CSE 454

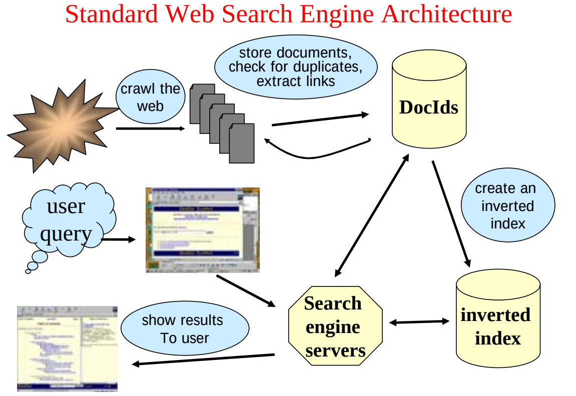
Inverted Indices
(with Compression & LSI)

Project Proto-Idea

- **Search + Tagging + Wiki + Social Network = ?**

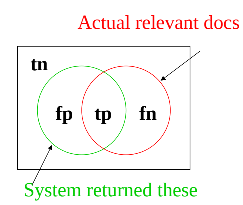
- **Project Reality**
  - Part 1 handed out tomorrow
  - If you want to do something different, let me know by tomorrow

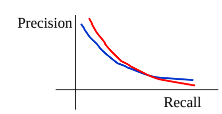
Standard Web Search Engine Architecture



Review: Precision & Recall

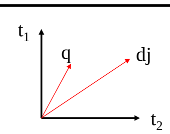
- **Precision** $\frac{tp}{tp + fp}$
  - Proportion of selected items that are correct
- **Recall** $\frac{tp}{tp + fn}$
  - Proportion of target items that were selected
- **Precision-Recall curve**
  - Shows tradeoff



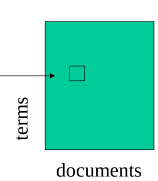


Review

- **Vector Space Representation**
  - Dot Product as Similarity Metric



- **TF-IDF for Computing Weights**
  - $w_{ij} = f(i,j) * \log(N/n_i)$

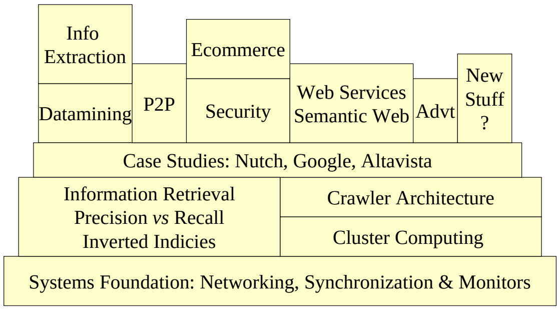


- **But How Process Efficiently?**

Today's Class

- **Efficient query processing**
  - Inverted indicies (creation & query processing)
  - Compression
- **Latent Semantic Indexing (LSI)**

Course Overview



Search Engine Components

- **Spider**
  – Getting the pages
- **Indexing**
  – Storing (e.g. in an inverted file)
- **Query Processing**
  – Booleans, …
- **Ranking**
  – Vector space model, PageRank, anchor text analysis
- **Summaries**
- **Refinement**

Efficient Retrieval

Document-term matrix

	t_1	t_2	. . .	t_j	. . .	t_m	nf
d_1	w_{11}	w_{12}	. . .	w_{1j}	. . .	w_{1m}	$1/\|d_1\|$
d_2	w_{21}	w_{22}	. . .	w_{2j}	. . .	w_{2m}	$1/\|d_2\|$
	. . .	. . .	. . .	. . .	. . .	. . .	
d_i	w_{i1}	w_{i2}	. . .	w_{ij}	. . .	w_{im}	$1/\|d_i\|$
	. . .	. . .	. . .	. . .	. . .	. . .	
d_n	w_{n1}	w_{n2}	. . .	w_{nj}	. . .	w_{nm}	$1/\|d_n\|$

w_{ij} is the weight of term t_j in document d_i

Most w_{ij}'s will be zero.

Naïve Retrieval

Consider query $q = (q_1, q_2, \ldots, q_j, \ldots, q_n)$, $nf = 1/|q|$.

How evaluate q?

(i.e., compute the similarity between q and every document)?

Method 1: Compare q w/ every document directly.

Document data structure:

$d_i : ((t_1, w_{i1}), (t_2, w_{i2}), \ldots, (t_j, w_{ij}), \ldots, (t_m, w_{im}), 1/|d_i|)$

- **Only terms with positive weights are kept.**
- **Terms are in alphabetic order.**

Query data structure:

$q : ((t_1, q_1), (t_2, q_2), \ldots, (t_j, q_j), \ldots, (t_m, q_m), 1/|q|)$

Naïve Retrieval (continued)

Method 1: Compare q with documents directly

initialize all $sim(q, d_i) = 0$;
for each document d_i $(i = 1, \ldots, n)$
{ for each term t_j $(j = 1, \ldots, m)$
if t_j appears in both q and d_i
$sim(q, d_i) \mathrel{+}= q_j * w_{ij}$;
$sim(q, d_i) = sim(q, d_i) \cancel{*(1/|q|)} *(1/|d_i|)$; }
sort documents in descending similarities;
display the top k to the user;

Observation

- Method 1 is not efficient
  – Needs to access most non-zero entries in doc-term matrix.
- Solution: Use Index (Inverted File)
  – Data structure to permit fast searching.
- Like an Index in the back of a text book.
  – Key words --- page numbers.
  – E.g, "Etzioni, 40, 55, 60-63, 89, 220"
  – Lexicon
  – Occurrences

Search Processing (Overview)

1. **Lexicon search**
   – E.g. looking in index to find entry
2. **Retrieval of occurrences**
   – Seeing where term occurs
3. **Manipulation of occurrences**
   – Going to the right page

Inverted Files

POS

1 **A file is a list of words by position**

10 **First entry is the word in position 1 (first word)**

20 **Entry 4562 is the word in position 4562 (4562nd word)**

30 **Last entry is the last word**

36 **An inverted file is a list of positions by word!**

FILE

a (1, 4, 40)
entry (11, 20, 31)
file (2, 38)
list (5, 41)
position (9, 16, 26)
positions (44)
word (14, 19, 24, 29, 35, 45)
words (7)
4562 (21, 27)

INVERTED FILE

aka "Index"

Inverted Files for Multiple Documents

LEXICON

WORD	NDOCS	PTR
jezebel	20	
jezer	3	
jezerit	1	
jeziah	1	
jeziel	1	
jezliah	1	
jezoar	1	
jezrahliah	1	
jezreel	39	

OCCURENCE INDEX

"jezebel" occurs
6 times in document 34,
3 times in document 44,
4 times in document 56 . . .

DOCID	OCCUR	POS 1	POS 2	. . .			
34	6	1	118	2087	3922	3981	5002
44	3	215	2291	3010			
56	4	5	22	134	992		

…

566	3	203	245	287

67	1	132

. . .

107	4	322	354	381	405		
232	6	15	195	248	1897	1951	2192
677	1	481					
713	3	42	312	802			

- **One method. Alta Vista uses alternative**

Many Variations Possible

- **Address space (flat, hierarchical)**
- **Record term-position information**
- **Precalculate TF-IDF info**
- **Stored header, font & tag info**
- **Compression strategies**

Using Inverted Files

Several data structures:

1. For each term t_j, create a list (**inverted file list**) that contains all document ids that have t_j.

   $I(t_j) = \{ (d_1, w_{1j}), (d_2, w_{2j}), \ldots, (d_i, w_{ij}), \ldots, (d_n, w_{nj}) \}$

   – **d_i is the document id number of the ith document.**
   – **Weights come from freq of term in doc**
   – **Only entries with non-zero weights should be kept.**

Inverted files continued

More data structures:

2. **Normalization factors** of documents are pre-computed and stored in an array

   nf[i] stores $1/|d_i|$.

Inverted files continued

More data structures:

3. Lexicon: a hash table for all terms in the collection.

	. . . . . .
t_j	pointer to $I(t_j)$
	. . . . . .

- **Inverted file lists are typically stored on disk.**
- **The number of distinct terms is usually very large.**

Digression…

- **Data structures on disk…**
- **Revisiting CSE 326**

Big O notation

Retrieval using Inverted files

initialize all $sim(q, d_i) = 0$;
for each term t_j in q
  { find I(t) using the hash table;
   for each (d_i, w_{ij}) in I(t)
     $sim(q, d_i) += q_j * w_{ij}$; }
for each document d_i
   $sim(q, d_i) = sim(q, d_i) * nf[i]$;
sort documents in descending similarities and
   display the top k to the user;

Observations about Method 2

- If doc d doesn't contain any term of query q, then d won't be considered when evaluating q.
- Only non-zero entries in the columns of the document-term matrix which correspond to query terms … are used to evaluate the query.
- Computes the similarities of multiple documents simultaneously (w.r.t. each query word)

Efficient Retrieval

Example (Method 2): Suppose

q = { (t1, 1), (t3, 1) }, 1/|q| = 0.7071
d1 = { (t1, 2), (t2, 1), (t3, 1) }, nf[1] = 0.4082
d2 = { (t2, 2), (t3, 1), (t4, 1) }, nf[2] = 0.4082
d3 = { (t1, 1), (t3, 1), (t4, 1) }, nf[3] = 0.5774
d4 = { (t1, 2), (t2, 1), (t3, 2), (t4, 2) }, nf[4] = 0.2774
d5 = { (t2, 2), (t4, 1), (t5, 2) }, nf[5] = 0.3333
I(t1) = { (d1, 2), (d3, 1), (d4, 2) }
I(t2) = { (d1, 1), (d2, 2), (d4, 1), (d5, 2) }
I(t3) = { (d1, 1), (d2, 1), (d3, 1), (d4, 2) }
I(t4) = { (d2, 1), (d3, 1), (d4, 1), (d5, 1) }
I(t5) = { (d5, 2) }

Efficient Retrieval

q = { (t1, 1), (t3, 1) }, 1/|q| = 0.7071

d1 = { (t1, 2), (t2, 1), (t3, 1) }, nf[1] = 0.4082
d2 = { (t2, 2), (t3, 1), (t4, 1) }, nf[2] = 0.4082
d3 = { (t1, 1), (t3, 1), (t4, 1) }, nf[3] = 0.5774
d4 = { (t1, 2), (t2, 1), (t3, 2), (t4, 2) }, nf[4] = 0.2774
d5 = { (t2, 2), (t4, 1), (t5, 2) }, nf[5] = 0.3333

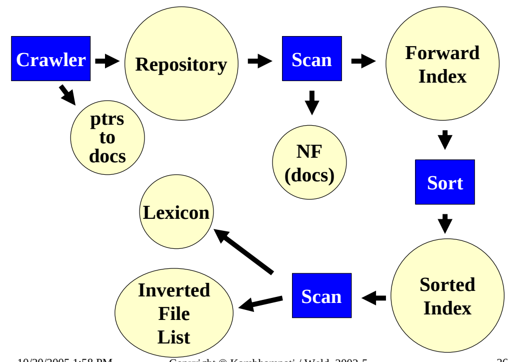
I(t1) = { (d1, 2), (d3, 1), (d4, 2) }
I(t2) = { (d1, 1), (d2, 2), (d4, 1), (d5, 2) }
I(t3) = { (d1, 1), (d2, 1), (d3, 1), (d4, 2) }
I(t4) = { (d2, 1), (d3, 1), (d4, 1), (d5, 1) }
I(t5) = { (d5, 2) }

After t1 is processed:
sim(q, d1) = 2, sim(q, d2) = 0,
sim(q, d3) = 1
sim(q, d4) = 2, sim(q, d5) = 0
After t3 is processed:
sim(q, d1) = 3, sim(q, d2) = 1,
sim(q, d3) = 2
sim(q, d4) = 4, sim(q, d5) = 0
After normalization:
sim(q, d1) = .87, sim(q, d2) = .29,
sim(q, d3) = .82
sim(q, d4) = .78, sim(q, d5) = 0

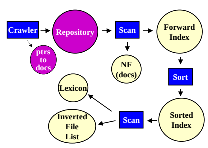
Efficiency versus Flexibility

- Storing computed document weights is good for efficiency, but bad for flexibility.
  - Recomputation needed if TF and IDF formulas change and/or TF and DF information changes.
- Flexibility improved by storing raw TF, DF information, but efficiency suffers.
- A compromise
  - Store pre-computed TF weights of documents.
  - Use IDF weights with query term TF weights instead of document term TF weights.

How Inverted Files are Created

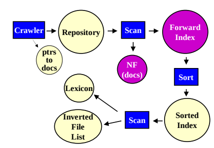


Creating Inverted Files



Repository
- File containing all documents downloaded
- Each doc has unique ID
- Ptr file maps from IDs to start of doc in repository

Creating Inverted Files

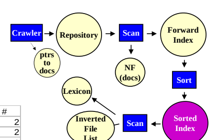


NF
- Length of each document

Forward Index

Term	Doc #	Pos
I	1	1
did	1	2
enact	1	3
julius	1	4
caesar	1	5
I	1	6
was	1	7
killed	1	
i'	1	
the	1	
capitol	1	

Creating Inverted Files



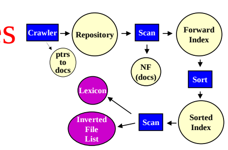
Sorted Index

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1

(positional info as well)

Creating Inverted Files



Lexicon

WORD	NDOCS	PTR
jezebel	20	
jezer	3	
jezerit	1	
jeziah	1	
jeziel	1	
jezliah	1	
jezoar	1	
jezrahliah	1	
jezreel	39	

DOCID	OCCUR	POS 1	POS 2	. . .			
34	6	1	118	2087	3922	3981	5002
44	3	215	2291	3010			
56	4	5	22	134	992		
566	3	203	245	287			
67	1	132					

Inverted File List

. . .

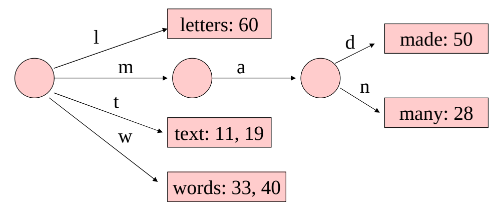
The Lexicon

- **Grows Slowly (Heap's law)**
  - $O(n^{\beta})$ where n=text size; β is constant ~0.4 – 0.6
  - E.g. for 1GB corpus, lexicon = 5Mb
  - Can reduce with stemming (Porter algorithm)
- **Store lexicon in file in lexicographic order**
  - Each entry points to loc in occurrence file (aka inverted file list)

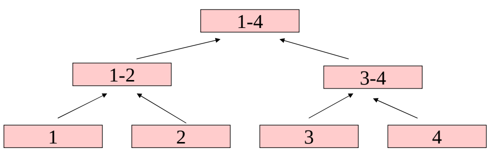
Construction

- **Build Trie (or hash table)**

1 6 9 11 17 19 24 28 33 40 46 50 55 60
This is a text. A text has many words. Words are made from letters.



Memory Too Small?



- **Merging**
  - When word is shared in two lexicons
  - Concatenate occurrence lists
  - O(n1 + n2)
- **Overall complexity**
  - O(n log(n/M)

Stop lists

- **Language-based stop list:**
  - words that bear little meaning
  - 20-500 words
  - http://www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/stop_words
- **Subject-dependent stop lists**
- **Removing stop words**
  - From document
  - From query

From Peter Brusilovsky Univ Pittsburg INFSCI 2140

Stemming

- **Are there different index terms?**
  - retrieve, retrieving, retrieval, retrieved, retrieves…
- **Stemming algorithm:**
  - (retrieve, retrieving, retrieval, retrieved, retrieves) ⇨ retriev
  - Strips prefixes of suffixes (-s, -ed, -ly, -ness)
  - Morphological stemming

Stemming Continued

- Can reduce vocabulary by ~ 1/3
- C, Java, Perl versions, python, c#
  www.tartarus.org/~martin/PorterStemmer
- Criterion for removing a suffix
  - Does "a document is about w_1" mean the same as
  - a "a document about w_2"
- Problems: sand / sander & wand / wander

Compression

- **What Should We Compress?**
  - Repository
  - Lexicon
  - Inv Index
- **What properties do we want?**
  - Compression ratio
  - Compression speed
  - Decompression speed
  - Memory requirements
  - Pattern matching on compressed text
  - Random access

Inverted File Compression

Each inverted list has the form $< f_t ; d_1 , d_2 , d_3 , \dots , d_{f_t} >$

A naïve representation results in a storage overhead of $(f + n) * \lceil \log N \rceil$

This can also be stored as $< f_t ; d_1 , d_2 - d_1 , \dots , d_{f_t} - d_{f_t - 1} >$

Each difference is called a d-gap. Since $\sum (d - gaps) \le N$,

each pointer requires fewer than $\lceil \log N \rceil$ bits.

Trick is encoding …. since worst case ….

Assume d-gap representation for the rest of the talk, unless stated otherwise

Slides adapted from Tapas Kanungo and David Mount, Univ Maryland

Text Compression

Two classes of text compression methods

- Symbolwise (or statistical) methods
  - **Estimate probabilities of symbols - modeling step**
  - **Code one symbol at a time - coding step**
  - **Use shorter code for the most likely symbol**
  - **Usually based on either arithmetic or Huffman coding**
- Dictionary methods
  - **Replace fragments of text with a single code word**
  - **Typically an index to an entry in the dictionary.**
    - **eg: Ziv-Lempel coding: replaces strings of characters with a pointer to a previous occurrence of the string.**
  - **No probability estimates needed**

Symbolwise methods are more suited for coding d-gaps

Classifying d-gap Compression Methods:

- **Global: each list compressed using same model**
  - **non-parameterized**: probability distribution for d-gap sizes is predetermined.
  - **parameterized**: probability distribution is adjusted according to certain parameters of the collection.
- **Local: model is adjusted according to some parameter, like the frequency of the term**
- **By definition, local methods are parameterized.**

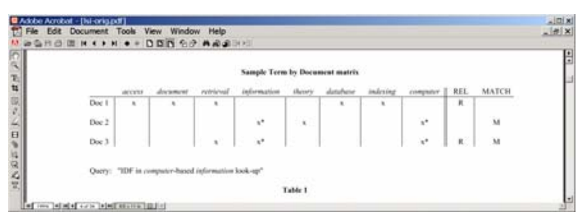
Conclusion

- **Local methods best**
- **Parameterized global models ~ non-parameterized**
  - Pointers not scattered randomly in file
- **In practice, best index compression algorithm is:**
  - Local Bernoulli method (using Golomb coding)
- **Compressed inverted indices usually faster+smaller than**
  - Signature files
  - Bitmaps

Local < Parameterized Global < Non-parameterized Global

Not by much

Motivating the Need for LSI



-- Relevant docs may not have the query terms
→ but may have many "related" terms
-- Irrelevant docs may *have* the query terms
→ but may not have any "related" terms

Terms and Docs as vectors in "factor" space

In addition to doc-doc similarity,
We can compute term-term distance

Document vector

Term vector

	a	b	c	d	e	f	g	h	I
Interface	0	0	1	0	0	0	0	0	0
User	0	1	1	0	1	0	0	0	0
System	2	1	1	0	0	0	0	0	0
Human	1	0	0	1	0	0	0	0	0
Computer	0	1	0	1	0	0	0	0	0
Response	0	1	0	0	1	0	0	0	0
Time	0	1	0	0	1	0	0	0	0
EPS	1	0	1	0	0	0	0	0	0
Survey	0	1	0	0	0	0	0	0	1
Trees	0	0	0	0	0	1	1	1	0
Graph	0	0	0	0	0	0	1	1	1
Minors	0	0	0	0	0	0	0	1	1

If terms are independent, the T-T similarity matrix would be diagonal
=If it is not diagonal, we can use the correlations to add related terms to the query
=But can also ask the question "Are there independent dimensions which define the space where terms & docs are vectors ?"

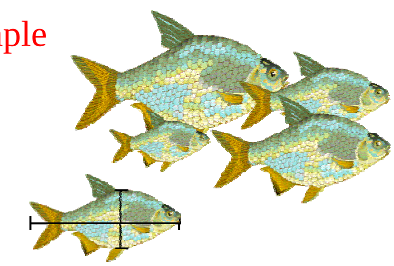
Latent Semantic Indexing

- **Creates modified vector space**
- **Captures transitive co-occurrence information**
  - If docs A & B don't share any words, with each other, but both share lots of words with doc C, then A & B will be considered similar
  - Handles polysemy (adam's apple) & synonymy
- **Simulates query expansion and document clustering (sort of)**

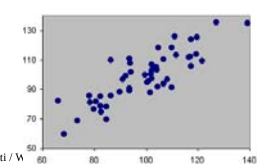
LSI Intuition

- **The key idea is to map documents and queries into a lower dimensional space (i.e., composed of higher level concepts which are in fewer number than the index terms)**
- **Retrieval in this reduced concept space might be superior to retrieval in the space of index terms**

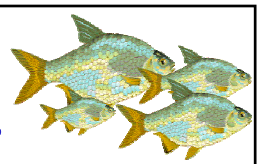
Visual Example



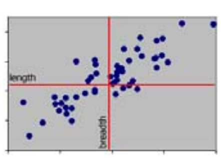
- **Classify Fish**
  - Length
  - Height

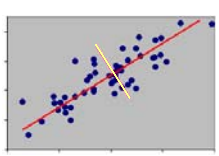


Move Origin



- **To center of centroid**
- **But are these the best axes?**

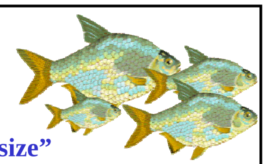




Better if one axis accounts for most data variation

What should we call the red axis?

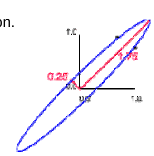
Reduce Dimensions

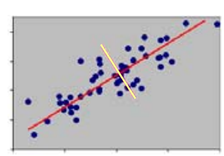


- **What if we *only* consider "size"**

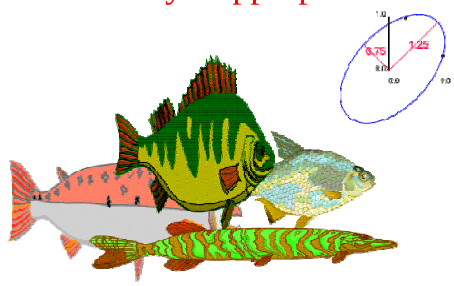
We retain 1.75/2.00 x 100 (87.5%) of the original variation.

Thus, by discarding the yellow axis we lose only 12.5% of the original information.





Not Always Appropriate



Linear Algebra Review

- Let A be a matrix
- X is an Eigenvector of A if
  - A*X= λX
- λ is an Eigenvalue
- Transpose:

$\boxed{A} * \boxed{X} = \lambda \boxed{X}$

$\boxed{A}^T =$

Latent Semantic Indexing Defns

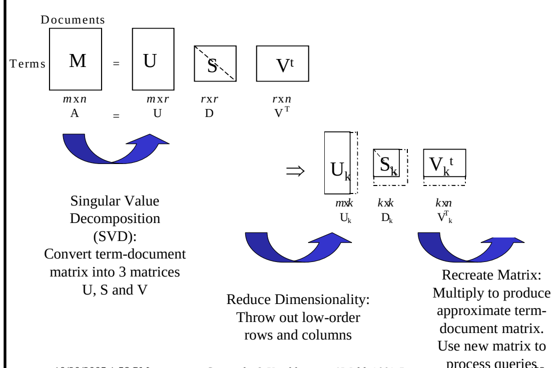
- **Let m be the total number of index terms**
- **Let n be the number of documents**
- **Let [Aij] be a term-document matrix**
  - With m rows and n columns
  - Entries = weights, wij, associated with the pair [ki,dj]
- **The weights can be computed with tf-idf**

Singular Value Decomposition

- **Factor [Aij] matrix into 3 matrices as follows:**
- **(Aij) = (U) (S) (V)^t**
  - (U) is the matrix of eigenvectors derived from (A)(A)^t
  - (V)^t is the matrix of eigenvectors derived from (A)^t(A)
  - (S) is an *r x r* diagonal matrix of singular values
    - r = min(t,n) that is, the rank of (Aij)
    - Singular values are the positive square roots of the eigen values of (A)(A)^t (also (A)^t(A))

U and V are orthogonal matrices

LSI in a Nutshell



Example

term	**ch2**	**ch3**	**ch4**	**ch5**	**ch6**	**ch7**	**ch8**	**ch9**
controllability	1	1	0	0	1	0	0	1
observability	1	0	0	0	1	1	0	1
realization	1	0	1	0	1	0	1	0
feedback	0	1	0	0	0	1	0	0
controller	0	1	0	0	1	1	0	0
observer	0	1	1	0	1	1	0	0
transfer function	0	0	0	0	1	1	0	0
polynomial	0	0	0	0	1	0	1	0
matrices	0	0	0	0	1	0	1	1

U (9x7) =

0.3996	-0.1037	0.5606	-0.3717	-0.3919	-0.3482	0.1029
0.4180	-0.0641	0.4878	0.1566	0.5771	0.1981	-0.1094
0.3464	-0.4422	-0.3997	-0.5142	0.2787	0.0102	-0.2857
0.1888	0.4615	0.0049	-0.0279	-0.2087	0.4193	-0.6629
0.3602	0.3776	-0.0914	0.1596	-0.2045	-0.3701	-0.1023
0.4075	0.3622	-0.3657	-0.2684	-0.0174	0.2711	0.5676
0.2750	0.1667	-0.1303	0.4376	0.3844	-0.3066	0.1230
0.2259	-0.3096	-0.3579	0.3127	-0.2406	-0.3122	-0.2611
0.2958	-0.4232	0.0277	0.4305	-0.3800	0.5114	0.2010

S (7x7) =

3.9901	0	0	0	0	0	0
0	2.2813	0	0	0	0	0
0	0	1.6705	0	0	0	0
0	0	0	1.3522	0	0	0
0	0	0	0	1.1818	0	0
0	0	0	0	0	0.6623	0
0	0	0	0	0	0	0.6487

V (7x8) = T

0.2917	-0.2674	0.3883	-0.5393	0.3926	-0.2112	-0.4505
0.3399	0.4811	0.0649	-0.3760	-0.6959	-0.0421	-0.1462
0.1889	-0.0351	-0.4582	-0.5788	0.2211	0.4247	0.4346
-0.0000	-0.0000	-0.0000	-0.0000	0.0000	-0.0000	0.0000
0.6838	-0.1913	-0.1609	0.2535	0.0050	-0.5229	0.3636
0.4134	0.5716	-0.0566	0.3383	0.4493	0.3198	-0.2839
0.2176	-0.5151	-0.4369	0.1694	-0.2893	0.3161	-0.5330
0.2791	-0.2591	0.6442	0.1593	-0.1648	0.5455	0.2998

This happens to be a rank-7 matrix
-so only 7 dimensions required

Singular values = Sqrt of Eigen values of AA^T

Now to Reduce Dimensions…

- In the matrix (S), select k largest singular values
- Keep the corresponding columns in (U) and $(V)^t$
- The resultant matrix is called $(M)_k$ and is given by
  - $(M)_k = (U)_k (S)_k (V)^t_k$
  - where k, k < r, is the dimensionality of the concept space
- The parameter k should be
  - large enough to allow fitting the characteristics of the data
  - small enough to filter out the non-relevant representational details

The classic over-fitting issue

Formally, this will be the rank-k (2) matrix that is closest to M in the matrix norm sense

U (9x7) =

0.3996	-0.1037	0.5606	-0.3717	-0.3919	-0.3482	0.10
0.4180	-0.0641	0.4878	0.1566	0.5771	0.1981	-0.1094
0.3464	-0.4422	-0.3997	-0.5142	0.2787	0.0102	-0.2857
0.1888	0.4615	0.0049	-0.0279	-0.2087	0.4193	-0.6629
0.3602	0.3776	-0.0914	0.1596	-0.2045	-0.3701	-0.1023
0.4075	0.3622	-0.3657	-0.2684	-0.0174	0.2711	0.5676
0.2750	0.1667	-0.1303	0.4376	0.3844	-0.3066	0.1230
0.2259	-0.3096	-0.3579	0.3127	-0.2406	-0.3122	-0.2611
0.2958	-0.4232	0.0277	0.4305	-0.3800	0.5114	0.2010

S (7x7) =

3.9901	0	0	0	0	0	0
0	2.2813	0	0	0	0	0
0	0	1.6705	0	0	0	0
0	0	0	1.3522	0	0	0
0	0	0	0	1.1818	0	0
0	0	0	0	0	0.6623	0
0	0	0	0	0	0	0.6487

V (7x8) =

0.2917	-0.2674	0.3883	-0.5393	0.3926	-0.2112	-0.4505
0.3399	0.4811	0.0649	-0.3760	-0.6959	-0.0421	-0.1462
0.1889	-0.0351	-0.4582	-0.5788	0.2211	0.4247	0.4346
-0.0000	-0.0000	-0.0000	-0.0000	0.0000	-0.0000	0.0000
0.6838	-0.1913	-0.1609	0.2535	0.0050	-0.5229	0.3636
0.4134	0.5716	-0.0566	0.3383	0.4493	0.3198	-0.2839
0.2176	-0.5151	-0.4369	0.1694	-0.2893	0.3161	-0.5330
0.2791	-0.2591	0.6442	0.1593	-0.1648	0.5455	0.2998

U2 (9x2) =

0.3996	-0.1037
0.4180	-0.0641
0.3464	-0.4422
0.1888	0.4615
0.3602	0.3776
0.4075	0.3622
0.2750	0.1667
0.2259	-0.3096
0.2958	-0.4232

S2 (2x2) =

3.9901	0
0	2.2813

V2 (8x2) = ^{T}

0.2917	-0.2674
0.3399	0.4811
0.1889	-0.0351
-0.0000	-0.0000
0.6838	-0.1913
0.4134	0.5716
0.2176	-0.5151
0.2791	-0.2591

U2*S2*V2 will be a 9x8 matrix
That approximates original matrix

What should be the value of k?

$USV^T = U_7S_7V_7^T$

term	ch2	ch3	ch4	ch5	ch6	ch7	ch8	ch9
controllability	1	1	0	0	1	0	0	1
observability	1	0	0	0	1	1	0	1
realization	1	0	1	0	1	0	1	0
feedback	0	1	0	0	0	1	0	0
controller	0	1	0	0	1	1	0	0
observer	0	1	1	0	1	1	0	0
transfer function	0	0	0	0	1	1	0	0
polynomial	0	0	0	0	1	0	1	0
matrices	0	0	0	0	1	0	1	1

K=2

5 components ignored

$U_2S_2V_2^T$

0.52835834 0.42813724 0.30949408 0.0 1.1355368 0.5239192 0.46880865 0.5063048
0.5256176 0.49655432 0.3201918 0.0 1.1684579 0.6059082 0.4382505 0.50338876
0.6729299 -0.015529543 0.29650056 0.0 1.1381099 -0.0052356124 0.82038856 0.6471
-0.0617774 0.76256883 0.10535021 0.0 0.3137232 0.9132189 -0.37838274 -0.06253
0.18889774 0.90294445 0.24125765 0.0 0.81799114 1.0865396 -0.1309748 0.17793834
0.25334513 0.95019233 0.27814224 0.0 0.9537667 1.1444798 -0.071810216 0.2397161
0.21838559 0.55592346 0.19392742 0.0 0.6775683 0.6709899 0.042878807 0.2077163
0.4517898 -0.033422917 0.19505836 0.0 0.75146574 -0.031091988 0.55994695 0.4345
0.60244554 -0.06330189 0.25684044 0.0 0.99175954 -0.06392482 0.75412846 0.5795

K=4

3 components ignored

$U_4S_4V_4^T$

1.1630535 0.67789733 0.17131016 0.0 0.85744447 0.30088043 -0.025483057 1.0295205
0.7278324 0.46981966 -0.1757451 0.0 1.0910251 0.6314231 0.11810507 1.0620605
0.78863835 0.20257005 1.0048805 0.0 1.0692837 -0.20266426 0.9943222 0.106248446
-0.03825318 0.7772852 0.12343567 0.0 0.30284256 0.89999276 -0.3883498 -0.06326774
0.013223715 0.8118903 0.18630582 0.0 0.8972661 1.1681904 -0.027708884 0.11395822
0.21186034 1.0470067 0.76812166 0.0 0.960058 1.0562774 0.1336124 -0.2116417
-0.18525022 0.31930918 -0.048827052 0.0 0.8625925 0.8834896 0.23821498 0.1617572
-0.008397698 -0.23121 0.2242676 0.0 0.9548515 0.14579195 0.89278513 0.1167786
0.30647483 -0.27917668 -0.101294056 0.0 1.1318822 0.13038804 0.83252335 0.70210195

K=6

One component ignored

$U_6S_6V_6^T$

1.0299273 1.0099105 -0.029033005 0.0 0.9757162 0.019038305 0.035608776 0.98004794
0.96788234 -0.010319378 0.030770123 0.0 1.0258299 0.9798115 -0.03772955 1.0212346
0.9165214 -0.026921304 1.0805727 0.0 1.0673982 -0.052518982 0.9011715 0.055653755
-0.19373542 0.9372319 0.1868434 0.0 0.15639876 0.87798584 -0.22921464 0.12886547
-0.029890355 0.9903935 0.028769515 0.0 1.0242295 0.98121595 -0.03527296 0.020075336
0.16586632 1.0537577 0.8398298 0.0 0.8660687 1.1044582 0.19631699 -0.11030859
0.035988174 0.01172187 -0.03462495 0.0 0.9710446 1.0226605 0.04260301 -0.023878671
-0.07636017 -0.024632007 0.07358454 0.0 1.0615499 -0.048087567 0.909685 0.050844945
[illegible] 0.03693092 1.0695065 0.96087193

Coordinate transformation inherent in LSI

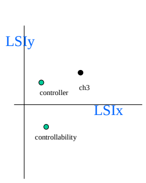
$M = U S V^T$

Mapping of keywords into LSI space is given by US

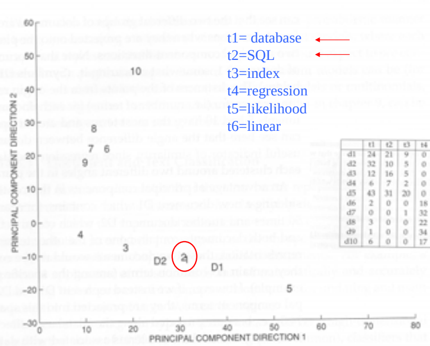
Mapping of a doc d=[w1….wk] into LSI space is given by dUS^{-1}

For k=2, the mapping is:

	LSx	LSy
controllability	1.5944439	-0.2365708
observability	1.6678618	-0.14623132
realization	1.3821706	-1.0087909
feedback	0.7533309	1.05282
controller	1.4372339	0.86141896
observer	1.6259657	0.82628685
Transfer function	1.0972775	0.38029274
polynomial	0.90136355	-0.7062905
matrices	1.1802715	-0.96544623



The base-keywords of
The doc are first mapped
To LSI keywords and
Then differentially weighted
By S^{-1}



t1= database
t2=SQL
t3=index
t4=regression
t5=likelihood
t6=linear

	t1	t2	t3	t4	t5	t6
d1	24	21	9	0	0	3
d2	32	10	5	0	3	0
d3	12	16	5	0	0	0
d4	6	7	2	0	0	0
d5	43	31	20	0	3	0
d6	2	0	0	18	7	16
d7	0	0	1	32	12	0
d8	3	0	0	22	4	2
d9	1	0	0	34	27	25
d10	6	0	0	17	4	23

Figure 14.3 Projected locations of the 10 documents (from table 14.2) in the two dimensional plane spanned by the first two principal components of the document-term matrix M.

Calculating Information Loss

$$\lambda_1, \ldots, \lambda_6 = \{77.4, 69.5, 22.9, 13.5, 12.1, 4.8\}.$$

In agreement with our intuition, most of the variance in the data is captured by the first two principal components. In fact, if we were to retain only these two principal components (as two surrogate terms instead of the six original terms), the fraction of variance that our two-dimensional representation retains is $\boxed{(\lambda_1^2+\lambda_2^2)/\sum_{i=1}^{6}\lambda_i^2 = 0.925}$; i.e., only 7.5% of the information has been lost (in a mean-square sense). If we represent the documents in the new two-dimensional principal component space, the coefficients for each document correspond to the first two columns of the U matrix:

d1	30.8998	-11.4912
d2	30.3131	-10.7801
d3	18.0007	-7.7138
d4	8.3765	-3.5611
d5	52.7057	-20.6051
d6	14.2118	21.8263
d7	10.8052	21.9140
d8	11.5080	28.0101
d9	9.5259	17.7666
d10	19.9219	45.0751

Should clean this up into a slide summarizing the info loss formula

SVD Computation complexity

- **For an m*n matrix SVD computation is**
  - $O(km^2n+k'n^3)$ complexity
    - k=4 and k'=22 for best algorithms
  - Approximate algorithms that exploit the sparsity of M are available (and being developed)

What LSI can do

- **LSI analysis effectively does**
  - Dimensionality reduction
  - Noise reduction
  - Exploitation of redundant data
  - Correlation analysis and Query expansion (with related words)
- **Any one of the individual effects can be achieved with simpler techniques (see thesaurus construction). But LSI does all of them together.**

LSI is *not* the most sophisticated dimensionality reduction technique

- **Dimensionality reduction is a useful technique for any classification/regression problem**
  - Text retrieval can be seen as a classification problem
- **Many other dimensionality reduction techniques**
  - Neural nets, support vector machines etc.
- **Compared to them, LSI is limited because it's *linear***
  - It cannot capture non-linear dependencies between original dimensions
  - E.g.