Document Clustering and Latent Semantic Indexing

CISC689/489-010, Lecture #18

Wednesday, April 22nd

Ben Carterette

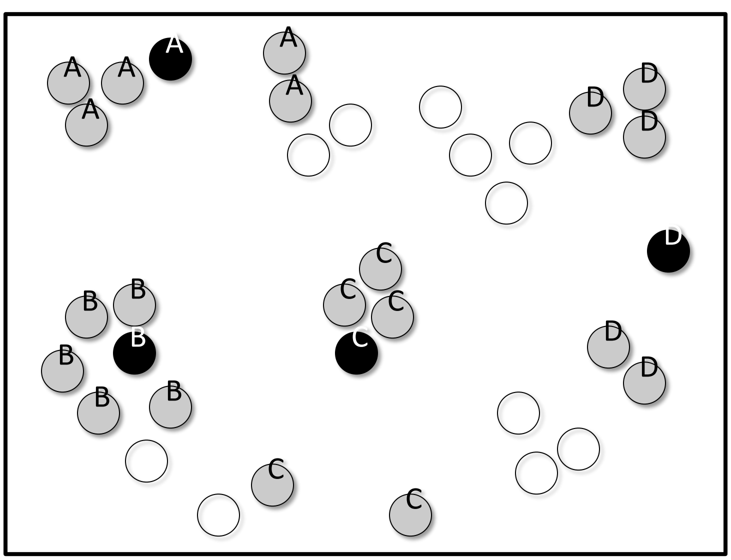
Clustering Review

- Cluster documents according to similarity in feature space
- Types of clustering:
  - Flat or hierarchical
  - "Hard" or "soft"
- Clustering algorithms:
  - K-means: flat, "hard" clusters
  - Agglomerative or divisive hierarchical: hierarchical, softer clusters

K-Nearest Neighbor Clustering

- Alternative idea: fixed-size soft clusters
- *K-nearest neighbor*: for each item *j*, cluster it with the *K* things most similar to it
- K-nearest neighbor clustering forms one cluster per item
  - Clusters overlap
  - Does not necessarily have to cluster everything

5-Nearest Neighbor Clustering



Evaluating Clustering

- Clustering will never be 100% accurate
  - Documents will be placed in clusters they don't belong in
  - Documents will be excluded from clusters they should be part of
  - A natural consequence of using term statistics to represent the information contained in documents
- Like retrieval and classification, clustering effectiveness must be evaluated
- Evaluating clustering is challenging, since it is an ***unsupervised*** learning task

Evaluating Clustering

- If labels exist, can use standard IR metrics, such as precision and recall
  - In this case we are evaluating the ability of our algorithm to discover the "true" latent information

	Class A	Class B	Class C	Class D
Cluster 1	A_1	B_1	C_1	D_1
Cluster 2	A_2	B_2	C_2	D_2
Cluster 3	A_3	B_3	C_3	D_3
Cluster 4	A_4	B_4	C_4	D_4

$$prec_{\text{cluster 1}} = \frac{A_1}{A_1 + B_1 + C_1 + D_1}$$

$$rec_{\text{cluster 1}} = \frac{A_1}{A_1 + A_2 + A_3 + A_4}$$

- This only works if you have some way to "match" clusters to classes
- What if there are fewer or more clusters than classes?

Evaluating Clusters

- "Purity": the ratio between the number of documents from the dominant class in C to the size of C
$$purity(C_i) = \frac{1}{|C_i|} \max_j |X \text{ s.t. } X \in C_i \text{ and } X \in K_j|$$
  - C_i is a cluster; K_j is a class
- Not such a great measure
  - Does not take into account coherence of the class
  - Optimized by making N clusters, one for each document

Evaluating Clusters

- With no labeled data, evaluation is even more difficult
- Best approach:
  - Evaluate the system that the clustering is part of
  - E.g. if clustering is used to aid retrieval, evaluate the cluster-aided retrieval
- What kinds of systems use clustering?

Clusters in IR Systems

- Automatic clustering has several uses:
  - Improving efficiency
  - Improving effectiveness of index
  - Improving effectiveness of retrieval

Improving Efficiency

- Clusters can be a time- and space-saving device:
  - Cluster all documents in the corpus
  - Compare queries to cluster representations rather than document representations
  - Store cluster information in inverted lists rather than document information

$$t_i \rightarrow (\boxed{cf_i}, (c_1, \boxed{tf_{1i}}), \ldots)$$

Cluster frequency

Cluster term frequency

- Important in the 70s and 80s, not so much today

Improving Effectiveness

- Recall the cluster hypothesis:
  - "Closely associated documents tend to be relevant to the same requests"
- We may be able to improve retrieval by finding documents that are closely associated with documents that are likely to be relevant
  - Even if they don't contain query terms
  - E.g. perhaps they contain related terms the user didn't think of ("industry" → "companies", "businesses", "producers", …)

Improving Effectiveness by Ranking Clusters

- As with clustering for efficiency, cluster all documents before indexing
- Store cluster information in inverted lists (but keep document information too)
- When a user enters a query, score the clusters
- Then score documents within the top-scoring clusters

Improving Effectiveness by Ranking Clusters

- Two approaches:
  - Rank the clusters from highest scoring to lowest scoring; within each cluster rank documents from highest scoring to lowest scoring
  - Rank the documents in the K highest-scoring clusters from highest score to lowest score
- Both tend to find relevant documents that are not found with non-clustering methods
  - But also miss relevant documents that are found with non-clustering methods

Scoring Clusters

- A cluster can be scored the same way as a document
  - Vector space model: cosine similarity between query vector and cluster centroid
  - Language model: $P(Q|C) = \prod_{t \in Q} P(t|C) = \prod_{t \in Q} (1 - \alpha_C) \frac{tf_{tC}}{|C|} + \alpha_C \frac{ctf_t}{|G|}$

Smoothing with collection

- Or other ways:
  - $$\begin{aligned} S(C,Q) &= \max_{D_i \in C} S(D_i, Q) \\ S(C,Q) &= \min_{D_i \in C} S(D_i, Q) \\ S(C,Q) &= \frac{1}{|C|} \sum_{D_i \in C} S(D_i, Q) \end{aligned}$$

Using Clusters to Adjust Document

- Language models require background to smooth with
- Clusters provide a background, perhaps more focused than using entire collection
- Smooth document language model with cluster background first, then with collection background
  - P(w | D) – frequency of term in document
  - P(w | C) – frequency of term in all documents in a cluster
  - P(w | G) – frequency of term in all documents in the collection

Smoothing Document Scores

- Basic approach: linear interpolation

$$P(w|D) = \lambda_1 \frac{tf}{|D|} + \lambda_2 \frac{\sum_{D \in C} tf}{\sum_{D \in C} |D|} + \lambda_3 \frac{ctf}{|G|}$$

- Better approach: smooth cluster with collection, then smooth document with both

$$P(w|D) = \lambda \frac{tf}{|D|} + (1 - \lambda) \left(\beta \frac{\sum_{D \in C} tf}{\sum_{D \in C} |D|} + (1 - \beta) \frac{ctf}{|G|} \right)$$

Query-Time Clustering

- Clustering the entire collection takes a long time
- Can only use simple algorithms like K-means
- Instead, cluster the top documents ranked for a query
- Use those clusters to re-score and re-rank the documents

Does it Work?

- Retrieving clusters:

Verdict: no apparent improvement

Collection	First-stage doc retrieval (QL+DM)	Group-average	Single-linkage	Complete-linkage	Centroid
AP (training)	0.2179	0.2161 (t=0.8)	0.2153 (t=0.8)	0.2130 (t=0.8)	0.2164 (t=0.7)
WSJ	0.2958	0.2902 (t=0.8)	0.2911 (t=0.8)	0.2889 (t=0.8)	0.2936 (t=0.8)

- Clustering at index time, use clusters for smoothing

Verdict: improvement over baselines

Collection	Simple Okapi	QL+DM	QL+CBDM	%chg
AP (K=2000)	0.2198	0.2179	0.2326 (+)	+6.73*
WSJ (K=2000)	0.2762	0.2958 (+)	0.3006 (+)	+1.62*
FT (K=2000)	0.2556	0.2610	0.2713 (+)	+3.95*
SJMN (K=2000)	0.2098	0.2032	0.2171 (+)	+6.88*
LA (K=2000)	0.2279	0.2468 (+)	0.2590 (+)	+4.94*
FR (K=1000)	0.2644	0.2875	0.3316	+15.37

Does it Work?

- Clustering at query time, use clusters for smoothing

Collection	Threshold	QL+TDM	QL+CBDM	%chg
AP	0.2	0.2107	0.2223	+5.46*
	0.4	0.2140	0.2247	+5.02*
	0.6	0.2113	0.2211	+4.64*
WSJ	0.2	0.2663	0.2954	+10.92*
	0.4	0.2707	0.3004	+10.95*
	0.6	0.2685	0.2998	+11.65*
FR	0.2	0.2409	0.2935	+21.84
	0.4	0.2265	0.2710	+19.64
	0.6	0.2276	0.2933	+28.84

Verdict: improvement over baselines

Clusters in IR – Summary

- Index-time clustering
  - Saves space in inverted file
  - Build topical hierarchies
  - Retrieve clusters of documents rather than individual documents
- Query-time clustering
  - After query is submitted, cluster results
  - Possibly detect subtopics or different interpretations of query

Latent Semantic Indexing

- Clustering only puts documents together based on term similarity
- Can we do more than that?
  - We'd like synonyms and highly related terms to count for more when calculating document similarity
  - And words that have multiple senses to count for less
- How can we do this?

Linear Algebra Background

- A *vector space* is defined by a set of linearly independent *basis vectors*
  - Linearly independent: no vector can be expressed as a linear combination of other vectors
- Every vector can be expressed as a linear combination of the basis vectors
- Example:

These three vectors form a 3-dimensional vector space

$$B = \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix}$$

These three vectors form a 2-dimensional vector space

$$B = \begin{bmatrix}1\\1\\0\end{bmatrix}, \begin{bmatrix}2\\2\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix}$$

Bases in the Vector Space Model

- When we discussed the VSM, we assumed bases could be formed from terms
  - Document and query vectors are linear combinations of term basis vectors
- But basis vectors have to be linearly independent—do terms satisfy this?
  - Probably not
  - Two terms that always appear together are not independent
  - More insidious example:
    - "bush" appears in documents about landscaping and documents about politics
    - The vector for "bush" may be a linear combination of many vectors for terms related to landscaping and terms related to politics

Bases in the Vector Space Model

- Is there a better way to choose the bases?
- *Semantic concepts*
  - Find a group of topics or concepts that are orthogonal
    - E.g. "landscaping" and "politics" topics are probably linearly independent
  - Each concept forms a basis vector
  - A document vector is a linear combination of concept vectors
- Note: this is not easy to do
  - Solving this problem would probably solve all of AI

Finding Linearly Independent Bases

- There are methods for finding linearly independent basis vectors
- We will apply one method and assume that the bases it produces represent "concepts"

Linear Algebra Background

- Matrix-vector multiplication:

$$Ax = \begin{bmatrix} a_{11} & a_{12} & \dots \\ a_{21} & a_{22} & \dots \\ \dots & & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \dots \\ a_{21}x_1 + a_{22}x_2 + \dots \\ \dots \end{bmatrix} = x_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \dots \end{bmatrix} + x_2 \begin{bmatrix} a_{12} \\ a_{22} \\ \dots \end{bmatrix}$$

  - A *transforms* x
- *Eigenvalues* and *eigenvectors*
  - Given a square matrix A, the eigenvectors of A are the vectors x such that Ax is a scalar multiple of x
    - i.e. the vectors that are only transformed by length, not by direction

$$Ax = \lambda x$$

Every x that satisfies this equation is an eigenvector.
The *eigenvalues* λ show how much A shortens or elongates x.

Eigenvectors and Eiegenvalues

- Example:

$$\begin{bmatrix} 6 & -2 \\ 4 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix} = 2 \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

- [1   2]' is an eigenvector of the matrix, and 2 is an eigenvalue
- How many eigenvalues are there for an n by n matrix?

$$Ax = \lambda x \Leftrightarrow (A - \lambda I)x = 0$$

  - x is nonzero only if *determinant* of A – λI = 0
  - The determinant is a polynomial in λ of degree n
  - Therefore there are at most n eigenvalues

Examples

- Example 1:

$$A = \begin{bmatrix} 30 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{aligned} \text{eigenvalues} &= 30, 20, 1 \\ \text{eigenvectors} &= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{aligned}$$

- Example 2:

$$A = \begin{bmatrix} 0.71 & 0.03 & -0.01 \\ 0.03 & 1.02 & 0.03 \\ -0.01 & 0.03 & 0.70 \end{bmatrix} \qquad \begin{aligned} \text{eigenvalues} &= 1.03, 0.72, 0.69 \\ \text{eigenvectors} &= \begin{bmatrix} -0.09 \\ -0.99 \\ -0.09 \end{bmatrix}, \begin{bmatrix} 0.82 \\ -0.03 \\ -0.57 \end{bmatrix}, \begin{bmatrix} 0.56 \\ -0.12 \\ 0.82 \end{bmatrix} \end{aligned}$$

- Exercise: verify that Ax = λx holds

Eigenvectors and Eigenvalues

- Eigenvectors of symmetric matrices are *linearly independent*
  - Why? If an eigenvector x could be written as a sum of other vectors, then $Ax = \lambda x$ would not be true—x would not be an eigenvector in the first place!
- Therefore eigenvectors of symmetric matrices form a basis
  - Every vector in the space is a linear combination of the eigenvectors

Eigenvectors and Eigenvalues

- Eigenvalues of real-valued matrices are real numbers
- Eigenvalues of *positive semidefinite* matrices are non-negative

Eigen Decompositions

- A square matrix A can be decomposed into a matrix product $A = U\Lambda U^{-1}$ where
  - U is a matrix with eigenvectors of A as columns
  - Λ is a matrix with eigenvalues of A in decreasing order on the diagonal and 0 elsewhere
- A symmetric square matrix A can be decomposed into a matrix product $A = Q\Lambda Q'$
  - Q is a real orthogonal matrix with normalized eigenvectors as columns

So How Does This Apply to IR?

- If we had the right kind of matrix, a decomposition might be able to find orthogonal "concepts" in the document/term data
- Those concepts might be a better basis for a vector space model
- We could take the K most important concepts (corresponding to the K greatest eigenvalues) to reduce the dimensionality of the space
- … but we don't have square symmetric matrices in IR

Eigen Decomp in IR

- Arrange N documents and V terms into a V x N matrix called A, where A_{ij} = term weight of term i in document j
  - This is neither square nor symmetric
- We can compute the following matrix products:
  - AA' = a V x V matrix of document similarities
  - A'A = a N x N matrix of term similarities
  - Note that these are both are square and symmetric—both have eigen decompositions

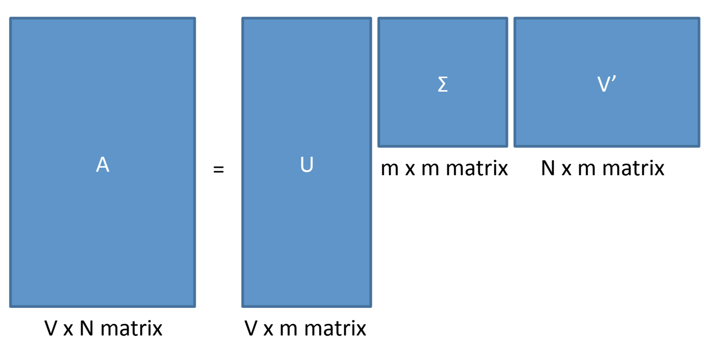
Singular Value Decomposition

- SVD uses AA' and A'A to compute an eigen decomposition of A
  - AA' and A'A have the same number m of eigenvectors
    - $m \leq \min(N, V)$
- Specifically: $A = U\Sigma V'$
  - U = a V x m matrix with eigenvectors of AA' as cols
  - V = a N x m matrix with eigenvectors of A'A as cols
  - Σ = an m x m matrix with the square roots of eigenvalues of AA' on the diagonal
    - Eigenvalues of AA' = eigenvalues of A'A

SVD on the Document-Term Matrix

- Eigenvectors of AA' represent "document concepts"
- Eigenvectors of A'A represent "term concepts"
- Eigenvalues give the relative weight of the concepts
- The occurrence of a term in a document is a linear combination of "term concepts" and "document concepts"

Illustration



Each column of U is a "document concept"
Each row of V' is a "term concept"

$$A_{ij} = \Sigma_{ij} \sum_{k=1}^{m} U_{ik} V_{kj}$$

Example

Technical Memo Example

Titles:

c1: *Human* machine *interface* for Lab ABC *computer* applications
c2: A *survey* of *user* opinion of *computer system response time*
c3: The *EPS user interface* management *system*
c4: *System* and *human system* engineering testing of *EPS*
c5: Relation of *user*-perceived *response time* to error measurement

m1: The generation of random, binary, unordered *trees*
m2: The intersection *graph* of paths in *trees*
m3: *Graph minors* IV: Widths of *trees* and well-quasi-ordering
m4: *Graph minors*: A *survey*

Terms	**Documents**								
	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
user	0	1	1	0	1	0	0	0	0
system	0	1	1	2	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	0	0	1	1	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

From Deerwester, Dumais, Harshman, “Indexing by Latent Semantic Analysis”

Example

$$T_0 = \begin{matrix}
0.22 & -0.11 & 0.29 & -0.41 & -0.11 & -0.34 & 0.52 & -0.06 & -0.41 \\
0.20 & -0.07 & 0.14 & -0.55 & 0.28 & 0.50 & -0.07 & -0.01 & -0.11 \\
0.24 & 0.04 & -0.16 & -0.59 & -0.11 & -0.25 & -0.30 & 0.06 & 0.49 \\
0.40 & 0.06 & -0.34 & 0.10 & 0.33 & 0.38 & 0.00 & 0.00 & 0.01 \\
0.64 & -0.17 & 0.36 & 0.33 & -0.16 & -0.21 & -0.17 & 0.03 & 0.27 \\
0.27 & 0.11 & -0.43 & 0.07 & 0.08 & -0.17 & 0.28 & -0.02 & -0.05 \\
0.27 & 0.11 & -0.43 & 0.07 & 0.08 & -0.17 & 0.28 & -0.02 & -0.05 \\
0.30 & -0.14 & 0.33 & 0.19 & 0.11 & 0.27 & 0.03 & -0.02 & -0.17 \\
0.21 & 0.27 & -0.18 & -0.03 & -0.54 & 0.08 & -0.47 & -0.04 & -0.58 \\
0.01 & 0.49 & 0.23 & 0.03 & 0.59 & -0.39 & -0.29 & 0.25 & -0.23 \\
0.04 & 0.62 & 0.22 & 0.00 & -0.07 & 0.11 & 0.16 & -0.68 & 0.23 \\
0.03 & 0.45 & 0.14 & -0.01 & -0.30 & 0.28 & 0.34 & 0.68 & 0.18
\end{matrix}$$

$$S_0 = \begin{matrix}
3.34 & & & & & & & & \\
 & 2.54 & & & & & & & \\
 & & 2.35 & & & & & & \\
 & & & 1.64 & & & & & \\
 & & & & 1.50 & & & & \\
 & & & & & 1.31 & & & \\
 & & & & & & 0.85 & & \\
 & & & & & & & 0.56 & \\
 & & & & & & & & 0.36
\end{matrix}$$

$$D_0 = \begin{matrix}
0.20 & -0.06 & 0.11 & -0.95 & 0.05 & -0.08 & 0.18 & -0.01 & -0.06 \\
0.61 & 0.17 & -0.50 & -0.03 & -0.21 & -0.26 & -0.43 & 0.05 & 0.24 \\
0.46 & -0.13 & 0.21 & 0.04 & 0.38 & 0.72 & -0.24 & 0.01 & 0.02 \\
0.54 & -0.23 & 0.57 & 0.27 & -0.21 & -0.37 & 0.26 & -0.02 & -0.08 \\
0.28 & 0.11 & -0.51 & 0.15 & 0.33 & 0.03 & 0.67 & -0.06 & -0.26 \\
0.00 & 0.19 & 0.10 & 0.02 & 0.39 & -0.30 & -0.34 & 0.45 & -0.62 \\
0.01 & 0.44 & 0.19 & 0.02 & 0.35 & -0.21 & -0.15 & -0.76 & 0.02 \\
0.02 & 0.62 & 0.25 & 0.01 & 0.15 & 0.00 & 0.25 & 0.45 & 0.52 \\
0.08 & 0.53 & 0.08 & -0.03 & -0.60 & 0.36 & 0.04 & -0.07 & -0.45
\end{matrix}$$

Verify: $T_0 S_0 D_0$ = original document-term matrix

Optimal Dimensionality Reduction

- SVD can also be used for dimensionality reduction
- Reduce Σ to the top-k largest eigenvalues
- For documents and terms, this effectively reduces the dimensionality to the k most important "concepts"
  - Furthermore, the reduction is optimal—it is the best reduction you could possibly do given the document-term matrix

Reducing to k=2 Concepts

X≈ *T* *S* *D '*

T

0.22	-0.11
0.20	-0.07
0.24	0.04
0.40	0.06
0.64	-0.17
0.27	0.11
0.27	0.11
0.30	-0.14
0.21	0.27
0.01	0.49
0.04	0.62
0.03	0.45

S

3.34	
	2.54

D '

0.20	0.61	0.46	0.54	0.28	0.00	0.02	0.02	0.08
-0.06	0.17	-0.13	-0.23	0.11	0.19	0.44	0.62	0.53

Xhihat =

0.16	0.40	0.38	0.47	0.18	-0.05	-0.12	-0.16	-0.09
0.14	0.37	0.33	0.40	0.16	-0.03	-0.07	-0.10	-0.04
0.15	0.51	0.36	0.41	0.24	0.02	0.06	0.09	0.12
0.26	0.84	0.61	0.70	0.39	0.03	0.08	0.12	0.19
0.45	1.23	1.05	1.27	0.56	-0.07	-0.15	-0.21	-0.05
0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
0.22	0.55	0.51	0.63	0.24	-0.07	-0.14	-0.20	-0.11
0.10	0.53	0.23	0.21	0.27	0.14	0.31	0.44	0.42
-0.06	0.23	-0.14	-0.27	0.14	0.24	0.55	0.77	0.66
-0.06	0.34	-0.15	-0.30	0.20	0.31	0.69	0.98	0.85
-0.04	0.25	-0.10	-0.21	0.15	0.22	0.50	0.71	0.62

Latent Semantic Analysis for Retrieval

$X \approx$ T S D'

T	
0.22	-0.11
0.20	-0.07
0.24	0.04
0.40	0.06
0.64	-0.17
0.27	0.11
0.27	0.11
0.30	-0.14
0.21	0.27
0.01	0.49
0.04	0.62
0.03	0.45

S	
3.34	
	2.54

D'								
0.20	0.61	0.46	0.54	0.28	0.00	0.02	0.02	0.08
-0.06	0.17	-0.13	-0.23	0.11	0.19	0.44	0.62	0.53

- Now D contains the new document vectors
  - Each a 2-D vector of “concept weights”
- To process a query Q:
  - Use T to transform it to the 2-D concept space
  - Weight the concepts using S
  - $Q' = Q' \, T \, S^{-1}$
  - Calculate cosine similarity between Q’ and each document

Example

$X \approx$ T S D'

T	
0.22	-0.11
0.20	-0.07
0.24	0.04
0.40	0.06
0.64	-0.17
0.27	0.11
0.27	0.11
0.30	-0.14
0.21	0.27
0.01	0.49
0.04	0.62
0.03	0.45

S	
3.34	
	2.54

D'								
0.20	0.61	0.46	0.54	0.28	0.00	0.02	0.02	0.08
-0.06	0.17	-0.13	-0.23	0.11	0.19	0.44	0.62	0.53

Q = “human computer”
Q’ = [1 0 1 0 0 0 0 0 0 0 0 0]
Q’T = [1*0.22+1*0.24 1*-0.11+1*0.04]
= [0.46 -0.07]
$Q'TS^{-1}$ = [0.46/3.34 -0.07/2.54] = [0.14 -0.03]

LSI: Does it Work?

- Seems to be a little better than standard vector space model
  - Recall is good: intuitively it is retrieving “clusters” of documents related by topic, which improves recall
  - Precision is OK: it can find things that are not really related
- When it does not do well, it is hard to understand what happened
- It takes a very long time to do the SVD
  - In 1990, one day for ~10,000 documents