SVD and LSI Tutorial4: Latent Semantic Indexing (LSI) How-to Calculations

Dr. E. Garcia
Mi Islita.com

Topics

Introduction

Warning: Term Count Model Ahead

Snakeoil Marketers and LSI

Debunking Few More SEO Myths

Constructing the LSI Term-Document Matrix

Computing U, S, V and V^T

Dimensionality Reduction: Computing U_k, S_k, V_k and V_k^T

Incorporating the Query and Ranking the Documents

LSI in Few Easy Steps

LSI and Adversarial Practices: When Web Spammers Strike

Summary

http://www.miislita.com/information-retrieval-tutorial/svd-lsi-tutorial-4-lsi-how-to-calculations.html

Tutorial Review

References

Introduction

In this tutorial you will learn how Singular Value Decomposition (SVD) is used in Latent Semantic Indexing (LSI) to score documents and queries. Do-it-yourself procedures using an online matrix calculator are described. After completing this tutorial, you should be able to

1. learn about basic LSI models, then move forward to advanced models.
2. understand how LSI ranks documents.
3. replicate all calculations and experiment with your own data.
4. get out of you head the many SEO myths and fallacies about LSI.
5. stay away from "LSI based" Snakeoil Search Marketers.

The later is a sector from the search marketing industry (SEOs/SEMs) that claim to provide "LSI based" services while giving a black eye to the marketing industry before the information retrieval community. These marketers will put out any possible pseudo-science argument in order to pitch you their services. I have exposed their tactics and tricks in the blog

Latest SEO Incoherences (LSI) - My Case Against "LSI based" Snakeoil Marketers.

Before delving into LSI a little review is in order. Believe it or not, this is related with these snakeoil sellers.

Assumptions and Prerequisites

This is a tutorial series, so I assume you have read previous parts. In Part 1 we mentioned that if you want to replicate the calculations you must be familiar with linear algebra or understand the material covered in:

- Matrix Tutorial 1: Stochastic Matrices
- Matrix Tutorial 2: Basic Matrix Operations
- Matrix Tutorial 3: Eigenvalues and Eigenvectors

If you are not familiar with linear algebra or have not read the tutorials, please STOP AND READ THESE.

You might also find useful these fast tracks, which are designed to serve as quick references for our readers:

Latent Semantic Indexing (LSI) Fast Track Tutorial
Singular Value Decomposition (SVD) Fast Track Tutorial

Warning: Term Count Model Ahead

One of the first Term Vector models was the Term Count Model. In this model the weight of term i in document j is defined as a local weight (L_{ij}):

Equation 1: $w_{ij} = L_{ij} = tf_{ij}$

where tf_{ij} is term frequency or number of times term i occurs in document j. So, let say that document 1 repeats *latent* 3 times and *semantic* 2 times. The weights of these terms in document j are then 3 and 2 in each case. If we treat queries as documents, then query weights are defined the same way. Thus, the weight of term i in a query q is defined as $w_{iq} = tf_{iq}$.

This model has many limitations and theoretical flaws. Among others:

- by repeating terms many times these become artificially relevant. Thus, the model is easy to game.
- long documents are favored since these tend to have more words and term occurrences. These will tend to score high simply because they longer, not because they are relevant.
- the relative global frequency of terms across a collection of documents is ignored.
- repetition does not mean that terms are relevant. These might be repeated within different contexts or topics.

There are other theoretical flaws, but these are the obvious ones. Today we can do better. Compared with Equation 1, a better description for term weights is given by

Equation 2: $w_{ij} = L_{ij}G_iN_j$

where

1. $L_{i,j}$ is the local weight for term i in document j.
2. G_i is the global weight for term i across all documents in the collection.
3. N_j is the normalization factor for document j.

Local weights are functions of how many times each term occurs in a document, global weights are functions of how many times documents containing each term appears in the collection, and the normalization factor corrects for discrepancies in the lengths of the documents across the collection. Advanced models incorporate global entropy measures, probabilistic measures and link analysis. Evidently, how documents are ranked is determined by which term weight scheme one uses with Equation 2. In 1999 Erica Chisholm and Tamara G. Kolda from Oak Ridge National Labs reviewed several term weight schemes using Equation 2 in New Term Weighting Formulas for the Vector Space. Many schemes have been proposed since then.

What does all this has to do with LSI and snake oil marketers? A lot.

Snakeoil Marketers and LSI

SEOs like to copy/paste/quote early papers on LSI not realizing that these describe LSI implementations where entries of the term-document matrix **A** are defined using the Term Count Model. Such implementations served a purpose. In the first LSI papers controlled collections free from spam and with documents of similar sizes and formats were used. LSI was applied to collection of titles, abstracts, research papers and regulated examinations (TOEFL, SAT, GRE, etc.) of more or less same size and free from commercial noise. To top off, by defining **A** using mere word occurrences those LSI implementations inherited some of the limitations and theoretical flaws associated to the Term Count Model.

It was immediately realized that retrieval performance can be improved by redefining term weights as given by Equation 2. This is particularly useful these days. Today we deal with unstructured Web collections, where global weights are important and where documents come in different lengths and formats. This is why many LSI implementations today are not based on Equation 1, but on Equation 2.

Figure 1 shows how Equation 1 and Equation 2 are incorporated in a term-document matrix. When we talk about LSI and Term Vector Models, it turns out that this matrix is their common denominator. Thus, how we define term weights is critical.

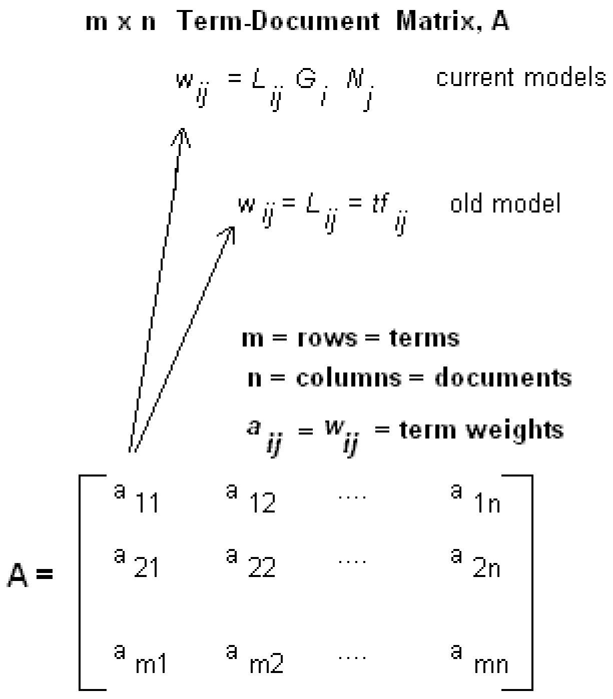
m x n Term-Document Matrix, A
$w_{ij} = L_{ij} G_i N_j$ current models
$w_{ij} = L_{ij} = tf_{ij}$ old model
m = rows = terms
n = columns = documents
$a_{ij} = w_{ij}$ = term weights
$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$

Figure 1. Term-document matrix. How we define term weights affects Term Vector and LSI implementations.

What are the chances for end users to replicate search engine scores? Since end users do not have access to the corpus of a search engine they cannot compute global weights, normalize all documents or compute global entropy values, among other limitations. Thus, their chances to replicate what a search engine does is close to zero.

This is why "LSI based" Snake Oil Marketers resource to primitive tools where word occurrences and just few number of search results are used. So, by default whatever these tools score is a poor caricature of what a search engine like Google or Yahoo! probably scores. You get what you pay for. If you want to be fooled by these firms, be my guest. As a sidekick, let me mention that in the first issue of IR Watch - The Newsletter our subscriber learned how LSI models based on mere word occurrences can be gamed by spammers.

Debunking Few More SEO Myths

Before scoring documents with LSI or with a term vector model we need to construct the term-document matrix **A**. This means that documents have been already indexed, unique terms have been extracted and term weights have been computed. Thus, the idea that LSI is document indexing is just another misconception. Our Document Indexing Tutorial explains what is/is not document indexing. Ruthven and Lalmas have summarized document indexing in details in A survey on the use of relevance feedback for information access systems.

Many search marketers claim that their link based services and products are designed to facilitate such "LSI document indexing". Others claims that you need to be part of a link building program or stuff web pages with links rich in synonyms, nouns and specific phrases to improve your chances. They also claim to have already identified through "keyword research" such phrases or "magic words" for you. Even others claim that you can identify these keywords by using their "LSI tools". The allegation here is that these keywords help search engines like Google to rank high your Web pages.

These are some of the many myths promoted in the search marketing industry. Other SEOs claim that search engines use LSI to score documents by identifying nouns, adjectives and other type of word senses and by "learning" where in the document these occur. Even others claim that search engines use LSI to selectively rank pages from specific web domains.

The fact is that when applying LSI to a term-document matrix we are not concerned about the nature of the terms or where in the documents these occur. Whether terms are nouns, articles, verbs, etc or are found in a special portion of a document like in anchor texts, links or specific passages or are part of a specific web domain plays no role when matrix **A** is decomposed via SVD.

There are other marketers claiming out of thin air that LSI is associative indexing like stem indexing. This is untrue since matrix **A** must be constructed before putting to work any LSI or term vector model. True that after having a collection of documents indexed one could use the concept matching power of LSI to cluster (regroup) documents and terms, but those are another twenty bucks.

Such regrouping is done using the **V** and **U** matrices and for several purposes: for example, to conduct query expansion or expand an answer set, to reclassify documents or even to construct an automatic thesaurus. The term "indexing" in LSI refers to these post-indexing activities more than to the initial indexing of documents. If you want to refer to these as "reindexing activities" that's fine with me.

How about the claim that search engines use LSI to score anchor text (links)? The only scenario I can think of is if a search engine uses LSI to score a collection of links and **nothing else**. Here the search engine could SVD a term-link matrix **L**, score links, and cluster results (links or terms). After that scores could be combined with other scoring schemes.

Whether such approach improves information retrieval performance and current document ranking schemes remains to be seen. One thing is clear: LSI as a link analysis tool is nothing new. In fact, objects-only analysis (links, titles, images, etc) is possible with LSI. Search engines using such approach to retrieve results from billion document collections under less than a second? Theoretically speaking, perhaps. Practically speaking, nope --at least not at the time of writing this tutorial.

Constructing the LSI Term-Document Matrix

Myths, SEO rants, and term count models aside, why then term vector and LSI models based on word occurrences are taught in graduate schools? Good question. These are taught **to introduce IR students to the basics**. After that students can move forward and experiment with more realistic models, compare techniques and draw conclusions.

Since I assume you are new to LSI, unfortunately I have to use the same simplistic approach in this tutorial. Thus, during the rest of this article I am presenting an LSI implementation based on the primitive Term Count Model (Equation 1). Once you have learned the basics, I will show you how Equation 2 is used with current LSI models. Baby steps first. Having said all that, let's move on.

For the working example that follows we will be using the following resources (1, 2):

1. Information Retrieval: Algorithms and Heuristics by David A. Grossman and Ophir Frieder. Springer, 2nd Edition, 2004. This is a book we have reviewed and we highly recommmend you to buy.
2. BlueBit Matrix Calculator. We recommend you to access online or buy this tool. It is a great tool for solving small matrices and testing things.

Let's proceed.

Open your copy of Grossman and Frieder in page 71, where the query is *gold silver truck* and the "collection" consists of just three "documents":

- d1: *Shipment of gold damaged in a fire.*
- d2: *Delivery of silver arrived in a silver truck.*
- d3: *Shipment of gold arrived in a truck.*

Please note that Grossman and Frieder use the following document indexing approach: (a) stopwords **are not** ignored, (b) text is tokenized and lowercased, (c) no stemming is used and (d) unique terms are sorted alphabetically. For our tutorial, this document indexing approach is good enough. However, in current LSI models stopwords and term occurring once in a document are usually ignored. Stemming and sorting is optional. All this is done to reduce computational overhead.

If we are interested in ranking documents using a Term Count Model we simply construct the following term-document matrix and query column matrix.

$$\begin{array}{l} \text{Terms} \\ \downarrow \\ \text{a} \\ \text{arrived} \\ \text{damaged} \\ \text{delivery} \\ \text{fire} \\ \text{gold} \\ \text{in} \\ \text{of} \\ \text{shipment} \\ \text{silver} \\ \text{truck} \end{array} \qquad \mathbf{A} = \begin{array}{ccc} \text{d1} & \text{d2} & \text{d3} \\ \downarrow & \downarrow & \downarrow \end{array} \!\!\!\! \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix} \qquad \mathbf{q} = \begin{array}{c} \text{q} \\ \downarrow \end{array} \!\!\!\! \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Figure 2. Term-document matrix and query matrix example.

After that we could rank results in decreasing order of cosine similarity values as described in The Classic Term Vector Model or as described in A Linear Algebra Approach to Term Vectors.

However, here we are using LSI so we ignore for now the query matrix and decompose the term-document matrix into three new matrices

Equation 3: $\mathbf{A} = \mathbf{USV}^{\mathbf{T}}$

Computing U, S, V and V^T

Even with a collection consisting of just 3 documents the **A** matrix is not that small, so we better resource to software to simplify calculations. In the process you should be able to "connect the dots" between the SVD knowledge acquired from past tutorials and what you are about to do.

For our tutorial any software or matrix calculator that does SVD is good enough. If you have MathLab or SciLab that would be great. If you don't have access to these try the JavaScript Singular Value Decomposition Calculator. Be aware that these tools come with their own learning curves and sign conventions (* See footnote). I like the Bluebit Matrix Calculator so this is the one I will be using.

Proceed as follows:

1. Access the Bluebit Matrix Calculator.
2. Select *Values are delimited by* **Spaces** and *Show results using* **4** *decimal digits*. You can overwrite later these parameters if you wish.
3. Check the *Singular Value Decomposition* option. Enter the data as given in matrix **A** of Figure 2.
4. Review the data you just have entered. If everything is correct, press **Calculate** button.

You should be able to generate the **U**, **S** and **V** matrices. Note that Bluebit does not gives you $\mathbf{V}^{\mathbf{T}}$. For our LSI test this actually is quite convenient since we don't need $\mathbf{V}^{\mathbf{T}}$. If you still want to generate a full SVD or use $\mathbf{V}^{\mathbf{T}}$ with other calculations then you need to transpose **V**. Figure 3 shows all these matrices. I am including $\mathbf{V}^{\mathbf{T}}$ for your perusal.

$$\mathbf{U} = \begin{bmatrix} -0.4201 & 0.0748 & -0.0460 \\ -0.2995 & -0.2001 & 0.4078 \\ -0.1206 & 0.2749 & -0.4538 \\ -0.1576 & -0.3046 & -0.2006 \\ -0.1206 & 0.2749 & -0.4538 \\ -0.2626 & 0.3794 & 0.1547 \\ -0.4201 & 0.0748 & -0.0460 \\ -0.4201 & 0.0748 & -0.0460 \\ -0.2626 & 0.3794 & 0.1547 \\ -0.3151 & -0.6093 & -0.4013 \\ -0.2995 & -0.2001 & 0.4078 \end{bmatrix} \qquad \mathbf{S} = \begin{bmatrix} 4.0989 & 0.0000 & 0.0000 \\ 0.0000 & 2.3616 & 0.0000 \\ 0.0000 & 0.0000 & 1.2737 \end{bmatrix}$$

$$\mathbf{V} = \begin{bmatrix} -0.4945 & 0.6492 & -0.5780 \\ -0.6458 & -0.7194 & -0.2556 \\ -0.5817 & 0.2469 & 0.7750 \end{bmatrix} \qquad \mathbf{V}^{\mathbf{T}} = \begin{bmatrix} -0.4945 & -0.6458 & -0.5817 \\ 0.6492 & -0.7194 & 0.2469 \\ -0.5780 & -0.2556 & 0.7750 \end{bmatrix}$$

Figure 3. SVD results from the Bluebit Matrix Calculator.

Note that **S** consists of three nonzero singular values, confirming that **A** is a Rank 3 Matrix.

* See footnote on BlueBit recent upgrade

Dimensionality Reduction: Computing $\mathbf{U}_k$, $\mathbf{S}_k$, $\mathbf{V}_k$ and $\mathbf{V}_k^T$

In LSI it is not the intent to reproduce **A**. What we want is to retain the largest *k* singular values. In the literature this is called dimensionality reduction.

As mentioned in Part 3 of this tutorial series, the choice of *k* is done **"by seat of the pants"**. How many *k* singular values or dimensions to keep is done more or less arbitrarily or must be determined experimentally since each collection is different. This is an active open area of research. Studies with thousand of documents suggest that *k* values around 100 give better retrieval performance. Definitely the selection of *k* impacts results.

For the example at hand let's just keep the first two singular values ($k = 2$). We call this a Rank 2 Approximation. Essentially this is what we do: we keep the first 2 columns of **U** and **V** and the first two rows and columns of **S**.

$$k = 2$$

$$\mathbf{U} \approx \mathbf{U}_k = \begin{bmatrix} -0.4201 & 0.0748 \\ -0.2995 & -0.2001 \\ -0.1206 & 0.2749 \\ -0.1576 & -0.3046 \\ -0.1206 & 0.2749 \\ -0.2626 & 0.3794 \\ -0.4201 & 0.0748 \\ -0.4201 & 0.0748 \\ -0.2626 & 0.3794 \\ -0.3151 & -0.6093 \\ -0.2995 & -0.2001 \end{bmatrix} \qquad \mathbf{S} \approx \mathbf{S}_k = \begin{bmatrix} 4.0989 & 0.0000 \\ 0.0000 & 2.3616 \end{bmatrix}$$

$$\mathbf{V} \approx \mathbf{V}_k = \begin{bmatrix} -0.4945 & 0.6492 \\ -0.6458 & -0.7194 \\ -0.5817 & 0.2469 \end{bmatrix} \qquad \mathbf{V}^T \approx \mathbf{V}_k^T = \begin{bmatrix} -0.4945 & -0.6458 & -0.5817 \\ 0.6492 & -0.7194 & 0.2469 \end{bmatrix}$$

Figure 4. A Rank 2 Approximation.

Incorporating the Query and Ranking the Documents

Now to incorporate the query we use the procedure described by Berry, Dumais and OBrien in Using Linear Algebra for Intelligent Information Retrieval (3). Since **S** is symmetric along its diagonal, $\mathbf{S}^T = \mathbf{S}$ and from Equation 3 we can see that

Equation 4: $\mathbf{A}^T = (\mathbf{U}\mathbf{S}\mathbf{V}^T)^T = \mathbf{V}\mathbf{S}\mathbf{U}^T$

Equation 5: $\mathbf{A}^T\mathbf{U}\mathbf{S}^{-1} = \mathbf{V}\mathbf{S}\mathbf{U}^T\mathbf{U}\mathbf{S}^{-1}$

Equation 6: $\mathbf{V} = \mathbf{A}^T\mathbf{U}\mathbf{S}^{-1}$

Now let's consider the case of **A** consisting of n > 1 documents. **V** must consists of n rows, each containing the coordinates of a document vector. For a given document vector **d** Equation 6 can be rewritten as

Equation 7: $\mathbf{d} = \mathbf{d}^{T}\mathbf{U}\mathbf{S}^{-1}$

Since in LSI a query is treated just as another document then the query vector is given by

Equation 8: $\mathbf{q} = \mathbf{q}^{T}\mathbf{U}\mathbf{S}^{-1}$

Thus, in the reduced *k*-dimensional space we can write

Equation 9: $\mathbf{d} = \mathbf{d}^{T}\mathbf{U}_{k}\mathbf{S}_{k}^{-1}$

Equation 10: $\mathbf{q} = \mathbf{q}^{T}\mathbf{U}_{k}\mathbf{S}_{k}^{-1}$

Equation 9 and Equation 10 contain the new coordinates of the vectors in this reduced space. Query-document cosine similarity measures are then possible using

Equation 11: $\mathbf{sim(q, d) = sim(q}^{T}\mathbf{U}_{k}\mathbf{S}_{k}^{-1}\mathbf{, d}^{T}\mathbf{U}_{k}\mathbf{S}_{k}^{-1})$

We can reuse Equation 9, 10 and 11 anytime we want to compare cosine similarities between documents and queries. However, with n number of documents this is a formidable task. So, let's simplify a bit further.

From **V** we can see that for n number of documents, this matrix must contain n number of rows holding eigenvector values. Each of these rows then holds the coordinates of individual document vectors. From Figure 4 these coordinates are:

d1(-0.4945, 0.6492)
d2(-0.6458, -0.7194)
d3(-0.5817, 0.2469)

It is now clear that in LSI the "right" eigenvectors from the SVD algorithm are document vector coordinates. So, we just need to compute the new query vector coordinates in the reduced space (see Figure 5). An analogous analysis demonstrates that the rows of **U** holds term vector coordinates.

$$q = q^T U_k S_k^{-1} \qquad k = 2$$

$$q = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} -0.4201 & 0.0748 \\ -0.2995 & -0.2001 \\ -0.1206 & 0.2749 \\ -0.1576 & -0.3046 \\ -0.1206 & 0.2749 \\ -0.2626 & 0.3794 \\ -0.4201 & 0.0748 \\ -0.4201 & 0.0748 \\ -0.2626 & 0.3794 \\ -0.3151 & -0.6093 \\ -0.2995 & -0.2001 \end{bmatrix} \begin{bmatrix} \frac{1}{4.0989} & 0.0000 \\ 0.0000 & \frac{1}{2.3616} \end{bmatrix}$$

$$q = \begin{bmatrix} -0.2140 & -0.1821 \end{bmatrix}$$

Figure 5. Computing the query vector

In Part 3 of this tutorial we introduced a shortcut where **S** was inverted. Figure 5 reveals that such inversion comes handy. By the way, Bluebit and almost any matrix calculator inverts matrices for you, so this is not a formidable task --with huge matrices, you can always resource to approximation methods.

Finally, we rank documents in descending order of cosine similarities as described in The Classic Vector Space Model. Figure 5 and 6 illustrate this.

$$\mathrm{sim}(q, d) = \frac{q \bullet d}{|q|\,|d|}$$

$$\mathrm{sim}(q, d_1) = \frac{(-0.2140)\,(-0.4945) + (-0.1821)\,(0.6492)}{\sqrt{(-0.2140)^2 + (-0.1821)^2}\,\sqrt{(-0.4945)^2 + (0.6492)^2}} = -0.0541$$

$$\mathrm{sim}(q, d_2) = \frac{(-0.2140)\,(-0.6458) + (-0.1821)\,(-0.7194)}{\sqrt{(-0.2140)^2 + (-0.1821)^2}\,\sqrt{(-0.6458)^2 + (-0.7194)^2}} = 0.9910$$

$$\mathrm{sim}(q, d_3) = \frac{(-0.2140)\,(-0.5817) + (-0.1821)\,(0.2469)}{\sqrt{(-0.2140)^2 + (-0.1821)^2}\,\sqrt{(-0.5817)^2 + (0.2469)^2}} = 0.4478$$

Ranking documents in descending order

$$d_2 > d_3 > d_1$$

Figure 5. Scoring and ranking documents in LSI

Thus, d2 is ranked higher than d3 and d1 while d1 receives the lowest rank.

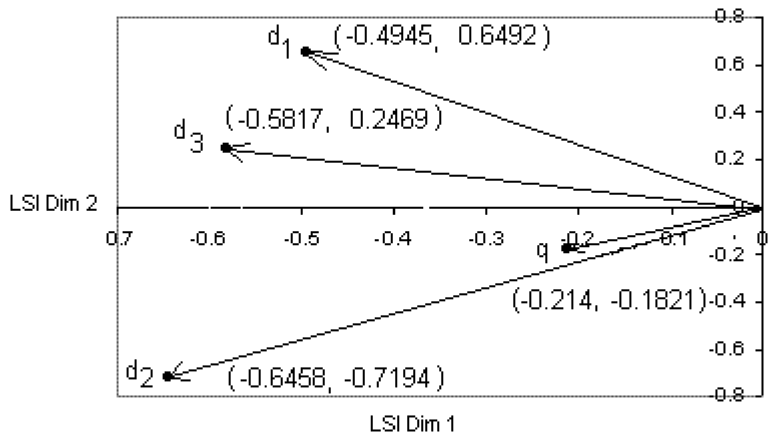

Figure 6. LSI query-document vectors.

Figure 5 and Figure 6 shows why document d2 scores higher than d3 and d1. Its vector is closer to the query vector than the other two vectors. Note that in LSI negative scores are possible.

Also note that Term Vector Theory is still used at the beginning and at the end of LSI. Thus, the idea that Term Vector Theory is divorced from LSI is incorrect. Both theories are complementary. Vectors are used to represent documents and terms, but now these are dependent from each other in the reduced space. In essence, LSI incorporates an additional processing layer for computing more meaningful vectors that otherwise one would not compute by using plain term vector models.

Please note

To insure that the results from this example were accurate, we repeated several times the calculations. We also obtained identical results using the Singular Value Decomposition Calculator. Unfortunately, when we compared these results with Grossman and Frieder results, it appears they published a little typo. In page 71 and 72 they have the following entry in the $\mathbf{V}^T$ matrix: -0.2469. All our tests indicate that the negative sign should not be there (0.2469). This little typo changes the cosine similarity computed for document d3, making d3 as relevant as d2 (*sim(q, d3)* = 0.9543 vs. *sim(q, d2)* = 0.9910)) when in fact this is not the case. For d3 we computed *sim(q, d3)* = 0.4478.

Luckly, the example at hand is the very same we used in the article The Classic Vector Space Model and those results confirm that d3 is not as relevant as d2. In their book the authors used the same example with other scoring systems and their own results indicate that in fact d3 is not as relevant as d2. True that relevance comparison between dissimilar scoring systems can be risky, but in this case no matter how we look at the results d3 is not as relevant as d2 for the intended query. We have notified the authors about the typo since then.

LSI in Few Easy Steps

Note that to rank documents with LSI, we just need to

1. compute weights using a specific term weight scoring system.
2. construct the term-document matrix **A**.
3. decompose **A**, compute and truncate all required matrices.
4. find new coordinates of query and document vectors in the reduced *k*-dimensional space.
5. sort documents in decreasing order of cosine similarity values.

Everything is pretty much straightforward and crystal clear.

You can now get out of your head all the SEO myths, misconceptions, fallacies, lies, and speculations you have heard in search marketing conferences, "LSI videos", forums and blogs. **There are no magic words here**.

LSI and Adversarial Practices: When Web Spammers Strike

Well this was a pretty basic tutorial on LSI. We have used mere term occurrences to create a term-document matrix. However, this way of scoring term weights inherits several flaws associated to the Term Count Model. As Grossman and Frieder mention in page 127 (emphasis added):

"The key to similarity is not that two terms happen to occur in the same document: it is that the two terms appear in the same ***CONTEXT*** --that is they have very similar neighboring terms."

Incidentally, this is precisely what one tries to estimate through EF-Ratios computed from answer sets: find the fraction of documents retrieved containing terms within similar neighborhoods. Only because any two documents mentions any two terms or more, this **does not mean** that there is a **contextuality relationship** between terms as these might occur in documents discussing different topics. This is illustrated in Figure 7.

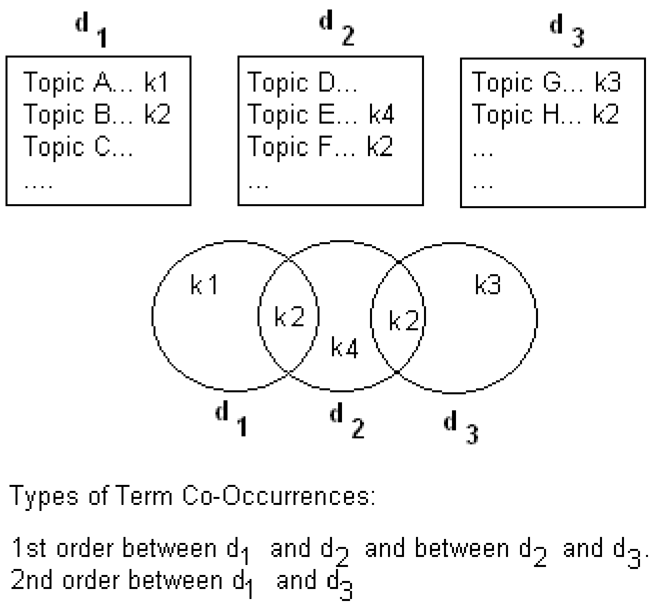

Figure 8. Co-Occurrence Orders.

Here we need to distinguish between **intradocument** and **interdocument** term-term co-occurrence. In addition to the **type** of co-occurrence we need to discriminate between the **order** of the co-occurrence phenomenon. Note that

d1 mentions terms k1 and k2.
d2 mentions terms k2 and k4.
d3 mentions terms k2 and k3.

In *individual documents*, terms co-occur. Now let examine the term co-occurrence phenomenon between documents.

There is a first order term-term co-occurrence relationship between documents d1 and d2.
There is also a first order term-term co-occurrence relationhsip between documents d2 and d3.
There is a second order term-term co-occurrence relationship between d1 and d3.

However, terms occur in different topics. In this case LSI similarity scores based on word counts do not reflect the actual relationship between terms. Add polysems and the scenario worsens.

What does this has to do with spammers?

Incidentally, more and more spammers are realizing this flaw and that spurious induced similarity can be injected into Web collections in at least three different ways:

1. by creating documents with several topics and selectively optimizing these for ranking purposes.
2. by writing content rich in related terms and synonyms.
3. by creating a network of such documents pointing to a target document.

The goal here is to rank high a document so users are exposed to a drop-by presentation of other topics (e.g. ads). And we have not considered yet well known "black", "white" and "gray" SEO optimization strategies designed to place documents at the top of search results or for the mere purpose of diluting search result pages.

At this point one wonders if the great LSI results obtained from controlled collections like from research abstracts, same length documents, email forensic studies, student term papers, regulated exams (SAT, TOEFL, GRE), limited pay-for performance collections, etc could ever be achieved with uncontrolled Web collections consisting of documents of different lengths, formats and containing many topics, news stories, ads, spam, link bombs, etc.

If you still want to trust SEO "LSI based" tools then be my guest.

Summary

In this part of the tutorial you have learned how Singular Value Decomposition (SVD) is used in Latent Semantic Indexing (LSI). You also have learned how documents and queries are scored. A simple procedure for ranking documents using just an online matrix calculator has been described. The Term Count Model has been used to assign weights to terms. Several SEO myths have been debunked and few LSI limitations have been pointed out. In particular, we have pointed out the problems of assigning weights to terms using term counts.

Intentionally we limited the discussion to computing query-document similarities. We have not included things like **folding-in** new documents in the reduced space or other techniques that use SVD or LSI. One of the great things about LSI is that the same dimensional space can be used to make document-document and term-term comparisons. In the old LSI literature this is done as follows:

1. the rows of **V** holds document vector coordinates, so to compare any two documents we could use the corresponding rows and compute cosine similaries. With this information we can do document clustering, classification, construct directories of similar documents, or even examine if any two documents are duplicates.
2. the rows of **U** holds term vector coordinates, so to compare any two terms we could use the corresponding rows and compute cosine similarities. With this information we can do some term clustering, automatic construction of a thesaurus, finding similar terms, or even conduct keyword research studies.

Today we have advanced methods for clustering documents and terms with LSI, even with LSI based on term count scores. These methods involve the use of the $\mathbf{A}^T\mathbf{A}$ and $\mathbf{AA}^T$ matrices (4, 5). As our subscriber learned in the first issue of IR Watch - The Newsletter, recent research indicates that high-order co-occurrence patterns found in $\mathbf{AA}^T$ might be at the heart of LSI. A lot of research concentrates now on tracing these connectivity paths back to their original source in the $\mathbf{AA}^T$ matrix. Unfortunately, most of the published articles are limited to word occurrences, but is a good start. That research is improving the current theoretical framework in which LSI models are based.

I am planning to cover these methods in a special article, but not as part of this tutorial series. Rest assure that at the time of writing this, current "LSI based" tools might not be using these methods, but making a caricature out of LSI technologies. Meanwhile, wait for the next article of this tutorial series.

* BlueBit Important Upgrade

Note 1 After this tutorial was written, BlueBit upgraded the SVD calculator and now is giving the $\mathbf{V}^T$ transpose matrix. We became aware of this today 10/21/06. This BlueBit upgrade doesn't change the calculations, anyway. Just remember that if using $\mathbf{V}^T$ and want to go back to **V** just switch rows for columns.

Note 2 BlueBit also uses now a different subroutine and a different sign convention. Absolutely none of these changes affect the final calculations and main findings of the example given in this tutorial. Why?

Readers familiar with eigenvectors know that if we multiply an **entire** column of **U** and the corresponding **entire** row of $\mathbf{V}^T$ by -1 this will not change the final results. The final graphical representation will simply flip coordinates since orthogonality is preserved, but the *relative* distribution of vectors remains the same.

We contacted Trifon Triantafillidis, inventor of the BlueBit Software, who confirmed this for us. Trifon kindly explained the following (and I partially quote):

"Recently we rewrote the software of the Online Matrix Calculator and instead of using Matrix ActiveX Component we used .NET Matrix Library. The two products MaXC and NML are based on different linear algebra libraries. MaXC is based on LINPACK and NML is based on LAPACK."

"LINPACK and LAPACK are libraries written in FORTRAN which perform linear algebra calculations. LAPACK is newer and faster whereas LINPACK is considered a bit outdated."

"Now, because MaXC and NML are based on different libraries give different results on SVD. The question is which one is correct? The answer is both!"

He then elegantly explained why when we use any sign convention, this should not affect the final results. Trifon added:

"Now, LINPACK and LAPACK use different routines to compute eigenvalues and eigenvectors. They return the same eigenvectors but their signs may be reversed. Starting from reversed signs in an eigenvector you can get reversed signs in a column of matrix U and reversed signs in a column of matrix V or reversed signs in a row of matrix V'."

"I know this may confuse readers of your tutorial at first but they have to understand that there is not any way to say that any sign is better than another. We should not care if the signs are reversed in eigenvectors or if the signs are reversed in a column of the U matrix or in a row of the V' matrix."

We are in debt to Trifon for such great explanation and brilliant piece of software.

Indeed, it should not matter as long as we multiply an entire row-column pair. In such case there is a preservation of orthogonality. However, if we change the sign of a single entry it would then matters. This is why Grossman and Frieder typo affected their results.

As mentioned, as long as the reader uses a software pack with the corresponding sign convention the final results will not change from software to software. The final data and graph will simply preserve its overall distribution, but with coordinates reversed.

For inquisitive readers, I am reproducing below the example given above using the upgraded version of BlueBit:

$$\mathbf{U} = \begin{bmatrix} -0.4201 & -0.075 & -0.046 \\ -0.2995 & 0.2001 & 0.4078 \\ -0.1206 & -0.275 & -0.4538 \\ -0.1576 & 0.3046 & -0.2006 \\ -0.1206 & -0.275 & -0.4538 \\ -0.2626 & -0.379 & 0.1547 \\ -0.4201 & -0.075 & -0.046 \\ -0.4201 & -0.075 & -0.046 \\ -0.2626 & -0.379 & 0.1547 \\ -0.3151 & 0.6093 & -0.4013 \\ -0.2995 & 0.2001 & 0.4078 \end{bmatrix} \qquad \mathbf{S} = \begin{bmatrix} 4.0989 & 0 & 0 \\ 0 & 2.3616 & 0 \\ 0 & 0 & 1.2737 \end{bmatrix} \qquad \mathbf{V}^T = \begin{bmatrix} -0.4945 & -0.6458 & -0.5817 \\ -0.6492 & 0.7194 & -0.2469 \\ -0.578 & -0.2556 & 0.775 \end{bmatrix}$$

$$\mathbf{q} = \begin{bmatrix} -0.2140 & 0.1821 \end{bmatrix}$$

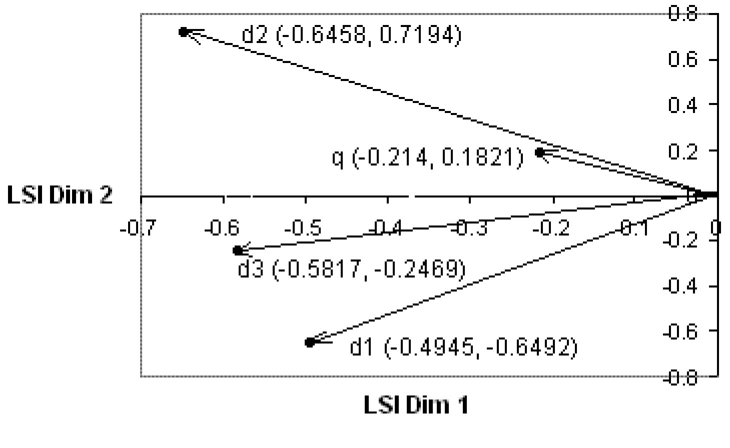

Figure 9. Recomputed Results with the BlueBit Upgrade.

Tutorial Review

1. Rework the example given in this tutorial, but this time remove all stopwords before constructing **A**. Define term weights as follows

query: $w_{iq} = tf_{iq}$
documents: $w_{ij} = tf_{ij}$

2. Rework the example given in this tutorial, but this time remove all stopwords before constructing **A**, Define term weights as follows

   query: $w_{iq} = tf_{iq}$
   documents: $w_{ij} = tf_{ij}*log(D/d_i)$

   where d_i is the number of documents containing term i. D is the collection size; in this case $D = 3$.
3. Repeat exercise 2. Define term weights as follows

   query: $w_{iq} = tf_{iq}$
   documents: $w_{ij} = tf_{ij}*log((D-d_i)/d_i)$

Bonus

Select a list of book titles or web page titles. Treat these as "documents". Consider that these conform a "collection". Extract unique terms, exclude stopwords, lowercase and tokenize the text. Construct the **A** matrix and use LSI to rank documents. Define term weights as follows:

query: $w_{iq} = tf_{iq}$
documents: $w_{ij} = (tfij/tfmax_{ij})*log((D-d_i)/d_i)$

where

$tfmax_{ij}$ = largest frequency of a term in a given document.

References

1. Information Retrieval: Algorithms and Heuristics, David A. Grossman and Ophir Frieder. Springer, 2nd Edition, 2004.
2. BlueBit Matrix Calculator., BlueBit.
3. Using Linear Algebra for Intelligent Information Retrieval. SIAM Review 37(4): 573-595. Berry, M., S. Dumais, and G. O'Brien. (1995).
4. Understanding LSI via the Truncated Term-term Matrix, 2005 Thesis, by Regis Newo (Germany) .
5. A Framework for Understanding Latent Semantic Indexing (LSI) Performance, April Kontostathis and William Pottenger (Lehigh University).