How Latent Semantic Indexing Solves the Pachyderm Problem

Michael A. Covington
Institute for Artificial Intelligence
The University of Georgia

2011

1 Introduction

Here I present a brief mathematical demonstration of how latent semantic indexing (LSI) solves the "pachyderm problem," the problem of indirect similarity between texts.

For brevity, I omit all background information about LSI.

I assume that the reader is familiar with ways of comparing individual texts by vocabulary, and has at least a vague memory of matrix multiplication.

2 The pachyderm problem

Elephants and pachyderms are essentially the same thing. (The word *pachyderm* originally denoted a class of animals, but since 1950 it has rarely been used to mean anything other than 'elephant.')

What this means is that if you are looking for texts about elephants, you probably want texts that mention pachyderms (even if they don't mention elephants at all), and vice versa. I call this the PACHYDERM PROBLEM.

How can a computer know that pachyderms and elephants are alike? Mainly by noticing that *most* texts that mention one of them also mention the other, and have other similar vocabulary; more generally, that *elephant* and *pachyderm* usually occur in the same contexts.

That is, the computer needs to recognize INDIRECT SIMILARITIES. Maybe text A does not directly resemble text B, but if A resembles C, D, and E, and B also resembles C, D, and E, then A and B should count as similar after all.

Analogous to the pachyderm problem is what we might call the RUN PROBLEM. To run a race is not the same as to run a computer program, and texts that contain the word *run* should not count as similar unless they have more in common than just that one word. That is, we want different instances of *run* to behave differently when they are in different contexts. Although I won't demonstrate it, LSI solves the run problem too.

3 A term-document matrix

Here is a made-up matrix of terms and their frequencies in different documents. You can see that Document 3 and Document 6 present the pachyderm problem: they are not a bit alike, but they both resemble Documents 4 and 5.

	Doc1	Doc2	Doc3	Doc4	Doc5	Doc6
apple	2	3	0	0	0	0
pear	3	4	0	0	0	0
elephant	0	0	2	2	3	0
pachyderm	0	0	0	2	4	2

We normally compare documents by comparing columns in this matrix. I'm going to assume you know various ways of doing this, such as vector comparison. You probably also know some ways of making vector comparison more effective, such as tf·idf weighting to de-emphasize the uninformative words. All those methods are just as applicable with LSI as without it. What we're going to do is transform this matrix in a way that solves the pachyderm problem.

4 Singular value decomposition

From here on we'll treat our term-document matrix as a matrix of numbers:

$$M = \begin{bmatrix} 2 & 3 & 0 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 3 & 0 \\ 0 & 0 & 0 & 2 & 4 & 2 \end{bmatrix}$$

SINGULAR VALUE DECOMPOSITION is a way of splitting any matrix M into three matrices U, D, and V^T which, when multiplied together, give you back the original matrix:

$$M = U\,D\,V^T$$

D is a diagonal matrix; that is, everything off its main diagonal is zero.

Notice also the notation V^T, which means that the matrix V that comes out of the decomposition algorithm needs to be flipped about its diagonal. In the following examples, that will be done.

The matrices have other special properties that are best appreciated by looking at an example. The singular value decomposition of our matrix M is:

$$\begin{bmatrix} -0.59 & 0.00 & 0.00 & 0.81 \\ -0.81 & 0.00 & 0.00 & -0.59 \\ 0.00 & 0.63 & -0.78 & 0.00 \\ 0.00 & 0.78 & 0.63 & 0.00 \end{bmatrix} \begin{bmatrix} 6.16 & 0 & 0 & 0 \\ 0 & 6.07 & 0 & 0 \\ 0 & 0 & 2.03 & 0 \\ 0 & 0 & 0 & 0.16 \end{bmatrix} \begin{bmatrix} -0.59 & -0.81 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.21 & 0.46 & 0.82 & 0.26 \\ 0.00 & 0.00 & -0.77 & -0.15 & 0.08 & 0.62 \\ -0.81 & 0.59 & 0.00 & 0.00 & 0.00 & 0.00 \end{bmatrix}$$

Roughly, the left-hand matrix U shows the relationships between the rows of the original one (although how this works is hard to see here because the original matrix doesn't have an overwhelmingly noticeable pattern). The right-hand matrix V^T shows the relationships between the columns of the original matrix. And the middle matrix D (called Σ in some books) shows how to mix them.

5 Now for an approximation...

Looking at the middle matrix D, you'll notice that not only are all the nonzero numbers on the main diagonal, they are sorted. The biggest numbers are at the top left.

What would happen if you chopped off the bottom right part of D — that is, set it to zero? It shouldn't change the result very much, since you're only removing the smallest factor.

In fact, if you change the middle matrix to

$$\begin{bmatrix} 6.16 & 0 & 0 & 0 \\ 0 & 6.07 & 0 & 0 \\ 0 & 0 & 2.03 & 0 \\ 0 & 0 & 0 & \mathbf{0} \end{bmatrix}$$

(making the change shown in boldface), you'll no longer be using the last column of the left-hand matrix, nor the last row of the right-hand matrix. (You'll just be multiplying them by zero.) In effect, you've whittled down the whole structure to this:

$$\begin{bmatrix} -0.59 & 0.00 & 0.00 \\ -0.81 & 0.00 & 0.00 \\ 0.00 & 0.63 & -0.78 \\ 0.00 & 0.78 & 0.63 \end{bmatrix} \begin{bmatrix} 6.16 & 0 & 0 \\ 0 & 6.07 & 0 \\ 0 & 0 & 2.03 \end{bmatrix} \begin{bmatrix} -0.59 & -0.81 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.21 & 0.46 & 0.82 & 0.26 \\ 0.00 & 0.00 & -0.77 & -0.15 & 0.08 & 0.62 \end{bmatrix}$$

Do the multiplication, and you still get a surprisingly good approximation of the original matrix:

$$\text{Original: } \begin{bmatrix} 2 & 3 & 0 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 3 & 0 \\ 0 & 0 & 0 & 2 & 4 & 2 \end{bmatrix} \quad \text{Reconstructed: } \begin{bmatrix} 2.11 & 2.92 & 0.00 & 0.00 & 0.00 & 0.00 \\ 2.92 & 4.06 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 2.00 & 2.00 & 3.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 2.00 & 4.00 & 2.00 \end{bmatrix}$$

Not bad! We threw away some numbers and hardly lost any information at all. (All these calculations were done with more significant digits than you see printed on the page.)

Let's do some more whittling. Let's trim the whole thing down to just two singular values (two numbers in the middle matrix) and the rows and columns that use them:

$$\begin{bmatrix} -0.59 & 0.00 \\ -0.81 & 0.00 \\ 0.00 & 0.63 \\ 0.00 & 0.78 \end{bmatrix} \begin{bmatrix} 6.16 & 0 \\ 0 & 6.07 \end{bmatrix} \begin{bmatrix} -0.59 & -0.81 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.21 & 0.46 & 0.82 & 0.26 \end{bmatrix}$$

What happens if we multiply these together? Not surprisingly the approximation is not as good as before. But it is *bad in a useful way.* Here's what we get:

$$\text{Original: } \begin{bmatrix} 2 & 3 & 0 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 3 & 0 \\ 0 & 0 & 0 & 2 & 4 & 2 \end{bmatrix} \quad \text{Reconstructed: } \begin{bmatrix} 2.11 & 2.93 & 0.00 & 0.00 & 0.00 & 0.00 \\ 2.92 & 4.06 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & \mathbf{0.79} & 1.76 & 3.13 & \mathbf{0.98} \\ 0.00 & 0.00 & \mathbf{0.98} & 2.19 & 3.89 & \mathbf{1.21} \end{bmatrix}$$

The conspicuous errors are the numbers shown in boldface, and look what has happened — they've somehow become a lot more like columns 4 and 5.

That is, *singular value decomposition has solved the pachyderm problem.* The frequencies of *elephant* and *pachyderm* have been blurred together. Recall that columns 3 and 6 were the odd ones, the ones that should have resembled each other but didn't. Now they do.

6 Using the right-hand matrix alone

The other important thing that has happened is that *we can compare documents by comparing the columns of the right-hand matrix V^T, rather than the original or reconstructed matrix.* That is, we don't have to do the reconstruction. It is obvious, looking at this right-hand matrix:

$$\begin{bmatrix} -0.59 & -0.81 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.21 & 0.46 & 0.82 & 0.26 \end{bmatrix}$$

that the first two columns are in one class and the latter four are in another. That is, Documents 1 and 2 resemble each other, and Documents 3, 4, 5, and 6 resemble each other. We have deliberately thrown away the row that would have been troubled by the pachyderm problem.

Some books refer to the rows of V, which are the same as the columns of V^T.

7 Folding-in

What if you want to compare the documents in the matrix with a document (or maybe a query) that was not around when you computed the singular value decomposition? You could put it into the original matrix and start over. But there is a shortcut. You can "fold it in."

Suppose the query contains *elephant* three times and none of the other terms. Then it corresponds to this column:

$$Q = \begin{bmatrix} 0 \\ 0 \\ 3 \\ 0 \end{bmatrix}$$

Let U_k, D_k, and V_k^T stand for our trimmed versions of the three matrices (originally U, D, and V^T, but we've trimmed them to k singular values).

We will need to transpose U_k (flip it about its main diagonal) to get U_k^T, and compute the inverse of D_k to get D_k^{-1}. Then we transform Q as follows:

$$Q_k = D_k^{-1} U_k^T Q = \begin{bmatrix} 0.00 \\ 0.31 \end{bmatrix}$$

which can be compared directly to columns in V_k^T. (Do the comparison by performing vector comparison just as you would have done in the original term-document matrix.) You see that this query is a perfect match for Documents 3, 4, 5, and 6, and no match at all for 1 and 2.

8 Implementation note

In this example I dealt with four words in six documents by limiting the decomposition to two singular values.

In real life, you are likely to have perhaps ten thousand words, hundreds of documents, and maybe 200 singular values. It is helpful to do tf·idf weighting before performing the decomposition.

9 Examples in R

Here is a term-document matrix that has the pachyderm problem:

```
> m
           Doc1 Doc2 Doc3 Doc4 Doc5 Doc6
apple         2    3    0    0    0    0
pear          3    4    0    0    0    0
elephant      0    0    2    2    3    0
pachyderm     0    0    0    2    4    2
```

Note that Doc3 and Doc6 are only indirectly related; they both resemble Doc4 and Doc5 but do not resemble each other.

Let's take the singular value decomposition of the matrix.

```
> svd(m)
$d
[1] 6.1623 6.0728 2.0302 0.1623

$u
        [,1]  [,2]   [,3]    [,4]
[1,] -0.5847 0.000  0.000  0.8112
[2,] -0.8112 0.000  0.000 -0.5847
[3,]  0.0000 0.627 -0.779  0.0000
[4,]  0.0000 0.779  0.627  0.0000

$v
        [,1]   [,2]     [,3]    [,4]
[1,] -0.5847 0.0000  0.00000 -0.8112
[2,] -0.8112 0.0000  0.00000  0.5847
[3,]  0.0000 0.2065 -0.76742  0.0000
[4,]  0.0000 0.4631 -0.14973  0.0000
[5,]  0.0000 0.8229  0.08426  0.0000
[6,]  0.0000 0.2566  0.61769  0.0000
```

Here `d` needs to be made into a diagonal matrix and `v` needs to be transposed. Doing this, and storing the three matrices in variables:

```
> d <- diag(svd(m)$d)
> u <- svd(m)$u
> vt <- t(svd(m)$v)
```

Now we have:

```
> d
      [,1]  [,2] [,3]   [,4]
[1,] 6.162 0.000 0.00 0.0000
[2,] 0.000 6.073 0.00 0.0000
[3,] 0.000 0.000 2.03 0.0000
[4,] 0.000 0.000 0.00 0.1623
> u
        [,1]  [,2]   [,3]    [,4]
[1,] -0.5847 0.000  0.000  0.8112
[2,] -0.8112 0.000  0.000 -0.5847
[3,]  0.0000 0.627 -0.779  0.0000
[4,]  0.0000 0.779  0.627  0.0000
> vt
        [,1]    [,2]    [,3]    [,4]    [,5]   [,6]
[1,] -0.5847 -0.8112  0.0000  0.0000 0.00000 0.0000
[2,]  0.0000  0.0000  0.2065  0.4631 0.82287 0.2566
[3,]  0.0000  0.0000 -0.7674 -0.1497 0.08426 0.6177
[4,] -0.8112  0.5847  0.0000  0.0000 0.00000 0.0000
```

Multiply them back together:

```
> u %*% d %*% vt
     [,1] [,2]      [,3] [,4] [,5]       [,6]
[1,]    2    3 0.000e+00    0    0  0.000e+00
[2,]    3    4 0.000e+00    0    0  0.000e+00
[3,]    0    0 2.000e+00    2    3 -1.363e-16
[4,]    0    0 1.055e-16    2    4  2.000e+00
```

That's a very close approximation to the original matrix. Let's zero out `d[4,4]`. That is equivalent to also zeroing out the last column of `u` and the last row of `vt`.

```
> d[4,4] <- 0
```

Multiply them again:

```
> u %*% d %*% vt
```

```
      [,1]  [,2]      [,3] [,4] [,5]       [,6]
[1,] 2.107 2.923 0.000e+00    0    0  0.000e+00
[2,] 2.923 4.055 0.000e+00    0    0  0.000e+00
[3,] 0.000 0.000 2.000e+00    2    3 -1.363e-16
[4,] 0.000 0.000 1.055e-16    2    4  2.000e+00
```

Very little change. Let's also zero out `d[3,3]`:

```
> d[3,3] <- 0
```

Multiply again:

```
> u %*% d %*% vt
      [,1]  [,2]   [,3]  [,4]  [,5]   [,6]
[1,] 2.107 2.923 0.0000 0.000 0.000 0.0000
[2,] 2.923 4.055 0.0000 0.000 0.000 0.0000
[3,] 0.000 0.000 0.7863 1.763 3.133 0.9769
[4,] 0.000 0.000 0.9769 2.191 3.893 1.2137
```

This is still mostly unchanged, but columns 3 and 6 suddenly resemble columns 4 and 5 (and each other) a lot more than they used to. We've solved the pachyderm problem.

As a shortcut, you can give arguments to the `svd()` function to tell it to limit itself to a specific number of singular values.

Literature

The computations described here, but not the notation, are based on:

Manning, Christopher R.; Raghavan, Prabhakar; and Schütze, Hinrich (2009) *Introduction to Information Retrieval.* Cambridge: Cambridge University Press. Free text on line at *http://nlp.stanford.edu/IR-book/.*

The algorithm for singular value decomposition can be found in the *Numerical Recipes* books by Press et al. for various programming languages.