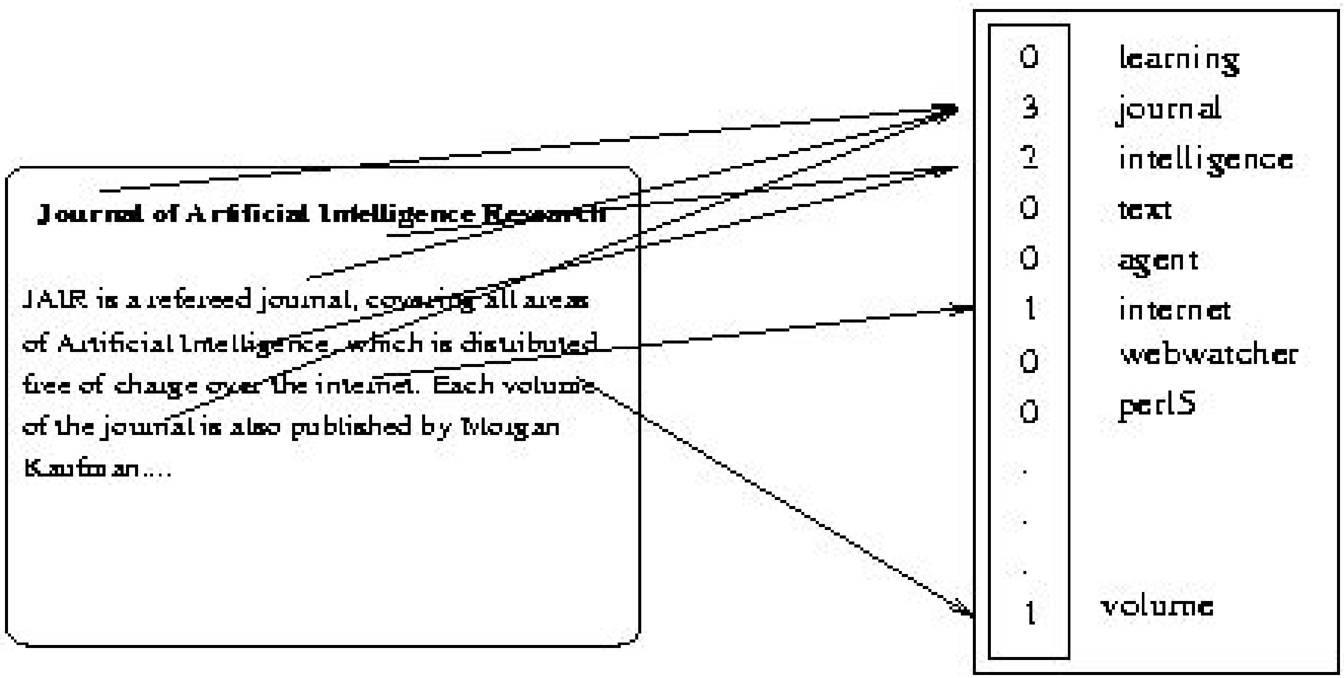
UNSUPERVISED LEARNING 2011

LECTURE :LATENT SEMANTIC INDEXING (LSI)

Based on www.cs.princeton.edu/picasso/mats/Lecture1_jps.ppt
Some slides are due **Eric Xing**

Vector Space Model for Documents

- Represent each document by a high-dimensional vector in the space of words



The Corpora Matrix

X =

	Doc 1	Doc 2	Doc 3	...
Word 1	3	0	0	...
Word 2	0	8	1	...
Word 3	0	1	3	...
Word 4	2	0	0	...
Word 5	12	0	0	...
...	0	0	0	...

t

N

t is the size of the vocabulary (~50,000)
N is the number of documents

Measure of similarity

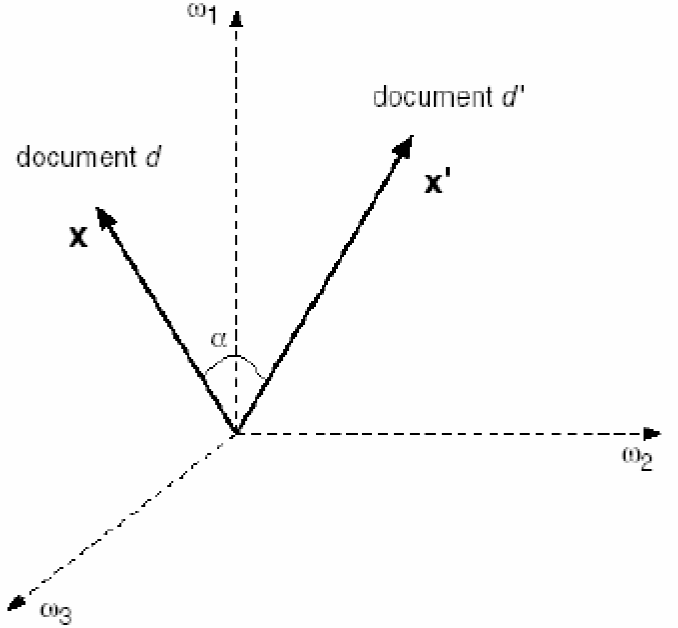


Figure 4.2 Cosine measure of document similarity.

$$sim(\mathrm{x}, \mathrm{x}') = \cos(\alpha) = \frac{\mathrm{x} \cdot \mathrm{x}'}{\|\mathrm{x}\| \, \|\mathrm{x}'\|}$$

Problems

- Looks for literal term matches
  - Terms in queries (esp short ones) don't always capture user's information need well
- Problems:
  - Synonymy: other words with the same meaning
    - Car and automobile

If x' and x do not share words $sim(\mathrm{x}, \mathrm{x}') = \cos(\alpha) = 0$

  - Polysemy: the same word having other meanings
    - Apple (fruit and company)

If x' and x share the word with different meaning:

$sim(\mathrm{x}, \mathrm{x}')_k$ is high.

Latent Semantic Indexing (LSI)

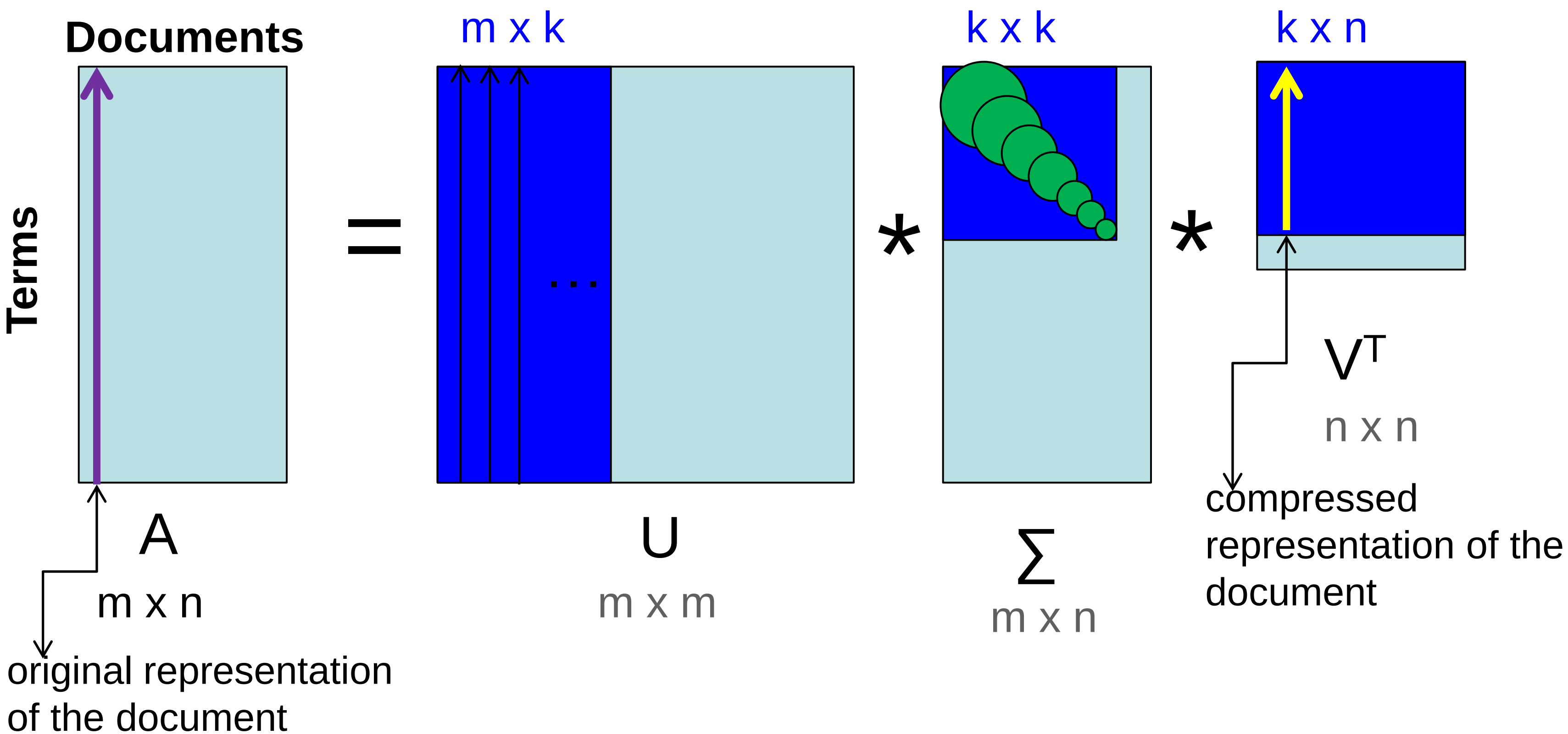
- Uses statistically derived conceptual indices instead of individual words for retrieval
- Assumes that there is some underlying or *latent* structure in word usage that is obscured by variability in word choice
- Key idea: instead of representing documents and queries as vectors in a t-dim space of terms
  - Represent them (and terms themselves) as vectors in a lower-dimensional space whose axes are concepts that effectively group together similar words
  - These axes are the Principal Components from PCA

Example

- Suppose we have keywords
  - Car, automobile, driver, elephant
- We want queries on car to also get docs about drivers and automobiles, but not about elephants
  - What if we could discover that the car, automobile and driver directions are strongly correlated, but elephant is not
  - How? Via correlations observed through documents
  - 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
  - E.g A has cars and drivers, B has automobiles and drivers
- When you scrunch down dimensions, small differences (noise) gets glossed over, and you get desired behavior

LSI

Take the vector representation in the original term space and transform it to new space.



Singular Value Decomposition

For an $m \times n$ matrix $\mathbf{A}$ of rank r there exists a factorization (Singular Value Decomposition = **SVD**) as follows:

$$A = U \Sigma V^T$$

$m \times m$ $m \times n$ V is $n \times n$

The columns of ***U*** are orthogonal eigenvectors of $\boldsymbol{AA^T}$.

The columns of ***V*** are orthogonal eigenvectors of $\boldsymbol{A^TA}$.

Eigenvalues $\lambda_1 \ldots \lambda_r$ of $\boldsymbol{AA^T}$ are the eigenvalues of $\boldsymbol{A^TA}$.

$$\sigma_i = \sqrt{\lambda_i}$$

$$\Sigma = diag(\sigma_1 \ldots \sigma_r)$$

Singular values.

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

Dimension Reduction in LSI

- 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

Dimension Reduction in LSI

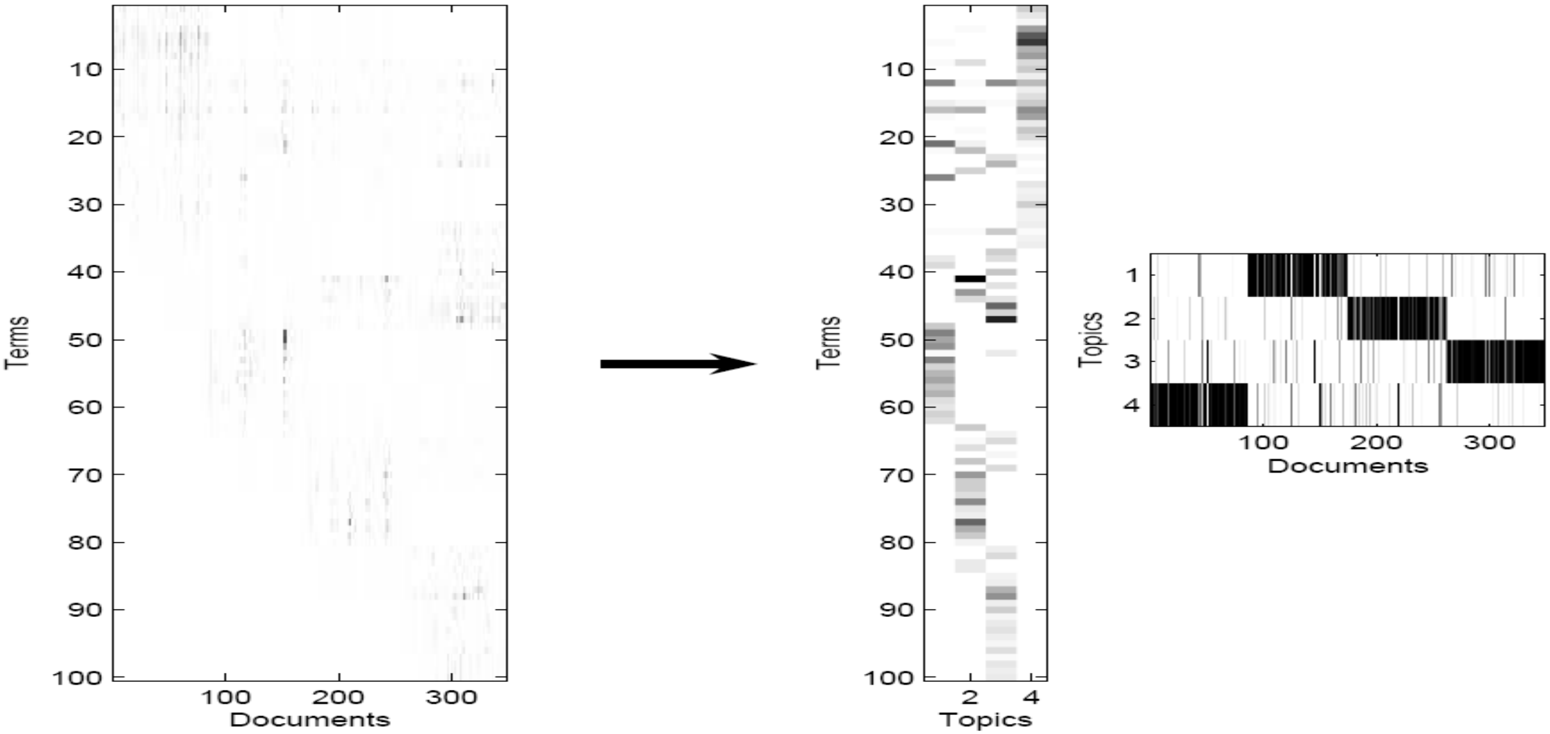
- In matrix $\sum$, select only *k* largest values
- Keep corresponding columns in U and V^T
- Matrix A_k is given by

$$A_k = U_k \sum_k V^T{}_k$$

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 detail

PCs can be viewed as Topics



In the sense of having to find quantities that are not observable directly

LSI: Satisfying a query

- Take the vector representation of the query in the original term space and transform it to concept space

$$d_q = x_q^T U_k \Sigma_k^{-1}$$

places the query pseudo-doc at the centroid of its corresponding terms' locations in the new space

- The document vector (in the concept space)that is nearest in direction to d_q is the best match.

$$\max(d_q V^T)$$

Following the 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

T

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

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

Singular values = Sqrt of Eigen values of AA^T

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

U (9x7) =

0.3996	-0.1037	0.5606	-0.3717	-0.3919	-0.3482	0
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

Querying

To query for *feedback controller*, the query vector would be
q = [0 0 0 1 1 0 0 0 0]' (' indicates transpose),

Let q be the query vector. Then the document-space vector corresponding to q is given by:

$$q'*U2*inv(S2) = Dq$$

Point at the centroid of the query terms' poisitions in the new space.
For the *feedback controller* query vector, the result is:

$$Dq = 0.1376 \quad 0.3678$$

To find the best document match, we compare the Dq vector against all the document vectors in the 2-dimensional $V2$ space. The document vector that is nearest in direction to Dq is the best match. The **cosine values** for the eight document vectors and the query vector are:

-0.3747 0.9671 0.1735 -0.9413 0.0851 0.9642 -0.7265 -0.3805

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) =

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

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
	-0.37	0.967	0.173	-0.94	0.08	0.96	-0.72	-0.38

What LSI can do

- LSI effectively does
  - Dimensionality reduction
  - Noise reduction
  - Exploitation of redundant data
  - Correlation analysis and Query expansion (with related words)
- Some of the individual effects can be achieved with simpler techniques (e.g. thesaurus construction). LSI does them together.
- LSI handles synonymy well, not so much polysemy
- Challenge: SVD is complex to compute ($O(n^3)$)
  - Needs to be updated as new documents are found/updated

LSI Conclusions

- SVD defined basis provide improvements over term matching
  - Interpretation difficult
  - Optimal dimension – open question
  - Variable performance on LARGE collections
  - Supercomputing muscle required

- Probabilistic approaches provide improvements over SVD
  - Clear interpretation of decomposition
  - Optimal dimension – open question
  - High variability of results due to nonlinear optimisation over HUGE parameter space
- Improvements marginal in relation to cost