the original pivot p^p and select only one
  - **Unconfirmed** – select two pivots (notation: ***RANDOM_2***)
- In the following, the ***mM_RAD_2*** policy is used.

M-tree: Split Policy

- Partition S to S_1 and S_2 according to p_1 and p_2

- Unbalanced
  - Generalized hyperplane
- Balanced
  - Larger covering radii
  - Worse than unbalanced one

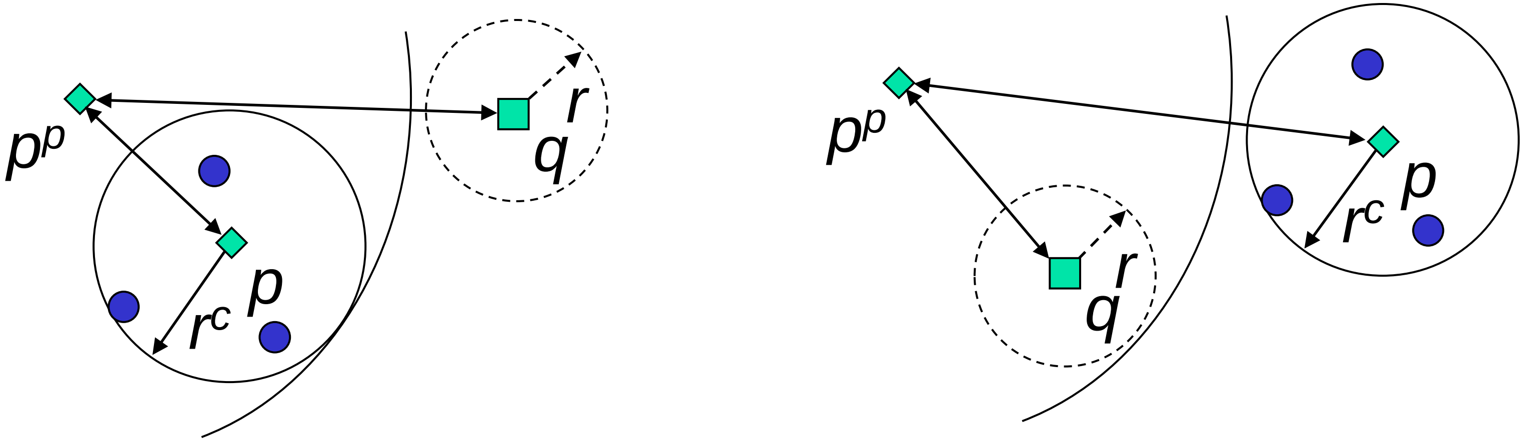




M-tree: Range Search

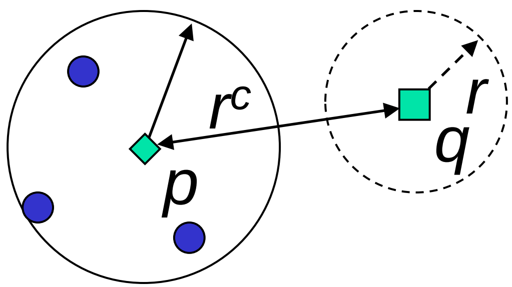
Given $R(q,r)$:

- Traverse the tree in a depth-first manner
- In an internal node, for each entry $\langle p, r^c, d(p,p^p), ptr \rangle$
  - Prune the subtree if $|d(q,p^p) - d(p,p^p)| - r^c > r$
  - Application of the pivot-pivot constraint



M-tree: Range Search (cont.)

- If not discarded, compute $d(q,p)$ and
  - Prune the subtree if $d(q,p) - r^c > r$
  - Application of the range-pivot constraint



- All non-pruned entries are searched recursively.

M-tree: Range Search in Leaf Nodes

- In a leaf node, for each entry $\langle o, d(o, o^p) \rangle$
  - Ignore entry if $|d(q, o^p) - d(o, o^p)| > r$
  - else compute $d(q, o)$ and check $d(q, o) \leq r$
  - Application of the object-pivot constraint

M-tree: *k-NN* Search

Given $k\text{-}NN(q)$:

- Based on a *priority queue* and the pruning mechanisms applied in the range search.
- Priority queue:
  - Stores pointers to sub-trees where qualifying objects can be found.
  - Considering an entry $E=\langle p,r^c,d(p,p^p),ptr\rangle$, the pair $\langle ptr,d_{min}(E)\rangle$ is stored.
  - $d_{min}(E)=max\{\ d(p,q) - r^c,\ 0\ \}$
- Range pruning: instead of fixed radius r, use the distance to the k-th current nearest neighbor.

Text Retrieval (Information retrieval)

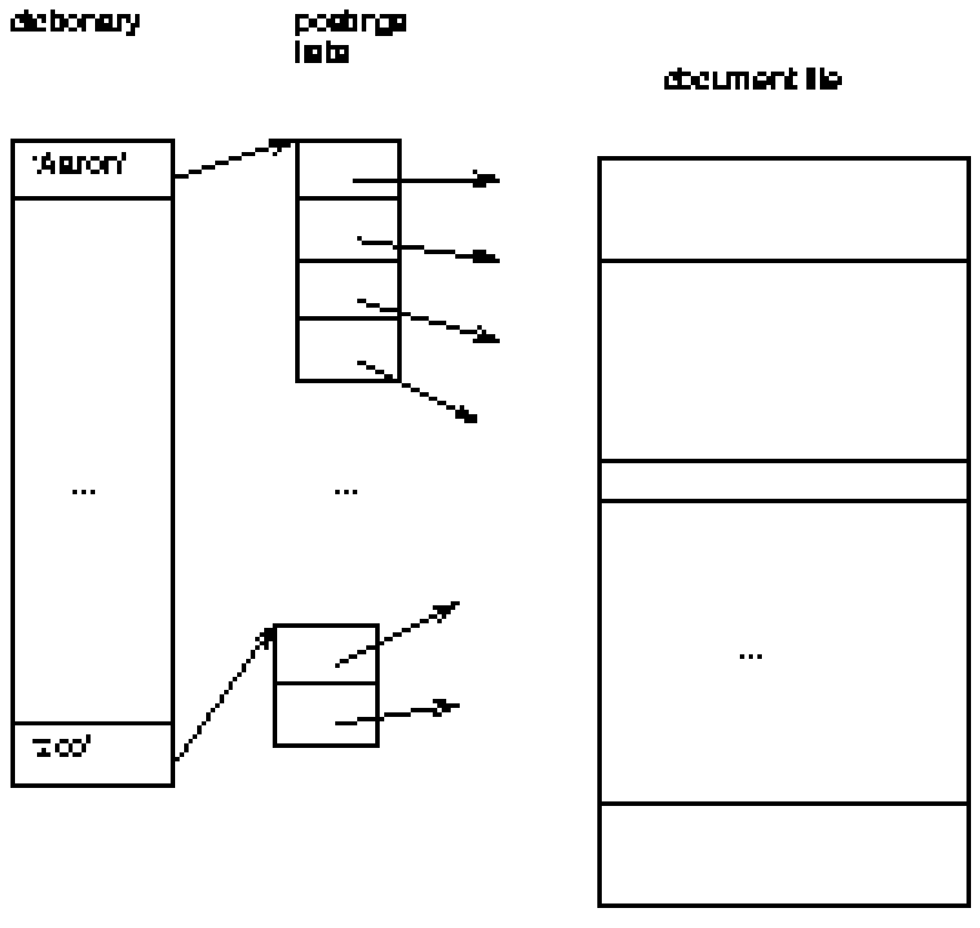
- Given a database of documents, find documents containing "*data*", "*retrieval*"
- Applications:
  - Web
  - law + patent offices
  - digital libraries
  - information filtering

Problem - Motivation

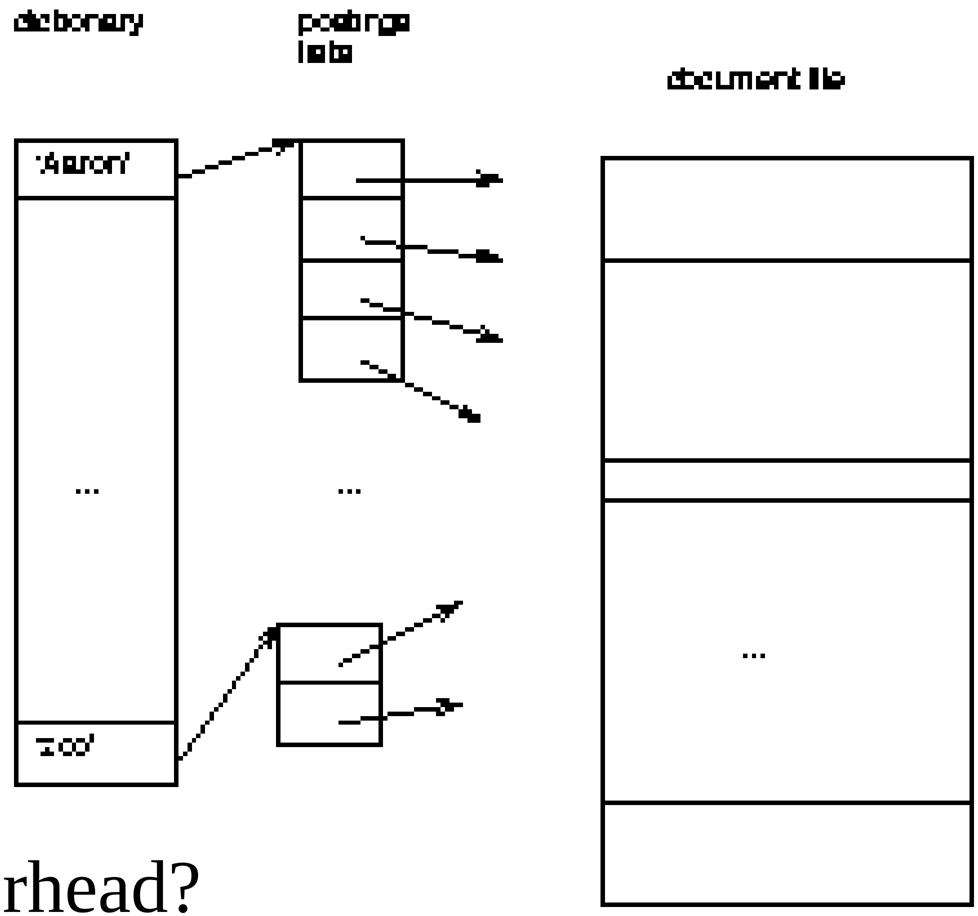
- Types of queries:
  - boolean ('data' AND 'retrieval' AND NOT ...)
  - additional features ('data' ADJACENT 'retrieval')
  - keyword queries ('data', 'retrieval')

- How to search a large collection of documents?

Text – Inverted Files



Text – Inverted Files



Q: space overhead?

A: mainly, the postings lists

Text – Inverted Files

- how to organize dictionary?
- stemming – Y/N?
  - Keep only the root of each word ex. inverted, inversion → invert
- insertions?

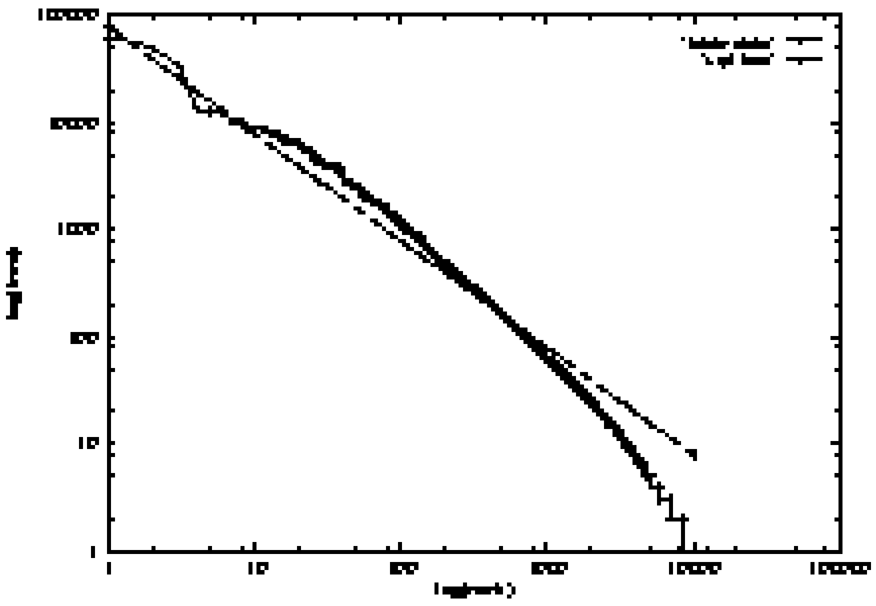
Text – Inverted Files

- how to organize dictionary?
  - B-tree, hashing, TRIEs, PATRICIA trees, ...
- stemming – Y/N?
- insertions?

Text – Inverted Files

- postings list – more Zipf distr.: eg., rank-frequency plot of 'Bible'

log(freq)



freq ~ 1/rank / ln(1.78V)

log(rank)

Text – Inverted Files

- postings lists
  - Cutting+Pedersen
    - (keep first 4 in B-tree leaves)
  - how to allocate space: [Faloutsos+92]
    - geometric progression
  - compression (Elias codes) [Zobel+] – down to 2% overhead!
  - Conclusions: needs space overhead (2%-300%), but it is the fastest

Text - Detailed outline

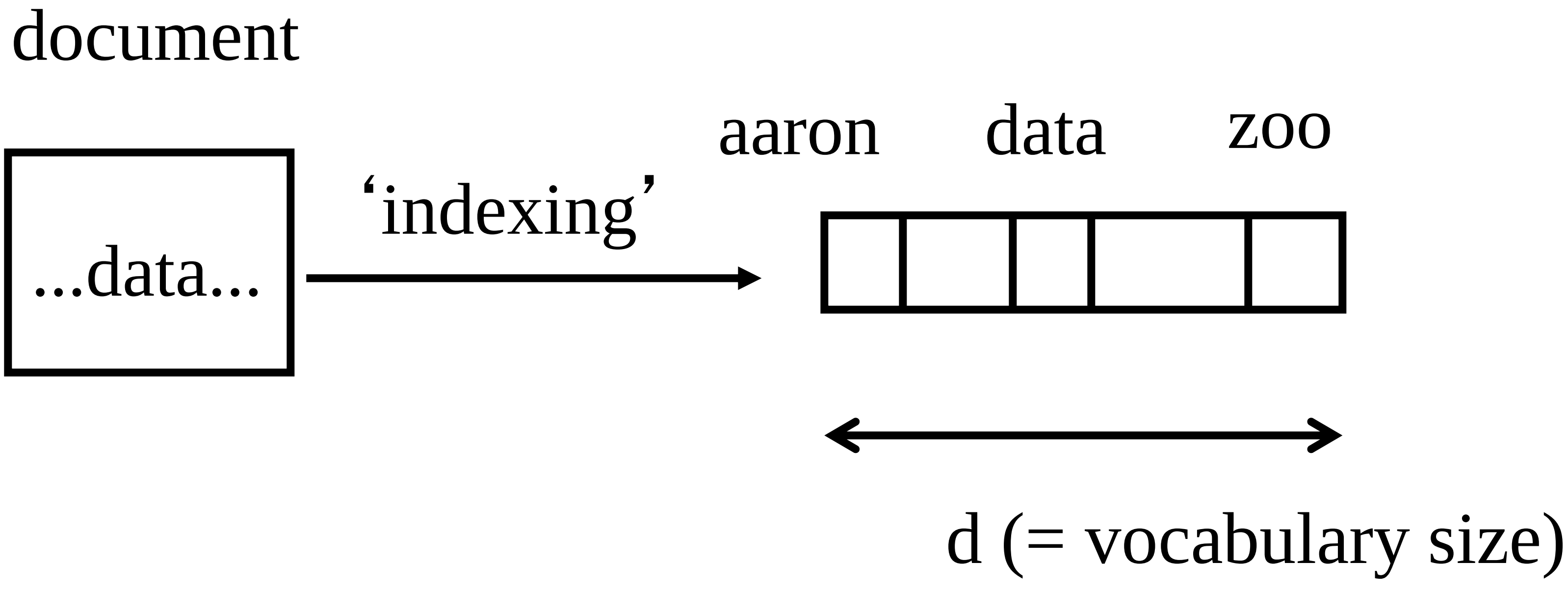
- Text databases
  - problem
  - inversion
  - signature files (a.k.a. Bloom Filters)
  - Vector model and clustering
  - information filtering and LSI

Vector Space Model and Clustering

- Keyword (free-text) queries (vs Boolean)
- each document: -> vector (HOW?)
- each query: -> vector
- search for ‘similar’ vectors

Vector Space Model and Clustering

- main idea: each document is a vector of size d: d is the number of different terms in the **database**



Document Vectors

- Documents are represented as "bags of words"

OR as vectors.

- A vector is like an array of floating points
- Has direction and magnitude
- Each vector holds a place for every term in the collection
- Therefore, most vectors are sparse

Document Vectors
One location for each word.

	nova	galaxy	heat	h'wood	film	role	diet	fur
A	10	5	3					

"Nova" occurs 10 times in text A
"Galaxy" occurs 5 times in text A
"Heat" occurs 3 times in text A
(Blank means 0 occurrences.)

Document Vectors
One location for each word.

“Hollywood” occurs 7 times in text I
“Film” occurs 5 times in text I
“Diet” occurs 1 time in text I
“Fur” occurs 3 times in text I

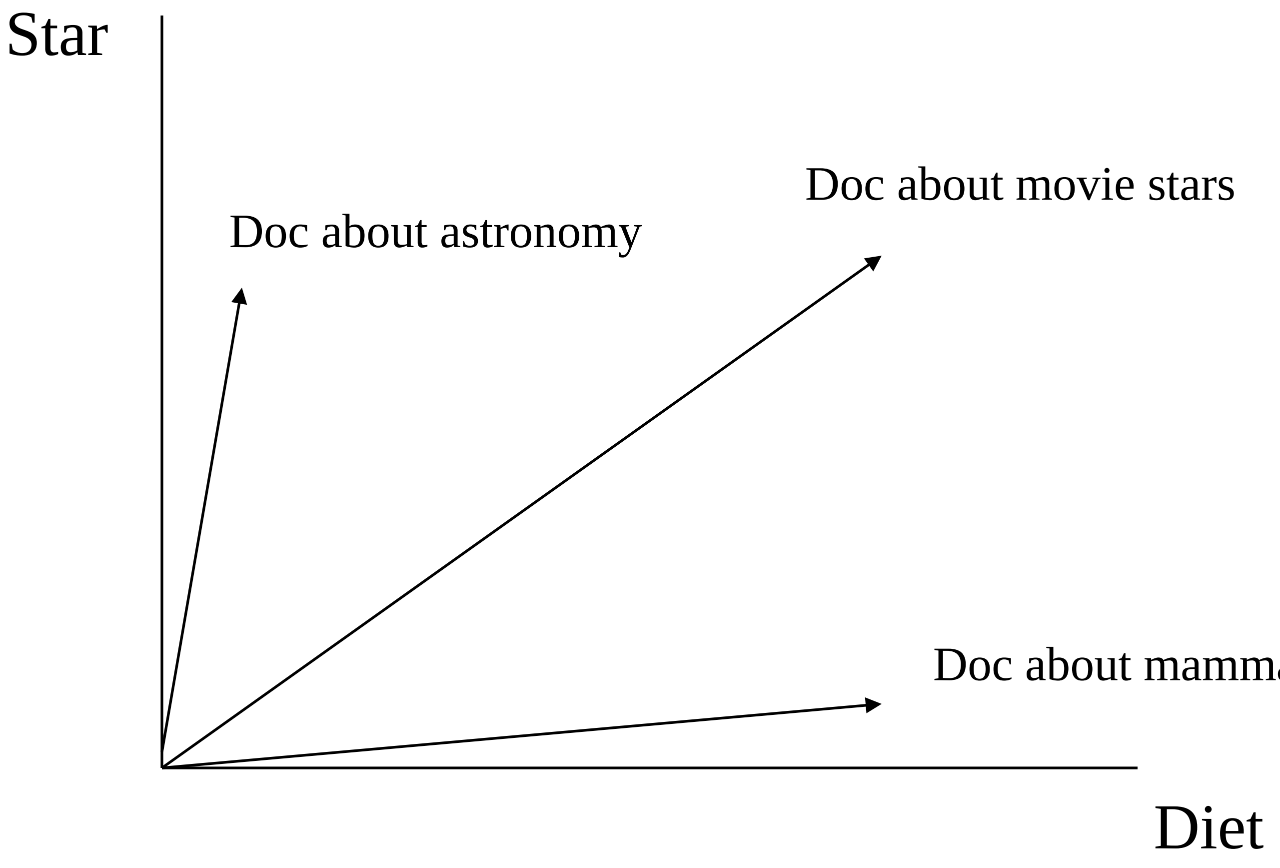
	nova	galaxy	heat	h’wood	film	role	diet	fur
I				7	5		1	3

Document Vectors

Document ids

	nova	galaxy	heat	h'wood	film	role	diet	fur
A	10	5	3					
B	5	10						
C				10	8	7		
D				9	10	5		
E							10	10
F							9	10
G	5		7			9		
H		6	10		2	8		
I				7	5		1	3

We Can Plot the Vectors



Assigning Weights to Terms

- Binary Weights
- Raw term frequency
- tf x idf
  - Recall the Zipf distribution
  - Want to weight terms highly if they are
    - frequent in relevant documents … BUT
    - infrequent in the collection as a whole

Binary Weights

- Only the presence (1) or absence (0) of a term is included in the vector

docs	*t1*	*t2*	*t3*
D1	1	0	1
D2	1	0	0
D3	0	1	1
D4	1	0	0
D5	1	1	1
D6	1	1	0
D7	0	1	0
D8	0	1	0
D9	0	0	1
D10	0	1	1
D11	1	0	1

Raw Term Weights

- The frequency of occurrence for the term in each document is included in the vector

docs	*t1*	*t2*	*t3*
D1	**2**	**0**	**3**
D2	**1**	**0**	**0**
D3	**0**	**4**	**7**
D4	**3**	**0**	**0**
D5	**1**	**6**	**3**
D6	**3**	**5**	**0**
D7	**0**	**8**	**0**
D8	**0**	**10**	**0**
D9	**0**	**0**	**1**
D10	**0**	**3**	**5**
D11	**4**	**0**	**1**

Assigning Weights

- tf x idf measure:
  - term frequency (tf)
  - inverse document frequency (idf) -- a way to deal with the problems of the Zipf distribution
- Goal: assign a tf * idf weight to each term in each document

tf x idf

$$w_{ik} = tf_{ik} * \log(N / n_k)$$

$T_k = \text{term } k$

$tf_{ik} = \text{frequency of term } \mathrm{T}_k \text{ in document } D_i$

$idf_k = \text{inverse document frequency of term } \mathrm{T}_k \text{ in } C$

$N = \text{total number of documents in the collection } C$

$n_k = \text{the number of documents in } C \text{ that contain } \mathrm{T}_k$

$$idf_k = \log\left(\frac{N}{n_k}\right)$$

Inverse Document Frequency

- IDF provides high values for rare words and low values for common words

For a collection of 10000 documents

$$\log\left(\frac{10000}{10000}\right) = 0$$

$$\log\left(\frac{10000}{5000}\right) = 0.301$$

$$\log\left(\frac{10000}{20}\right) = 2.698$$

$$\log\left(\frac{10000}{1}\right) = 4$$

Similarity Measures for document vectors (seen as sets)

$|Q \cap D|$ Simple matching (coordination level match)

$2\dfrac{|Q \cap D|}{|Q| + |D|}$ Dice's Coefficient

$\dfrac{|Q \cap D|}{|Q \cup D|}$ Jaccard's Coefficient

$\dfrac{|Q \cap D|}{|Q|^{1/2} \times |D|^{1/2}}$ Cosine Coefficient

$\dfrac{|Q \cap D|}{\min(|Q|, |D|)}$ Overlap Coefficient

tf x idf normalization

- Normalize the term weights (so longer documents are not unfairly given more weight)
  - normalize usually means force all values to fall within a certain range, usually between 0 and 1, inclusive.

$$w_{ik} = \frac{tf_{ik} \log(N / n_k)}{\sqrt{\sum_{k=1}^{t} (tf_{ik})^2 [\log(N / n_k)]^2}}$$

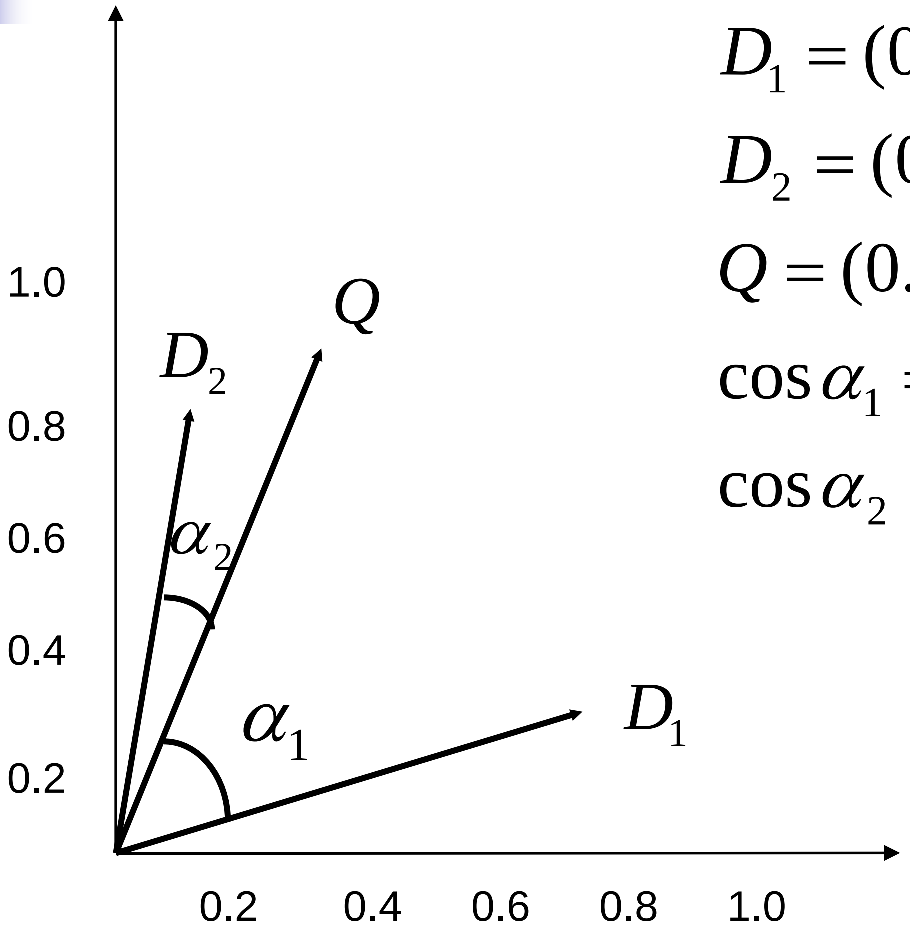
Vector space similarity
(use the weights to compare the documents)

Now, the similarity of two documents is :

$$sim(D_i, D_j) = \sum_{k=1}^{t} w_{ik} * w_{jk}$$

This is also called the cosine, or normalized inner product.

Computing Similarity Scores



$$D_1 = (0.8, 0.3)$$

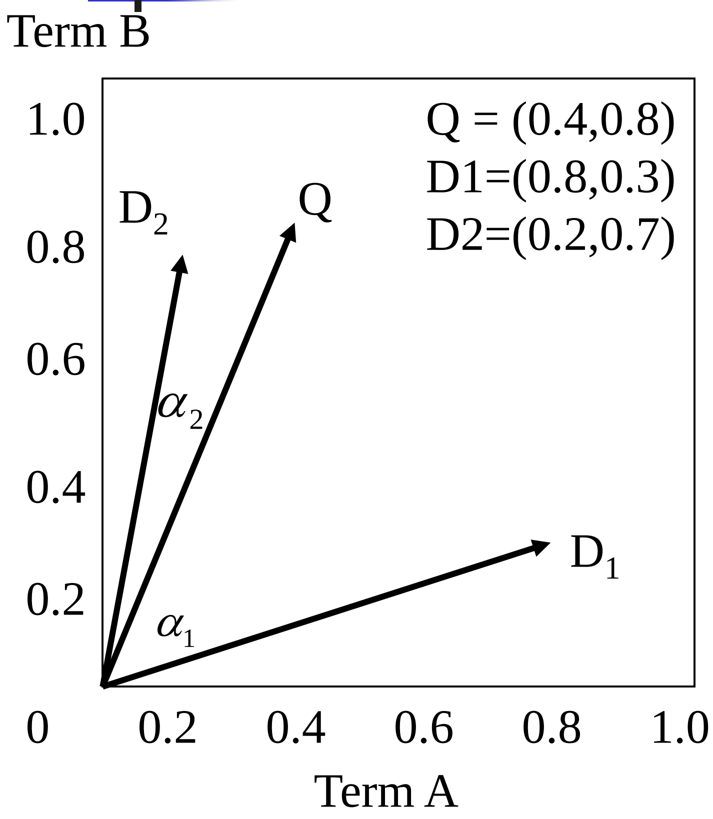
$$D_2 = (0.2, 0.7)$$

$$Q = (0.4, 0.8)$$

$$\cos \alpha_1 = 0.74$$

$$\cos \alpha_2 = 0.98$$

Vector Space with Term Weights and Cosine Matching



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

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

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

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

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

$$sim(Q,D_1)=\frac{.56}{\sqrt{0.58}}=0.74$$

Text - Detailed outline

- Text databases
  - problem
  - full text scanning
  - inversion
  - signature files (a.k.a. Bloom Filters)
  - Vector model and clustering
  - information filtering and LSI

Information Filtering + LSI

- [Foltz+,' 92] Goal:
  - users specify interests (= keywords)
  - system alerts them, on suitable news-documents
- Major contribution: LSI = Latent Semantic Indexing
  - latent ('hidden') concepts

Information Filtering + LSI

Main idea

- map each document into some 'concepts'
- map each term into some 'concepts'

'Concept' :~ a set of terms, with weights, e.g.

  - "data" (0.8), "system" (0.5), "retrieval" (0.6) -> DBMS_concept

Information Filtering + LSI

Pictorially: term-document matrix (BEFORE)

	'data'	'system'	'retrieval'	'lung'	'ear'
TR1	1	1	1		
TR2	1	1	1		
TR3				1	1
TR4				1	1

Information Filtering + LSI

Pictorially: concept-document matrix and...

	'DBMS-concept'	'medical-concept'
TR1	1	
TR2	1	
TR3		1
TR4		1

Information Filtering + LSI

... and concept-term matrix

	'DBMS-concept'	'medical-concept'
data	1	
system	1	
retrieval	1	
lung		1
ear		1

Information Filtering + LSI

Q: How to search, eg., for ‘system’?

Information Filtering + LSI

A: find the corresponding concept(s); and the corresponding documents

	'DBMS-concept'	'medical-concept'
data	1	
system	1	
retrieval	1	
lung		1
ear		1

	'DBMS-concept'	'medical-concept'
TR1	1	
TR2	1	
TR3		1
TR4		1

Information Filtering + LSI

A: find the corresponding concept(s); and the corresponding documents

	'DBMS-concept'	'medical-concept'
data	1	
system	1	
retrieval	1	
lung		1
ear		1

	'DBMS-concept'	'medical-concept'
TR1	1	
TR2	1	
TR3		1
TR4		1

Information Filtering + LSI

Thus it works like an (automatically constructed) thesaurus:

we may retrieve documents that DON'T have the term 'system', but they contain almost everything else ('data', 'retrieval')