Dimensionality reduction with LSI

Jean Mark Gawron
Linguistics
San Diego State University
gawron@mail.sdsu.edu
http://www.rohan.sdsu.edu/∼gawron

May 2, 2014

1 Multidimensional scaling and LSI

Latent Semantic Indexing (LSI)[1] is a technique for reducing the dimensionality of matrix representations of relations LSI was introduced in Deerwester et al. (1990) as a strategy for information retrieval (IR). It was posed there as a kind of solution to certain classic problems of document retrieval, because it compressed data into a smaller space by capturing certain generalizations in a way that might query systems more powerful. These aspects are discussed in Secton 5.

Multidimensional scaling (MDS) is a class of data analysis techniques for representing data points in some low dimensional space. MDS has applications in data reduction, data modeling, data analysis, and of course, visualization. LSI can be thought of a special case of MDS. But more than that: For a certain class of MDS problems, the LSI solution is optimal (Bartell et al. 1992).

We will begin by trying to motivate some of the basic ideas from an MDS perspective, because MDS emphasizes preserving similarity relations and understanding how LSI preserves similarity relations is key to understanding its utility in other applications.

The structure of the notes is as follows. First we pose the problem LSI solves as an MDS problem of a specific kind, and then discuss how LSI is an optimal solution. This in turn gives a good first approximation of the conditions under which LSI works best.

[1]This little exposition of LSI contains nothing that is not already said in Deerwester et al. (1990), Bartell et al. (1992), and Manning et al. (2008). The main purpose of this document is to bring the insights of both together in one place.

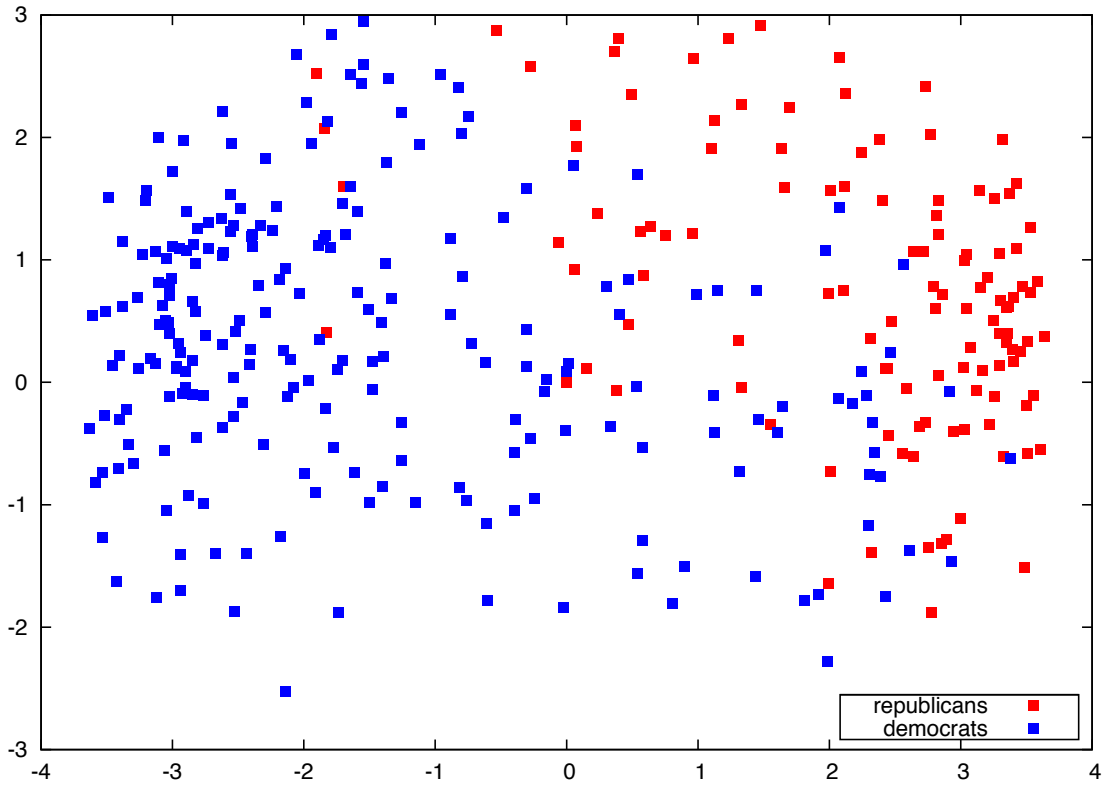


Figure 1: House votes data reduced to 2D by LSI

2 The problem

Let's suppose we have a set of data, for example, a table representing how each member of the United States House of Representatives voted on 16 bills in 1984 (Schlimmer 1987) in data taken from the Congressional Quarterly Almanac, as in Figure 2. The names of the bills are given, and a partial list of the actual votes of all 435 representatives, including abstensions, are given below it. Thus the dataset is a 435x16 table. We may think of this as a representation of each representative, based on 16 votes, and therefore as a representation in a 16 dimensional space. Suppose that, for visualization purposes, we want to put this in a a 2 dimensional space, like the one in Figure 1. How can we make sense of such a task?

This a problem in what is known as Multidimensional scaling, and we can restate it as follows:

1. Define a similarity measure among the objects in the original higher dimensional representation (for us a measure of the similarity between rows in the vote table in Figure 2).

2. Find a lower dimensional representation of the same objects that preserves the similarity relations as well as possible.

Specifically, the kinds of representations of interest are real-numbered vectors (or

	Bill
V1.	handicapped-infants
V2.	water-project-cost-sharing
V3.	adoption-of-the-budget-resolution
V4.	physician-fee-freeze
V5.	el-salvador-aid
V6.	religious-groups-in-schools
V7.	anti-satellite-test-ban
V8.	aid-to-nicaraguan-contras
V9.	mx-missile
V10.	immigration
V11.	synfuels-corporation-cutback
V12.	education-spending
V13.	superfund-right-to-sue
V14.	crime
V15.	duty-free-exports
V16.	export-administration-act-south-africa

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16
1	n	y	n	y	y	y	n	n	n	y	NA	y	y	y	n	y
2	n	y	n	y	y	y	n	n	n	n	n	y	y	y	n	NA
				...												

Figure 2: The Bills and some sample rows from HouseVotes84 data

columns of numbers). So the first thing we want to do is turn our data into numbers. This is fairly easy. We choose 1 for a 'y', 0 for an 'NA', and -1 for an 'n'.

Our two rows of vote data (omitting the member IDs) now look like this:

$$\begin{array}{cccccccccccccccc} -1 & 1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 & 1 & 0 & 1 & 1 & 1 & -1 & 1 \\ -1 & 1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 & 1 & -1 & 0 \end{array} \quad (1)$$

We will call the 16 numbers representing the voting pattern of a representative a **vote vector**. And we will call the table (or **matrix**) representing just the voting records of each house member M. M has 435 rows and 16 columns (we are leaving out the member IDs). We refer to the vote vector of the ith member as $M[i]$, the vote of member i on bill j as $M[i, j]$.

We now propose a way of measuring the **overall** similarity of two vote vectors $M[1]$ and $M[2]$:

$$\text{sim}(M[1],\ M[2]) = \sum_i M[1, i] * M[2, j]$$

What this says is that for any given bill i, we do

$$M[1, i] * M[2, i]$$

This is the product of $M[1]$'s vote on bill i with $M[2]$'s vote on bill i. Given the possible vote values are 0, 1, -1, we have :

	r_2 vote		
r_1 vote	-1	0	1
-1	1	0	-1
0	0	0	0
1	-1	0	1

So whenever two votes agree we get a contribution of +1 to the overall sum, whenever they differ we get a deduction of 1, and when there is an abstention, we get a contribution of 0. The maximum possible similarity of two representatives is 16, the maximum dissimilarity is -16. For the two representatives given in equation (1), the total similarity score is 12, because of 13 agreements, 1 disagreement, and two votes where one or the other representative abstained. This may not be perfect (it might,for example, undervalue the information we get from a shared pattern of abstentions), but overall it makes sense.

The particular operation we used to compute the similarity of two vote vectors is quite important is mathematics and statistics: it is called the **dot product** or **scalar product** or **inner product**. Dot product is written with the symbol $\cdot$ and it is usually thought of as an operation on number sequences, or rows of numbers. The mathematical term is **vector**. In our example we would write:

$$M[1] \cdot M[2] = \sum_j M[1][j] * M[2][j]$$

Read the left hand side as follows: "The dot product of vector $M[1]$ with vector $M[2]$".

The dot product operation has been used to compute similarity measures for many different kinds of data (as well as to compute things that have nothing to do with similarity). For example, if multivariate data is centered (so that the origin is at the center of all the scattered points), then covariance correlation and Pearson correlation, important statistical measures of similarity, are both dot products.[2] Although it does not always make sense to use dot product as the similarity operation, it does cover a lot of cases sensibly, and it is sometimes worthwhile to convert data (by operations such as centering) into a form in which dot product captures similarity intuitions. This is essentially what we did when chose -1,0, and +1 as our vote representations, because it guaranteed that shared vote values would make a positive contribution to the similarity score, and disagreements would always make a negative contribution.

We can now get a little more specific about what we want to do to solve our MDS task. Suppose we consider two pairs of vote vectors v_1 and v_2, and v_3 and v_4 and suppose

$$v_1 \cdot v_2 > v_3 \cdot v_4$$

Here $v_1 \cdot v_2$ stands for the dot product of v_1 with v_2. So this means in our original representation, v_1 is more similar to v_2 than v_3 is to v_4. Now suppose we reduce all these vectors to a 2D representation with some transformation of the data we'll call T. Then we'd like it to be the case that the similarity relations are preserved as much as possible. That is, we'd like it to be the case that:

$$T(v_1) \cdot T(v_2) < T(v_3) \cdot T(v_4)$$

as much as possible.

Now imagine an even bigger table than the one we started with, that represents all the similarity relations in our original data. That is, we want a table S such that

$$S[i, j] = M[i, *] \cdot M[j, *] \tag{2}$$

In terms of our our original data set with 435 house members and 16 votes, this is a much larger 435 x 435 table in which each cell (i, j) represents the similarity in the voting record of house member i with house member j.

Using (2), we can state our problem succinctly.

Dimensionality reduction preserving similarity

We want a table S_2 which is based on M_2, a 2-dimensional representation of the data in M in the following way:

$$S_2[i, j] = M_2[i] \cdot M_2[j].$$

[2]Pearson r_{12} is the cosine of p_1 and p_2, which is the dot product of the centered unit vectors corresponding to p_1 and p_2. The covariance is $\sigma_1 \sigma_2 r_{12}$.

And we want the similarity values in S_2 to match those in S as well as possible.

This reduces the problem to one of measuring the similarity of two tables. S and S_2. Write $S - S_2$ for the table recording discrepancies between S and S_2, obtained by subtracting the values in S_2 from the corresponding values in S. One way to measure the size of this discrepancy is to use something called the **Frobenius norm**: This is simply the square root of the sum of the squares in the discrepancy table. If we let $S - S_2 = X$, then the Frobenius norm of X, written $|| X ||$ is:

$$|| X ||= \sqrt{\sum_{i}^{M} \sum_{j}^{N} X[i,j]^2}$$

What we want, then, is the S_2 that minimizes X. This is no longer a general statement of an MDS problem. It has made certain specific commitments that might not always be appropriate. But they apply to a wide range of data.

A table like S, containing similarity measures for all the pairs in a dataset is called a **similarity matrix**. A similarity matrix might be computed in any number of different ways, but our particular way, based on a dot product operation, has some advantages. For example, one important question about a similarity table is whether it can be consistently be converted to a representation using Euclidean distance. This is important, for example, in a visualization application, where the Euclidean distance is going to correspond to the distance between points in the 2D or 3D image. It turns out that there is a simple way of converting from a dot-product based similarity measures to distances. Rewriting the **cosine theorem** to use a dot product, we have:

$$\text{dist}^2(v_1, v_2) =| v_1 | + | v_2 | -2v_1 \cdot v2$$

Here $| v_1 |$ represents the length of vector v_1. Let's take the special (but fairly frequent) case in which the vectors all have the same length k. Then this becomes:

$$\text{dist}^2(v_1, v_2) = 2(k^2 - v_1 \cdot v_2)$$

This shows that when the dot product (similarity) gets bigger, the distance gets smaller, so in this special case preserving dot product relations preserves distance relations perfectly. Now in fact the vectors in our example do NOT all have the same length (because representatives differ in how many votes they abstained on), but there is not a lot of variation in abstentions, so the distances between points in our image still do a pretty good job of capturing the underlying similarity relation.

Now it turns out that the main subject matter of linear algebra is large tables consisting of ordered sequences of vectors of any size filled with real numbers, like our vote table M; tables like M are called **matrices**, and there is a very compact way of stating the operation needed to produce our similarity table S:

$$S = MM' \tag{3}$$

Here M' denotes the **transpose** of M, that is, the matrix you get by exchanging the rows and columns of M. More precisely:

$$M[i,j] = M'[j,i]$$

So if M is a matrix with i row and j columns, M' is a matrix with j rows and i columns.

In ordinary algebra where variables generally stand for numbers, placing two variables x and y right next to each other as in xy is generally taken to stand for their product. In a linear algebra setting, placing two matrices M and N next to each other, as in MN, generally stands for their **Matrix product**; So (3) says that table S is the matrix product of table M with its transposition M'.

Now the matrix product MN of two tables M and N is defined in terms of dot products of rows of M with columns of N. The (i,j) cell of the MN is the dot product of the ith row of M with the jth column of N

$$MN[i,j] = M[i,*] \cdot N[*,j]$$

Here we use $M[i,*]$ for the ith row of M, and $N[*,j]$ for the j row of N. So we can now apply our definition of matrix product to (3) and rewrite it as:

$$S[i,j] = M[i,*] \cdot M'[*,j] \tag{4}$$

And since M' just exchanges the rows and columns of M, this is:

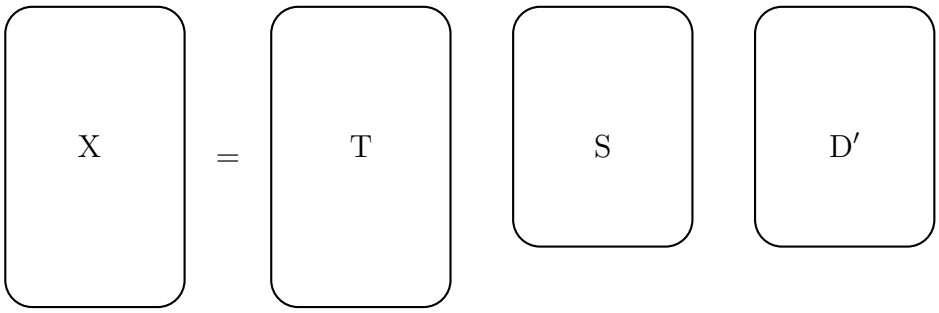
$$S[i,j] = M[i,*] \cdot M[j,*] \tag{5}$$

This is just our original definition of S in (2).

What we have learned thus far: All the similarity information we want to preserve about M can be placed in a big square similarity table S, and there is a compact algebraic expression for defining S, namely (3). But linear algebra can provide a good deal more than compactness in this instance. It can help us find the table S_2 which is based on a 2-dimensional M_2, which minimizes the Frobenius norm with S. That is, it can provide a solution to the dimensionality reduction problem. The term 2-dimensional is actually a little misleading here. The two-dimensional version of M we are going to find will actually have the same number of rows and columns as M, but only two of the columns are **linearly independent**, a technical notion we will not explain here. In the following discussion, rather than speak of a two-dimensional matrix, we will use the more correct terminology and speak of a matrix **of rank 2**, meaning, in this context that only two of the columns are linearly independent.

The solution is based on something called the **singular value decomposition** (SVD) of a matrix. It is a theorem of linear algebra that that any matrix can be

factored into a matrix product of 3 other matrices, which is called its SVD:



(6)

t x d		t x m	m x m	m x d
X	=	T	S	D′

The middle matrix S is what is called a diagonal matrix, because it has non zero values only on the diagonal; these diagonal values are square roots of the so-called **Eigenvalues** of XX', arranged in order from largest to smallest. We will say more about the specific properties of the SVD in the next section. For now we try to describe what it does for us. The SVD of X is closely related to another product. If we replace S with a diagonal matrix S_2 with only the two largest Eigenvalues and 0';s everywhere else, we get an approximation of X we'll call X_2, which has rank 2. An important theorem due to Eckhart and Young (Eckart and Young 1936) shows that this approximation is the closest rank 2 approximation of X, in the sense that the discrepancy between X and X_2 has a smaller Frobenius norm than the discrepancy between X and any other rank 2 matrix. It turns out that the SVD of the similarity matrix of X is very closely related to the SVD of its simarity matrix S:

$$S = XX' = TS^2T'$$

And the solution to our dimensionality reduction problem, the best rank 2 approximation of S, is therefore

$$S_2 = TS_2^2T'.$$

In sum, then, our problem can be formulated in terms of matrix products, and then a theorem of linear algebra provides us with a solution.

In the next section we discuss SVD futher, illustrating some of its properties through another application in which dot product is a sensible similarity measure, document similarity. We conclude this section with a brief introduction to that idea. Consider the case of a term-document matrix represent t terms (or words) and d documents. Suppose there are t possible terms (or words). Documents are represented as vectors of size t, including only 1s and 0s, with a 1 in position k representing the fact that term k has occurred in the document. Thus a set of d documents gives us a large t by d matrix. The rows represent words, the columns documents. The entries are all either 0 or 1. A 1 means the word occurs in the

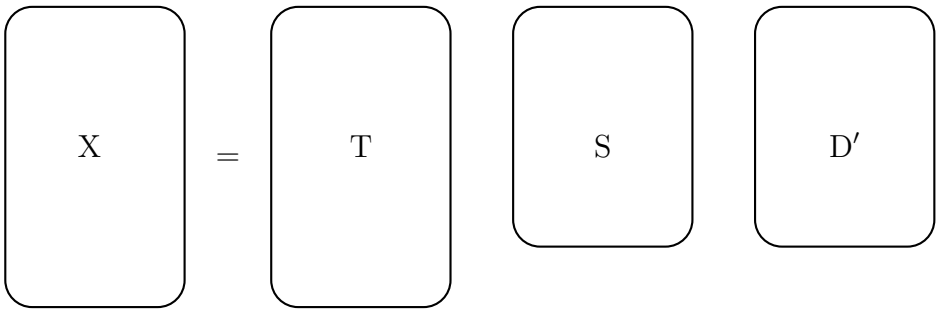
corresponding document. A 0 means it does not. For example,

	*doc*1	*doc*2	*doc*3	*doc*4	*doc*5
tennis	0	1	1	1	1
ball	1	1	1	0	1
racquet	0	0	0	1	0
possum	1	1	0	0	1
burrito	0	1	1	1	1

Taking the dot product of two columns yields the number of words that occur in both documents. If all the documents are of roughly equal size this is a fairly decent measure of document similarity. If as is usually the case, they are not of equal size, a better similarity measure is acheived by **normalizing** the data. Each document vector is divided by its length, yielding a vector of length 1 pointing in the same direction. The dot product of such normalized vectors turns out to yield the cosine of the angle between the vectors. This is a very important measure of similarity which has nice mathematical properties like varying from -1 to 1. For our purposes here the important point is that it is a still special case of dot product, so that our MDS assumptions still apply, and SVD will yield useful results in dimensionality reduction.

3 More about SVD

We return to SVD of a matrix:



(7)

t x d		t x m	m x m	m x d
X	=	T	S	D′

Here are some properties of the matrices in the decomposition:

1. T has orthogonal unit-length column vectors. ($\mathrm{T}'\mathrm{T} = \mathrm{I}$)
2. D has orthogonal unit-length column vectors. ($\mathrm{D}'\mathrm{D} = \mathrm{I}$)
3. S is a diagonal matrix of what are known as "singular values" (more on these

below). It has the following form, where each s_i is a singular value.

$$\begin{bmatrix} \sigma_1 & 0 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & 0 & \dots & 0 \\ \vdots & & \vdots & & & \vdots \\ 0 & \dots & 0 & 0 & \sigma_{m-1} & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_m \end{bmatrix}$$

By convention, $s_1 \geq s_2 \geq \cdots \geq s_m$

4. m is the rank of X ($\leq \min(t, d)$).

Properties 1 and 2 are quite important, so let us take a closer look. What kind of matrix A is such that $\mathrm{A}'\mathrm{A} = I$? The conditions are simple (row i of A is written as A_{i*}, column j as A_{*j}):

1. Each column A_{*j} of A is orthogonal to all the others:

$$\mathrm{A}'_{*j}\mathrm{A}_{*i} = 0 \text{ if } i \neq j$$

2. Each column A_{*j} is a unit vector; i.e.,

$$\mathrm{A}'_{j*}\mathrm{A}_{*j} = 1$$

As an example,

$$\begin{bmatrix} .7071 & 0 & .7071 & 0 & 0 \\ 0 & .5 & 0 & .5 & .7071 \end{bmatrix} \begin{bmatrix} .7071 & 0 \\ 0 & .5 \\ .7071 & 0 \\ 0 & .5 \\ 0 & .7071 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I$$

The result is the identity matrix. I will refer to the property in question (A′ A = I) as *column orthogonality*.[3] As we will see, column orthogonality play a crucial role in securing the dimensionality reduction of LSI.

Column orthogonality follows from the way singular value decompositions are defined. The column vectors of T and D are all *eigenvectors* of XX' and X'X that belong to distinct *eigenvalues*. Such eigen vectors are necessarily orthogonal; by convention the particular eigenvectors chosen are unit vectors. The **singular values** of S are the eigenvalues of those vectors. Column j in T corresponds to eigenvalue S_{jj}.[4]

[3]The term orthogonality is usually reserved for square matrices.

[4]Eigenvalues and eigenvectors are standardly covered in an introductory linear algebra course. If we think of a square matrix M as a vector to vector operator, then the Eigenvectors are the vectors whose direction is unchanged by M. The Eigenvalues are the change in scale of such a vector. So if:

$$(i) \quad \mathrm{M}v = \lambda v$$

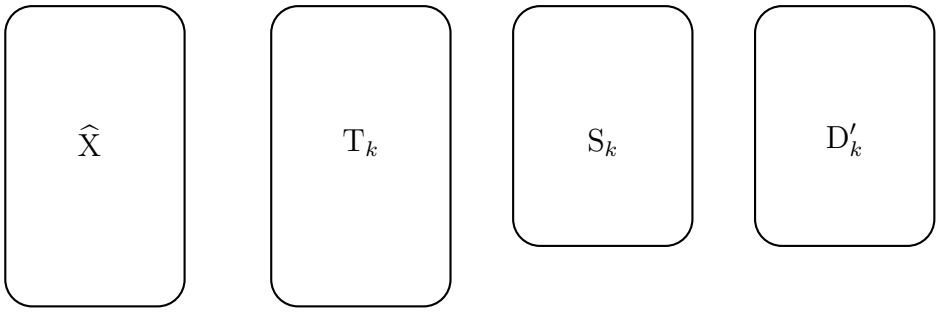
then v is an eigenvector of M and λ is a eigenvalue. The discussion here necessarily omits many important details. For a fuller discussion of SVD and an excellent illustration of some applications, see Strang (1986).

Multiplying the factors in (7) together returns matrix X. Latent Semantic Indexing, first developed in Deerwester et al. (1990) for IR, begins when we explore approximations. The linear combination in (7) can be thought of as a weighted sum of the products of the columns of T with the rows of D' , with the weights supplied by the terms of diagonal matrix S :

$$\mathrm{X} = \sum_{i=0}^{m} \mathrm{S}_{ii} \mathrm{T}_{*i} \mathrm{D}'_{i*} \tag{8}$$

Note that the sum here is a sum of t by d matrices. Matrix-multiplying a single t x 1 column of T by a single 1 x d row of D' yields a t by d matrix, and the appropriate S_{kk} assigns a weight to it. Because of the kind of product each matrix came from, each is a rank 1 matrix which can be thought of as a factor or dimension of X. Thus the SVD is a way of expressing X as a weighted sum of rank 1 factors.

Since the terms of S are arranged in descending magnitude (assumption 3), we can get an approximation of the total sum X by considering only the first k terms. Each successive sum approximates X a little better. Moreover, Equation (8) shows we get exactly the result of summing k terms by preserving the form of (7) and simply zeroing all but the first k terms on the diagonal of S . If we refer to the matrix obtained by zeroing out all but the first k terms on the diagonal of S as S_k, then we have the following picture:



$$\widehat{\mathrm{X}} = \mathrm{T}_k \quad \mathrm{S}_k \quad \mathrm{D}'_k \tag{9}$$

Equation (9) also implements another modification suggested by equation (8): In using S_k to computing the sum in equation (8), the products of rows of T and columns of D' with indices higher than k will be multiplied by 0. The same result for $\widehat{\mathrm{X}}$ may thus be obtained by considering T_k, a version of T with all but the first k columns omitted (therefore a t x k matrix), and D'_k a version of D' with all but the first k rows omitted (therefore a k by d matrix).

Observe that despite this truncation of T and D' the resulting product $\widehat{\mathrm{X}}$ is still a t by d matrix. In general, $\widehat{\mathrm{X}}$ won't have more 0's than X ; it is an approximate, blurred version of X which contains most of what we might call the semantic mass of X .[5] It has rank k, which means it can be expressed as a sum of k rank 1 matrices.

[5]Note that we can compute exactly the "error" of the approximation by zeroing out the *other*

Less grandly put, it is simply what we get when we interrupt the summing at k. What, then, is the point?

As noted in the previous section, The Eckart and Young theorem (Eckart and Young 1936) tells us that $\widehat{\mathrm{X}}$ is the *best* rank k approximation; it has the lowest **Frobenius norm** relative to X of any matrix of rank k. The Frobenius norm gives a measure of matrix similarity.

But in fact the the rank k SVD does something more than just produce a good approximation of the original table; it also provides us with good good k-dimension representations of documents (columns of $\widehat{\mathrm{X}}$) and terms (rows of $\widehat{\mathrm{X}}$). And this is what is of most interest in an application like information retrieval. The next section deals with this aspect of LSI.

4 A k-dimensional representation

In this section we show why the SVD approximation provides us with k-dimensional representations of documents and terms.

This leads to an easy derivation of some equations for representing pesudodocuments and terms in k-space, of immediate applicability in IR.

Assumptions to this point:

1. We represent terms and documents in a large matrix in which there are d terms (rows) and t documents (columns). So the starting representation of a document is a column of X , a vector with t positions (that is, a t-dimensional representation).

2. We replace the input term-document table X with an SVD approximation as described in section 1:

$$\underset{\text{t X d}}{\widehat{\mathrm{X}}} = \underset{\text{t X k}}{\mathrm{T}_k} \quad \underset{\text{k X k}}{\mathrm{S}_k} \quad \underset{\text{k X d}}{\mathrm{D}'_k} \tag{10}$$

3. We compare documents in the new model by looking at dot product of columns of $\widehat{\mathrm{X}}$.

But as the dimension-tracking line of (10) shows, the dimensions of the new "blurred" version of our input table are the same as the ones we started with. So in what sense have we "reduced" dimensions?

---

values in S . Call this $\mathrm{S}_{\overline{k}}$. Then if

$$\overline{\mathrm{X}} = \mathrm{T}_k \mathrm{S}_{\overline{k}} \mathrm{D}'_k$$

it follows that:

$$\mathrm{X} = \overline{\widehat{\mathrm{X}}} + \widehat{\mathrm{X}}$$

But as we will see, the interest of LSI is that the difference between X and $\overline{\widehat{\mathrm{X}}}$ may not just be due to lost information (error), but to the fact that X contains some redundant information. In other words, $\widehat{\mathrm{X}}$ capture some generalizations about X .

One virtue of an SVD decomposition (Deerwester et al. 1990, 14) is that it gives us a way of computing dot products by looking at dot products in a reduced matrix, with r nonzero rows, where r is the rank of $\widehat{\mathrm{X}}$.

Here is how that works. We can always look at dot products between columns of any matrix A by looking at $\mathrm{A}'\mathrm{A}$, a large square matrix whose ij-th cell contains the dot product of A_{*i} with A_{*j}.[6] If we do this with the original X , and substitute in the SVD, the special properties of SVD come into play:

(1)	$\mathrm{X}'\mathrm{X}$	=	$(\mathrm{TSD}')'\mathrm{TSD}'$	
(2)		=	$(\mathrm{D}''(\mathrm{TS})')''\mathrm{TSD}'$	Theorem A
(3)		=	$(\mathrm{D}''(\mathrm{S}'\mathrm{T}')'')''\mathrm{TSD}'$	Theorem A
(4)		=	$\mathrm{DS}'\mathrm{T}'\mathrm{TSD}'$	
(5)		=	$\mathrm{DS}'\mathrm{SD}'$	Column Orthogonality of T

Note that it is orthogonality which licenses the cancellation of T and the consequent simplification in line 5. Line 5 tells us that looking at the square matrix $\mathrm{DS}'\mathrm{SD}'$ will give us all the dot products we want. This result carries over from XX' to $\widehat{\mathrm{X}}\widehat{\mathrm{X}}'$.[7]

Of course we don't actually need to build the big table $\mathrm{DS}'\mathrm{SD}'$ to do document comparisons. Line 5 also tells us that this matrix is the product of DS' with its transpose:

$$(\mathrm{SD}')' = ((\mathrm{D}')'\mathrm{S}')'' = \mathrm{DS}'$$

So we can simply do dot products between rows of DS'. The result carries over to DS'_k, which will produce a matrix with only k nonzero rows. In approximating document similarity for $\widehat{\mathrm{X}}$, then, we are entitled to go from a d x t table to a d x k table. So this is the sense in which our approximation reduces our representation of documents from the t-dimensions we started with to k-dimensions. We can go from doing dot-product comparisons in a t-dimensional space to doing them in a smaller k-dimensional space.

Symmetrically, it turns out to be legitimate to look at $\mathrm{T}_k\mathrm{S}_k$ as our representation of terms, because we can compare rows of $\widehat{\mathrm{X}}$ simply by looking at rows of $\mathrm{T}_k\mathrm{S}_k$ (proof in appendix). And this means we move from a t x d table to a t x k table.

The j-th row of DS is just:

$$\mathrm{DS}_{j*} = \mathrm{S}_{jj} * \mathrm{D}_{j*}$$

So DS just weights the dimensions in D.

The above observations tell us that the matrix we want to do dot products on to get similarity scores in our new 2-dimensional representation is $\mathrm{T}_k\mathrm{S}_k$. When we

[6] This is obviously related to the similarity matrix of the previous section, but it is the similarity maxtrix for columns ($\mathrm{A}'\mathrm{A}$) rather than for rows AA'.

[7] For theorem A, which is a standard linear algebra identity, see the appendix.

look at the $T_k S_k$ we get by doing LSI on the house of representatives data above, our two example voting records come out looking like this:

$$\begin{array}{ll} -3.36152427 & 0.61666413 \\ -3.50447733 & -0.19117607 \end{array}$$

Note that the numbers have changed considerably; values are now real numbers rather than one of the integers $\{-1, 0, 1\}$, and they fall into a greater range than what we started with. In general we may be able to find interpretations for the dimensions in a dimensionality reduced representation (the left to right axis of Figure 1 might be called liberal-conservative), but the numbers will not be meaningful.

5 Some results

The problem addressed by LSI can be viewed as an extension of the kind of problem addressed by *factor analysis*: The goal of factor analysis is to explain the variance of a number of observed variables in terms of a smaller number of unobserved variables called **factors** or **latent variables**. One kind of insight offered by a factor analysis is an estimate of the degree to which variablility is due to common factors. Generally factor analysis limits itself to a few readily **interpretable** dimensions. In LSI, the goal is to discover underlying commonalities among the variables, and interpretability of the dimensions is not a high priority. Therefore, systems with hundreds of dimensions may be explored and have proved useful. In this section, We discuss some of the classic problems of IR addressed by LSI in information retrieval. It has been applied in a number of other areas, including computer vision, memory modeling, linguistics, psychology, and MDS.

What we have seen thus far is that there is a particular way of factoring any matrix into what is called a singular value decomposition. We have also seen that a simple manipulation of this decomposition that involves choosing the k largest singular values gives us an approximation of the original matrix which provides certain guarantees of being the best approximation with a rank-k matrix. Finally we saw that the approximation can meaningfully be viewed as a lower dimensional approximation with fewer variables than we started with.

What we have not discussed yet is how useful this approximation is. The goal of the LSI approximation is to discover underlying commonalities among the variables. That it does in fact discover them is shown by the utility of SVDs in applications such as image-processing. Strang (1986:444) discusses applying LSI-like techniques in matrix representations of satellite images, transmitting only the largest 60 of 1000 singular values, and still recovering significant image structure.

The goals with image processing are rather clear. The issues in information retrieval are a bit more ticklish. Two problems that afflict most approaches to IR are synonymy and polysemy. The problem of synonymy is that the documents relevant to a query containing the word *coriander* may lack the word *coriander* and contain the word *cilantro*. The problem of polysemy is that a query containing

the word *tank* may retrieve a set of documents containing phrases like *septic tank* and *tank top*, when only those containing the phrase *Bradley tank* were of interest. Both problems are related to the fact that documents containing more general words (*hypernyms*) and more specific word (*hyponyms*) may also be of interest; thus, for a query containing *car*, documents containing *vehicle* and *BMW* may both be of interest. In other words, a whole network of word associated to those in the query may be of interest. The hope offered by a technique like LSI is that compressing the similarity space of documents down to k terms may capture some of these associations.

For IR, then, the problem of choosing the right k is not just one of economy (choose the smallest k that does the job). The problem is to choose a k large enough to capture significant commonalities, but not so large as to simply memorize X . The natural measure of success is improvement in both precision and recall in document retrieval. Landauer et al. (1998) summarizes some early results [[Summarize some results]]

6 Appendix: Some background linear algebra

Theorem A	$AB = (B'A')'$
Theorem B	$(AB)' = B'A'$

Proof for XX' case:

(1)	XX'	=	$TSD'(TSD')'$	
(2)		=	$TSD'(D''(TS)')''$	Theorem A
(3)		=	$TSD'(D(TS)')$	Transposition
(4)		=	$TSD'(D(S'T')'')$	Theorem A
(5)		=	$TSD'(DS'T')$	Transposition
(6)		=	$TSS'T'$	Orthogonality of D
(7)	$(TS)(ST')$	=	TS^2T'	Diagonality of S

We see in step 7 that similarity matrix XX' can be expressed as the product of TS with ST' In fact by theorem B, and the diagonality of S , ST' is the transpose of TS:

$$(TS)' = S''T' = ST'$$

Thus taking dot products of rows of TS is equivalent to taking dot products of rows of X; the result carries over to $\widehat{X}\widehat{X}'$ and T_kS_k.

7 Practicalities

Many statistics and math packages include a singular value decomposition or svd command. Most of these trace their lineage back to the LAPACK (Linear Algebra package), written in Fortran 90, built on top of BLAS (Basic Linear Algenra Subprograms).

Python	from numpy.linalg import svd (LAPACK dgesdd)
Matlab	svd
R	svd, svd package

For this course, the additional code needed to compute an n-dimensional approximation (LSI) is the Python module **lsi**. The numpy version of svd is used, so numpy must be available.

SVD is quite computationally expensive. For an m x n matrix, we have a time bound of order

$$km^2n + k'n^3$$

k and k' are relatively small constants dependent on the algorithm. So you might want to be sure $m > n$ when you pass in your matrix, and transpose it if necessary, but the SVD implementation may do this for you. Check the documentation.

8 Problems

In Section 2, we introduced the similarity table S defined on our house of representatives data as follows:

$$S[i, j] = M[i, *] \cdot M[j, *] \tag{11}$$

This is a 435 x 435 table in which each cell (i, j) represents the similarity in the voting record of house member i with house member j.

The following questions concern S:

1. Assuming house member 231 voted on all 16 bills, what is the value of cell $S[231, 231]$?

2. In general, what is called the **diagonal** of a matrix S is the cells (i, i) where the row and column number are the same. For house members that voted on all 16 bills, what values fill the diagonal of matrix S?

3. What about the members with abstensions?

4. What is the minimum possible value of a cell of S on the diagonal? How would that minimum be acheived?

5. Our representation of house member votes does not produce vectors of constant length, because of abstentions. Propose an alternative representation of a voting record in which all the vectors have the same length, regardless of how many abstentions a representative has. In thinking about this, you can just use the sum of a vector's squared values as a substitute for its length. Hint: Use more than 16 dimensions to represent a voting record.

References

Bartell, Brian T., Garrison W. Cottrell, and Richard K. Belew. 1992. Latent semantic indexing is an optimal special case of multidimensional scaling. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR 92*, 161–167. Association for Computing Machinery.

Deerwester, S., S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science* 41(6):391–407.

Eckart, C., and G. Young. 1936. The approximation of one matrix by another of lower rank. *Psychometrika* 1(3):211–218.

Landauer, Thomas K, Peter W Foltz, and Darrell Laham. 1998. An introduction to latent semantic analysis. *Discourse processes* 25(2-3):259–284.

Manning, C.D., P. Raghavan, and H. Sch
"utze. 2008. *An Introduction to Information Retrieval* . Cambridge, UK: Cambridge University Press.

Schlimmer, J. 1987. *Concept acquisition through representational adjustment*. PhD thesis, Department of Information and Computer Science, University of California. As reported by [1].

Strang, Gilbert (Ed.). 1986. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press.