Patterns in Unstructured Data

Discovery, Aggregation, and Visualization

A Presentation to the Andrew W. Mellon Foundation by

Clara Yu

John Cuadrado

Maciej Ceglowski

J. Scott Payne

National Institute for Technology and Liberal Education (NITLE)

INTRODUCTION - THE NEED FOR SMARTER SEARCH ENGINES

As of early 2002, there were just over two billion web pages listed in the Google search engine index, widely taken to be the most comprehensive. No one knows how many more web pages there are on the Internet, or the total number of documents available over the public network, but there is no question that the number is enormous and growing quickly. Every one of those web pages has come into existence within the past ten years. There are web sites covering every conceivable topic at every level of detail and expertise, and information ranging from numerical tables to personal diaries to public discussions. Never before have so many people had access to so much diverse information.

Even as the early publicity surrounding the Internet has died down, the network itself has continued to expand at a fantastic rate, to the point where the quantity of information available over public networks is starting to exceed our ability to search it. Search engines have been in existence for many decades, but until recently they have been specialized tools for use by experts, designed to search modest, static, well-indexed, well-defined data collections. Today's search engines have to cope with rapidly changing, heterogenous data collections that are orders of magnitude larger than ever before. They also have to remain simple enough for average and novice users to use. While computer hardware has kept up with these demands - we can still search the web in the blink of an eye - our search algorithms have not. As any Web user knows, getting reliable, relevant results for an online search is often difficult.

For all their problems, online search engines have come a long way. Sites like Google are pioneering the use of sophisticated techniques to help distinguish content from drivel, and the arms race between search engines and the marketers who want to manipulate them has spurred innovation. But the challenge of finding relevant content online remains. Because of the sheer number of documents available, we can find interesting and relevant results for any search query at all. The problem is that those results are likely to be hidden in a mass of semi-relevant and irrelevant information, with no easy way to distinguish the good from the bad.

Precision, Ranking, and Recall - the Holy Trinity

In talking about search engines and how to improve them, it helps to remember what distinguishes a useful search from a fruitless one. To be truly useful, there are generally three things we want from a search engine:

1. We want it to give us all of the relevant information available on our topic.
2. We want it to give us only information that is relevant to our search
3. We want the information ordered in some meaningful way, so that we see the most relevant results first.

The first of these criteria - getting all of the relevant information available - is called recall. Without good recall, we have no guarantee that valid, interesting results won't be left out of our result set. We want the rate of false negatives - relevant results that we never see - to be as low as possible.

The second criterion - the proportion of documents in our result set that is relevant to our search - is called precision. With too little precision, our useful results get diluted by irrelevancies, and we are left with the task of sifting through a large set of documents to find what we want. High precision means the lowest possible rate of false positives.

There is an inevitable tradeoff between precision and recall. Search results generally lie on a continuum of relevancy, so there is no distinct place where relevant results stop and extraneous ones begin. The wider we cast our net, the less precise our result set becomes. This is why the third criterion, ranking, is so important. Ranking has to do with whether the result set is ordered in a way that matches our intuitive understanding of what is more and what is less relevant. Of course the concept of 'relevance' depends heavily on our own immediate needs, our interests, and the context of our search. In an ideal world, search engines would learn our individual preferences so well that they could fine-tune any search we made based on our past expressed interests and pecadilloes. In the real world, a useful ranking is anything that does a reasonable job distinguishing between strong and weak results.

The Platonic Search Engine

Building on these three criteria of precision, ranking and recall, it is not hard to envision what an ideal search engine might be like:

- **Scope:** The ideal engine would be able to search every document on the Internet
- **Speed:** Results would be available immediately
- **Currency:** All the information would be kept completely up-to-date
- **Recall:** We could always find every document relevant to our query
- **Precision:** There would be no irrelevant documents in our result set
- **Ranking:** The most relevant results would come first, and the ones furthest afield would come last

Of course, our mundane search engines have a way to go before reaching the Platonic ideal. What will it take to bridge the gap?

For the first three items in the list - scope, speed, and currency - it's possible to make major improvements by throwing resources at the problem. Search engines can always be made more comprehensive by adding content, they can always be made faster with better hardware and programming, and they can always be made more current through frequent updates and regular purging of outdated information.

Improving our trinity of precision, ranking and recall, however, requires more than brute force. In the following pages, we will describe one promising approach, called latent

semantic indexing, that lets us make improvements in all three categories. LSI was first developed at Bellcore in the late 1980's, and is the object of active research, but is surprisingly little-known outside the information retrieval community. But before we can talk about LSI, we need to talk a little more about how search engines do what they do.

INSIDE THE MIND OF A SEARCH ENGINE

Taking Things Literally

If I handed you stack of newspapers and magazines and asked you to pick out all of the articles having to do with French Impressionism, it is very unlikely that you would pore over each article word-by-word, looking for the exact phrase. Instead, you would probably flip through each publication, skimming the headlines for articles that might have to do with art or history, and then reading through the ones you found to see if you could find a connection.

If, however, I handed you a stack of articles from a highly technical mathematical journal and asked you to show me everything to do with n-dimensional manifolds, the chances are high (unless you are a mathematician) that you would have to go through each article line-by-line, looking for the phrase "n-dimensional manifold" to appear in a sea of jargon and equations.

The two searches would generate very different results. In the first example, you would probably be done much faster. You might miss a few instances of the phrase French Impressionism because they occured in an unlikely article - perhaps a mention of a business figure's being related to Claude Monet - but you might also find a number of articles that were very relevant to the search phrase French Impressionism, even though they didn't contain the actual words: articles about a Renoir exhibition, or visiting the museum at Giverny, or the Salon des Refuss.

With the math articles, you would probably find every instance of the exact phrase n-dimensional manifold, given strong coffee and a good pair of eyeglasses. But unless you knew something about higher mathematics, it is very unlikely that you would pick out articles about topology that did not contain the search phrase, even though a mathematician might find those articles very relevant.

These two searches represent two opposite ways of searching a document collection. The first is a conceptual search, based on a higher-level understanding of the query and the search space, including all kinds of contextual knowledge and assumptions about how newspaper articles are structured, how the headline relates to the contents of an article, and what kinds of topics are likely to show up in a given publication.

The second is a purely mechanical search, based on an exhaustive comparison between a certain set of words and a much larger set of documents, to find where the first appear in the second. It is not hard to see how this process could be made completely automatic: it requires no understanding of either the search query or the document collection, just time and patience.

Of course, computers are perfect for doing rote tasks like this. Human beings can never take a purely mechanical approach to a text search problem, because human beings can't help but notice things. Even someone looking through technical literature in a foreign language will begin to recognize patterns and clues to help guide them in selecting candidate articles, and start to form ideas about the context and meaning of the search. But computers know nothing about context, and excel at performing repetitive tasks quickly. This rote method of searching is how search engines work.

Every full-text search engine, no matter how complex, finds its results using just such a mechanical method of exhaustive search. While the techniques it uses to rank the results may be very fancy indeed (Google is a good example of innovation in choosing a system for ranking), the actual search is based entirely on keywords, with no higher-level understanding of the query or any of the documents being searched.

John Henry Revisited

Of course, while it is nice to have repetitive things automated, it is also nice to have our search agent understand what it is doing. We want a search agent who can behave like a librarian, but on a massive scale, bringing us relevant documents we didn't even know to look for. The question is, is it possible to augment the exhaustiveness of a mechanical keyword search with some kind of a conceptual search that looks at the meaning of each document, not just whether or not a particular word or phrase appears in it? If I am searching for information on the effects of the naval blockade on the economy of the Confederacy during the Civil War, chances are high that a number of documents pertinent to that topic might not contain every one of those keywords, or even a single one of them. A discussion of cotton production in Georgia during the period 1860-1870 might be extremely revealing and useful to me, but if it does not mention the Civil War or the naval blockade directly, a keyword search will never find it.

Many strategies have been tried to get around this 'dumb computer' problem. Some of these are simple measures designed to enhance a regular keyword search - for example, lists of synonyms for the search engine to try in addition to the search query, or fuzzy searches that tolerate bad spelling and different word forms. Others are ambitious exercises in artificial intelligence, using complex language models and search algorithms to mimic how we aggregate words and sentences into higher-level concepts.

Unfortunately, these higher-level models are really bad. Despite years of trying, no one has been able to create artificial intelligence, or even artificial stupidity. And there is growing agreement that nothing short of an artificial intelligence program can consistently extract higher-level concepts from written human language, which has

proven far more ambiguous and difficult to understand than any of the early pioneers of computing expected.

That leaves natural intelligence, and specifically expert human archivists, to do the complex work of organizing and tagging data to make a conceptual search possible.

STRUCTURED DATA - EVERYTHING IN ITS PLACE

The Joys of Taxonomy

Anyone who has ever used a card catalog or online library terminal is familiar with structured data. Rather than indexing the full text of every book, article, and document in a large collection, works are assigned keywords by an archivist, who also categorizes them within a fixed hierarchy. A search for the keywords Khazar empire, for example, might yield several titles under the category Khazars - Ukraine - Kiev - History, while a search for beet farming might return entries under Vegetables - Postharvest Diseases and Injuries - Handbooks, Manuals, etc.. The Library of Congress is a good example of this kind of comprehensive classification - each work is assigned keywords from a rigidly constrained vocabulary, then given a unique identifier and placed into one or more categories to facilitate later searching.

While most library collections do not feature full-text search (since so few works in print are available in electronic form), there is no reason why structured databases can't also include a full-text search. Many early web search engines, including Yahoo, used just such an approach, with human archivists reviewing each page and assigning it to one or more categories before including it in the search engine's document collection.

The advantage of structured data is that it allows users to refine their search using concepts rather than just individual keywords or phrases. If we are more interested in politics than mountaineering, it is very helpful to be able to limit a search for Geneva summit to the category Politics-International-20th Century, rather than Switzerland-Geography. And once we get our result, we can use the classifiers to browse within a category or sub-category for other results that may be conceptually similar, such as Rejkyavik summit or SALT II talks, even if they don't contain the keyword Geneva.

You Say Vegetables::Tomato, I Say Fruits::Tomato

We can see how assigning descriptors and classifiers to a text gives us one important advantage, by returning relevant documents that don't necessarily contain a verbatim match to our search query. Fully described data sets also give us a view of the 'big picture' - by examining the structure of categories and sub-categories (or taxonomy), we can form a rough image of the scope and distribution of the document collection as a whole.

But there are serious drawbacks to this approach to categorizing data. For starters, there are the problems inherent in any kind of taxonomy. The world is a fuzzy place that sometimes resists categorization, and putting names to things can constrain the ways in which we view them. Is a tomato a fruit or a vegetable? The answer depends on whether you are a botanist or a cook. Serbian and Croatian are mutually intelligible, but have different writing systems and are spoken by different populations with a dim view of one another. Are they two different languages? Russian and Polish have two words for 'blue', where English has one. Which is right? Classifying something inevitably colors the way in which we see it.

Moreover, what happens if I need to combine two document collections indexed in different ways? If I have a large set of articles about Indian dialects indexed by language family, and another large indexed by geographic region, I either need to choose one taxonomy over the other, or combine the two into a third. In either case I will be re-indexing a lot of the data. There are many efforts underway to mitigate this problem - ranging from standards-based approaches like Dublin Core to rarefied research into ontological taxonomies (finding a sort of One True Path to classifying data). Nevertheless, the underlying problem is a thorny one.

One common-sense solution is to classify things in multiple ways - assigning a variety of categories, keywords, and descriptors to every document we want to index. But this runs us into the problem of limited resources. Having an expert archivist review and classify every document in a collection is an expensive undertaking, and it grows more expensive and time-consuming as we expand our taxonomy and keyword vocabulary. What's more, making changes becomes more expensive. Remember that if we want to augment or change our taxonomy (as has actually happened with several large tagged linguistic corpora), there is no recourse except to start from the beginning. And if any document gets misclassified, it may never be seen again.

Simple schemas may not be descriptive enough to be useful, and complex schemas require many thousands of hours of expert archivist time to design, implement, and maintain. Adding documents to a collection requires more expert time. For large collections, the effort becomes prohibitive.

Better Living Through Matrix Algebra

So far the choice seems pretty stark - either we live with amorphous data that we can only search by keyword, or we adopt a regimented approach that requires enormous quantities of expensive skilled user time, filters results through the lens of implicit and explicit assumptions about how the data should be organized, and is a chore to maintain. The situation cries out for a middle ground, some way to at least partially organize complex data without human intervention in a way that will be meaningful to human users. Fortunately for us, techniques exist to do just that.

LATENT SEMANTIC INDEXING

Taking a Holistic View

Regular keyword searches approach a document collection with a kind of accountant mentality: a document contains a given word or it doesn't, with no middle ground. We create a result set by looking through each document in turn for certain keywords and phrases, tossing aside any documents that don't contain them, and ordering the rest based on some ranking system. Each document stands alone in judgement before the search algorithm - there is no interdependence of any kind between documents, which are evaluated solely on their contents.

Latent semantic indexing adds an important step to the document indexing process. In addition to recording which keywords a document contains, the method examines the document collection as a whole, to see which other documents contain some of those same words. LSI considers documents that have many words in common to be semantically close, and ones with few words in common to be semantically distant. This simple method correlates surprisingly well with how a human being, looking at content, might classify a document collection. Although the LSI algorithm doesn't understand anything about what the words *mean*, the patterns it notices can make it seem astonishingly intelligent.

When you search an LSI-indexed database, the search engine looks at similarity values it has calculated for every content word, and returns the documents that it thinks best fit the query. Because two documents may be semantically very close even if they do not share a particular keyword, LSI does not require an exact match to return useful results. Where a plain keyword search will fail if there is no exact match, LSI will often return relevant documents that don't contain the keyword at all.

To use an earlier example, let's say we use LSI to index our collection of mathematical articles. If the words n-dimensional, manifold and topology appear together in enough articles, the search algorithm will notice that the three terms are semantically close. A search for n-dimensional manifolds will therefore return a set of articles containing that phrase (the same result we would get with a regular search), but also articles that contain just the word topology. The search engine understands nothing about mathematics, but examining a sufficient number of documents teaches it that the three terms are related. It then uses that information to provide an expanded set of results with better recall than a plain keyword search.

Ignorance is Bliss

We mentioned the difficulty of teaching a computer to organize data into concepts and demonstrate understanding. One great advantage of LSI is that it is a strictly mathematical approach, with no insight into the meaning of the documents or words it analyzes. This makes it a powerful, generic technique able to index any cohesive

document collection in any language. It can be used in conjunction with a regular keyword search, or in place of one, with good results.

Before we discuss the theoretical underpinnings of LSI, it's worth citing a few actual searches from some sample document collections. In each search, a red title or astrisk indicates that the document doesn't contain the search string, while a blue title or astrisk informs the viewer that the search string is present.

- In an AP news wire database, a search for Saddam Hussein returns articles on the Gulf War, UN sanctions, the oil embargo, and documents on Iraq that do not contain the Iraqi president's name at all.
- Looking for articles about Tiger Woods in the same database brings up many stories about the golfer, followed by articles about major golf tournaments that don't mention his name. Constraining the search to days when no articles were written about Tiger Woods still brings up stories about golf tournaments and well-known players.
- In an image database that uses LSI indexing, a search on Normandy invasion shows images of the Bayeux tapestry - the famous tapestry depicting the Norman invasion of England in 1066, the town of Bayeux, followed by photographs of the English invasion of Normandy in 1944.

In all these cases LSI is 'smart' enough to see that Saddam Hussein is somehow closely related to Iraq and the Gulf War, that Tiger Woods plays golf, and that Bayeux has close semantic ties to invasions and England. As we will see in our exposition, all of these apparently intelligent connections are artifacts of word use patterns that already exist in our document collection.

HOW LSI WORKS

The Search for Content

We mentioned that latent semantic indexing looks at patterns of word distribution (specifically, word co-occurence) across a set of documents. Before we talk about the mathematical underpinnings, we should be a little more precise about what kind of words LSI looks at.

Natural language is full of redundancies, and not every word that appears in a document carries semantic meaning. In fact, the most frequently used words in English are words that don't carry content at all: functional words, conjunctions, prepositions, auxilliary verbs and others. The first step in doing LSI is culling all those extraeous words from a document, leaving only content words likely to have semantic meaning. There are many ways to define a content word - here is one recipe for generating a list of content words from a document collection:

1. Make a complete list of all the words that appear anywhere in the collection

2. Discard articles, prepositions, and conjunctions
3. Discard common verbs (know, see, do, be)
4. Discard pronouns
5. Discard common adjectives (big, late, high)
6. Discard frilly words (therefore, thus, however, albeit, etc.)
7. Discard any words that appear in every document
8. Discard any words that appear in only one document

This process condenses our documents into sets of content words that we can then use to index our collection.

Thinking Inside the Grid

Using our list of content words and documents, we can now generate a term-document matrix. This is a fancy name for a very large grid, with documents listed along the horizontal axis, and content words along the vertical axis. For each content word in our list, we go across the appropriate row and put an 'X' in the column for any document where that word appears. If the word does not appear, we leave that column blank.

Doing this for every word and document in our collection gives us a mostly empty grid with a sparse scattering of X-es. This grid displays everthing that we know about our document collection. We can list all the content words in any given document by looking for X-es in the appropriate column, or we can find all the documents containing a certain content word by looking across the appropriate row.

Notice that our arrangement is binary - a square in our grid either contains an X, or it doesn't. This big grid is the visual equivalent of a generic keyword search, which looks for exact matches between documents and keywords. If we replace blanks and X-es with zeroes and ones, we get a numerical matrix containing the same information.

The key step in LSI is decomposing this matrix using a technique called singular value decomposition. The mathematics of this transformation are beyond the scope of this article (a rigorous treatment is available here), but we can get an intuitive grasp of what SVD does by thinking of the process spatially. An analogy will help.

Breakfast in Hyperspace

Imagine that you are curious about what people typically order for breakfast down at your local diner, and you want to display this information in visual form. You decide to examine all the breakfast orders from a busy weekend day, and record how many times the words bacon, eggs and coffee occur in each order.

You can graph the results of your survey by setting up a chart with three orthogonal axes - one for each keyword. The choice of direction is arbitrary - perhaps a bacon axis in the x direction, an eggs axis in the y direction, and the all-important coffee axis in the z direction. To plot a particular breakfast order, you count the occurence of each keyword,

and then take the appropriate number of steps along the axis for that word. When you are finished, you get a cloud of points in three-dimensional space, representing all of that day's breakfast orders.

If you draw a line from the origin of the graph to each of these points, you obtain a set of vectors in 'bacon-eggs-and-coffee' space. The size and direction of each vector tells you how many of the three key items were in any particular order, and the set of all the vectors taken together tells you something about the kind of breakfast people favor on a Saturday morning.

What your graph shows is called a term space. Each breakfast order forms a vector in that space, with its direction and magnitude determined by how many times the three keywords appear in it. Each keyword corresponds to a separate spatial direction, perpendicular to all the others. Because our example uses three keywords, the resulting term space has three dimensions, making it possible for us to visualize it. It is easy to see that this space could have any number of dimensions, depending on how many keywords we chose to use. If we were to go back through the orders and also record occurences of sausage, muffin, and bagel, we would end up with a six-dimensional term space, and six-dimensional document vectors.

Applying this procedure to a real document collection, where we note each use of a content word, results in a term space with many thousands of dimensions. Each document in our collection is a vector with as many components as there are content words. Although we can't possibly visualize such a space, it is built in the exact same way as the whimsical breakfast space we just described. Documents in such a space that have many words in common will have vectors that are near to each other, while documents with few shared words will have vectors that are far apart.

Latent semantic indexing works by projecting this large, multidimensional space down into a smaller number of dimensions. In doing so, keywords that are semantically similar will get squeezed together, and will no longer be completely distinct. This blurring of boundaries is what allows LSI to go beyond straight keyword matching. To understand how it takes place, we can use another analogy.

Singular Value Decomposition

Imagine you keep tropical fish, and are proud of your prize aquarium - so proud that you want to submit a picture of it to *Modern Aquaria* magazine, for fame and profit. To get the best possible picture, you will want to choose a good angle from which to take the photo. You want to make sure that as many of the fish as possible are visible in your picture, without being hidden by other fish in the foreground. You also won't want the fish all bunched together in a clump, but rather shot from an angle that shows them nicely distributed in the water. Since your tank is transparent on all sides, you can take a variety of pictures from above, below, and from all around the aquarium, and select the best one.

In mathematical terms, you are looking for an optimal mapping of points in 3-space (the fish) onto a plane (the film in your camera). 'Optimal' can mean many things - in this case it means 'aesthetically pleasing'. But now imagine that your goal is to preserve the relative distance between the fish as much as possible, so that fish on opposite sides of the tank don't get superimposed in the photograph to look like they are right next to each other. Here you would be doing exactly what the SVD algorithm tries to do with a much higher-dimensional space.

Instead of mapping 3-space to 2-space, however, the SVD algorithm goes to much greater extremes. A typical term space might have tens of thousands of dimensions, and be projected down into fewer than 150. Nevertheless, the principle is exactly the same. The SVD algorithm preserves as much information as possible about the relative distances between the document vectors, while collapsing them down into a much smaller set of dimensions. In this collapse, information is lost, and content words are superimposed on one another.

Information loss sounds like a bad thing, but here it is a blessing. What we are losing is noise from our original term-document matrix, revealing similarities that were latent in the document collection. Similar things become more similar, while dissimilar things remain distinct. This reductive mapping is what gives LSI its seemingly intelligent behavior of being able to correlate semantically related terms. We are really exploiting a property of natural language, namely that words with similar meaning tend to occur together.

LSI EXAMPLE - INDEXING A DOCUMENT

Putting Theory into Practice

While a discussion of the mathematics behind singular value decomposition is beyond the scope of our article, it's worthwhile to follow the process of creating a term-document matrix in some detail, to get a feel for what goes on behind the scenes. Here we will process a sample wire story to demonstrate how real-life texts get converted into the numerical representation we use as input for our SVD algorithm.

The first step in the chain is obtaining a set of documents in electronic form. This can be the hardest thing about LSI - there are all too many interesting collections not yet available online. In our experimental database, we download wire stories from an online newspaper with an AP news feed. A script downloads each day's news stories to a local disk, where they are stored as text files.

Let's imagine we have downloaded the following sample wire story, and want to incorporate it in our collection:

O'Neill Criticizes Europe on Grants
PITTSBURGH (AP)

Treasury Secretary Paul O'Neill expressed irritation Wednesday that European countries have refused to go along with a U.S. proposal to boost the amount of direct grants rich nations offer poor countries.
The Bush administration is pushing a plan to increase the amount of direct grants the World Bank provides the poorest nations to 50 percent of assistance, reducing use of loans to these nations.

The first thing we do is strip all formatting from the article, including capitalization, punctuation, and extraneous markup (like the dateline). LSI pays no attention to word order, formatting, or capitalization, so can safely discard that information. Our cleaned-up wire story looks like this:

o'neill criticizes europe on grants treasury secretary paul o'neill expressed irritation wednesday that european countries have refused to go along with a us proposal to boost the amount of direct grants rich nations offer poor countries the bush administration is pushing a plan to increase the amount of direct grants the world bank provides the poorest nations to 50 percent of assistance reducing use of loans to these nations

The next thing we want to do is pick out the content words in our article. These are the words we consider semantically significant - everything else is clutter. We do this by applying a stop list of commonly used English words that don't carry semantic meaning. Using a stop list greatly reduces the amount of noise in our collection, as well as eliminating a large number of words that would make the computation more difficult. Creating a stop list is something of an art - they depend very much on the nature of the data collection. You can see our full wire stories stop list here.
Here is our sample story with stop-list words highlighted:

o'neill criticizes europe on grants treasury secretary paul o'neill expressed irritation wednesday that european countries have refused to go along with a US proposal to boost the amount of direct grants rich nations offer poor countries the bush administration is pushing a plan to increase the amount of direct grants the world bank provides the poorest nations to 50 percent of assistance reducing use of loans to these nations

Removing these stop words leaves us with an abbreviated version of the article containing content words only:

> o'neill criticizes europe grants treasury secretary paul o'neill expressed irritation european countries refused US proposal boost direct grants rich nations poor countries bush administration pushing plan increase amount direct grants world bank poorest nations assistance loans nations

However, one more important step remains before our document is ready for indexing. Notice how many of our content words are plural nouns (grants, nations) and inflected verbs (pushing, refused). It doesn't seem very useful to have each inflected form of a content word be listed seperately in our master word list - with all the possible variants, the list would soon grow unwieldy. More troubling is that LSI might not recognize that the different variant forms were actually the same word in disguise. We solve this problem by using a stemmer.

Stemming

While LSI itself knows nothing about language (we saw how it deals exclusively with a mathematical vector space), some of the preparatory work needed to get documents ready for indexing is very language-specific. We have already seen the need for a stop list, which will vary entirely from language to language and to a lesser extent from document collection to document collection. Stemming is similarly language-specific, derived from the morphology of the language. For English documents, we use an algorithm called the Porter stemmer to remove common endings from words, leaving behind an invariant root form. Here are some examples of words before and after stemming:

```
information -> inform
presidency -> presid
presiding      -> presid
happiness -> happi
happily -> happi
discouragement -> discourag
battles -> battl
```

And here is our sample story as it appears to the stemmer:

> o'neill criticizes europe grants treasury secretary paul o'neill expressed irritation european countries refused US proposal boost direct grants rich nations poor countries bush administration pushing plan increase amount direct grants world bank poorest nations assistance loans nations

Note that at this point we have reduced the original natural-language news story to a series of word stems. All of the information carried by punctuation, grammar, and style is gone - all that remains is word order, and we will be doing away with even that by

transforming our text into a word list. It is striking that so much of the meaning of text passages inheres in the number and choice of content words, and relatively little in the way they are arranged. This is very counterintuitive, considering how important grammar and writing style are to human perceptions of writing.

Having stripped, pruned, and stemmed our text, we are left with a flat list of words:

```
administrat
amount
assist
bank
boost
bush
countri (2)
direct
europ
express
grant (2)
increas
irritat
loan
nation (3)
o'neill
paul
plan
poor (2)
propos
push
refus
rich
secretar
treasuri
US
world
```

This is the information we will use to generate our term-document matrix, along with a similar word list for every document in our collection.

THE TERM-DOCUMENT MATRIX

Doing the Numbers

As we mentioned in our discussion of LSI, the term-document matrix is a large grid representing every document and content word in a collection. We have looked in detail at how a document is converted from its original form into a flat list of content words. We prepare a master word list by generating a similar set of words for every document in our collection, and discarding any content words that either appear in every document (such words won't let us discriminate between documents) or in only one document (such words tell us nothing about relationships across documents). With this master word list in hand, we are ready to build our TDM.

We generate our TDM by arranging our list of all content words along the vertical axis, and a similar list of all documents along the horizontal axis. These need not be in any particular order, as long as we keep track of which column and row corresponds to which keyword and document. For clarity we will show the keywords as an alphabetized list.

We fill in the TDM by going through every document and marking the grid square for all the content words that appear in it. Because any one document will contain only a tiny subset of our content word vocabulary, our matrix is very sparse (that is, it consists almost entirely of zeroes).

Here is a fragment of the actual term-document marix from our wire stories database:

```
Document:  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  {
3000 more columns }



aa       0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  ...
amotd    0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  ...
aaliyah  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  ...
aarp     0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  ...
ab       0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  ...
...
zywicki  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  ...
```

We can easily see if a given word appears in a given document by looking at the intersection of the appropriate row and column. In this sample matrix, we have used ones to represent document/keyword pairs. With such a binary scheme, all we can tell about any given document/keyword combination is whether the keyword appears in the document.

This approach will give acceptable results, but we can significantly improve our results by applying a kind of linguistic favoritism called term weighting to the value we use for each non-zero term/document pair.

Not all Words are Created Equal

Term weighting is a formalization of two common-sense insights:

1. Content words that appear several times in a document are probably more meaningful than content words that appear just once.
2. Infrequently used words are likely to be more interesting than common words.

The first of these insights applies to individual documents, and we refer to it as local weighting. Words that appear multiple times in a document are given a greater local weight than words that appear once. We use a formula called logarithmic local weighting to generate our actual value.

The second insight applies to the set of all documents in our collection, and is called global term weighting. There are many global weighting schemes; all of them reflect the fact that words that appear in a small handful of documents are likely to be more significant than words that are distributed widely across our document collection. Our own indexing system uses a scheme called inverse document frequency to calculate global weights.

By way of illustration, here are some sample words from our collection, with the number of documents they appear in, and their corresponding global weights.

```
word     count     global weight


unit       833     1.44
cost       295     2.47
project    169     3.03
tackle     40      4.47
wrestler   7       6.22
```

You can see that a word like wrestler, which appears in only seven documents, is considered twice as significant as a word like project, which appears in over a hundred.

There is a third and final step to weighting, called normalization. This is a scaling step designed to keep large documents with many keywords from overwhelming smaller documents in our result set. It is similar to handicapping in golf - smaller documents are given more importance, and larger documents are penalized, so that every document has equal significance.

These three values multiplied together - local weight, global weight, and normalization factor - determine the actual numerical value that appears in each non-zero position of our term/document matrix.

Although this step may appear language-specific, note that we are only looking at word frequencies within our collection. Unlike the stop list or stemmer, we don't need any outside source of linguistic information to calculate the various weights. While weighting isn't critical to understanding or implementing LSI, it does lead to much better results, as it takes into account the relative importance of potential search terms.

The Moment of Truth

With the weighting step done, we have done everything we need to construct a finished term-document matrix. The final step will be to run the SVD algorithm itself. Notice that this critical step will be purely mathematical - although we know that the matrix and its contents are a shorthand for certain linguistic features of our collection, the algorithm doesn't know anything about what the numbers mean. This is why we say LSI is language-agnostic - as long as you can perform the steps needed to generate a term-document matrix from your data collection, it can be in any language or format whatsoever.

You may be wondering what the large matrix of numbers we have created has to do with the term vectors and many-dimensional spaces we discussed in our earlier explanation of how LSI works. In fact, our matrix is a convenient way to represent vectors in a high-dimensional space. While we have been thinking of it as a lookup grid that shows us which terms appear in which documents, we can also think of it in spatial terms. In this interpretation, every column is a long list of coordinates that gives us the exact position of one document in a many-dimensional term space. When we applied term weighting to our matrix in the previous step, we nudged those coordinates around to make the document's position more accurate.

As the name suggests, singular value decomposition breaks our matrix down into a set of smaller components. The algorithm alters one of these components (this is where the number of dimensions gets reduced), and then recombines them into a matrix of the same shape as our original, so we can again use it as a lookup grid. The matrix we get back is an approximation of the term-document matrix we provided as input, and looks much different from the original:

```
              a       b       c       d       e       f       g       h
i       j       k


aa       -0.0006 -0.0006  0.0002  0.0003  0.0001  0.0000  0.0000 -0.0001
0.0007  0.0001  0.0004 ...
amotd    -0.0112 -0.0112 -0.0027 -0.0008 -0.0014  0.0001 -0.0010  0.0004
-0.0010 -0.0015  0.0012 ...
```

```
aaliyah -0.0044 -0.0044 -0.0031 -0.0008 -0.0019  0.0027  0.0004  0.0014
-0.0004 -0.0016  0.0012 ...
aarp     0.0007  0.0007  0.0004  0.0008 -0.0001 -0.0003  0.0005  0.0004
0.0001  0.0025  0.0000 ...
ab      -0.0038 -0.0038  0.0027  0.0024  0.0036 -0.0022  0.0013 -0.0041
0.0010  0.0019  0.0026 ...
...
zywicki -0.0057  0.0020  0.0039 -0.0078 -0.0018  0.0017  0.0043 -0.0014
0.0050 -0.0020 -0.0011 ...
```

Notice two interesting features in the processed data:

- The matrix contains far fewer zero values. Each document has a similarity value for most content words.
- Some of the similarity values are negative. In our original TDM, this would correspond to a document with fewer than zero occurences of a word, an impossibility. In the processed matrix, a negative value is indicative of a very large semantic distance between a term and a document.

This finished matrix is what we use to actually search our collection. Given one or more terms in a search query, we look up the values for each search term/document combination, calculate a cumulative score for every document, and rank the documents by that score, which is a measure of their similarity to the search query. In practice, we will probably assign an empirically-determined threshold value to serve as a cutoff between relevant and irrelevant documents, so that the query does not return every document in our collection.

The Big Picture

Now that we have looked at the details of latent semantic indexing, it is instructive to step back and examine some real-life applications of LSI. Many of these go far beyond plain search, and can assume some surprising and novel guises. Nevertheless, the underlying techniques will be the same as the ones we have outlined here.

APPLICATIONS

What Can LSI Do For Me Today?

Throughout this document, we have been presenting LSI in its role as a search tool for unstructured data. Given the shortcomings in current search technologies, this is undoubtedly a critical application of semantic indexing, and one with very promising results. However, there are many applications of LSI that go beyond traditional information retrieval, and many more that extend the notion of what a search engine is, and how we can best use it. To illustrate this, here are just a few examples of the areas where exciting work is happening (or should be happening) with LSI:

- **Relevance Feedback**

  Most regular search engines work best when searching a small set of keywords, and very quickly decline in recall when the number of search terms grows high. Because LSI shows the reverse behavior (the more it knows about a document, the better it is at finding similar ones), a latent semantic search engine can allow a user to create a 'shopping cart' of useful results, and then go out and search for futher results that most closely match the stored ones. This lets the user do an iterative search, providing feedback to guide the search engine towards a useful result.

- **Archivist's Assistant**

  In introducing LSI we contrasted it with more traditional approaches to structuring data, including human-generated taxonomies. Given LSI's strength at partially structuring unstructured data, the two techniques can be used in tandem. This is potentially a very powerful combination - it would allow archivists to use their time much more efficiently, enhancing, labeling and correcting LSI-generated categories rather than having to index every document from scratch. In the next section, we will look at a data visualization approach that could be used in conjunction with LSI to create a sophisticated, interactive application for archivist use.

-
- **Automated Writing Assessment**

  By comparing student writing against a large data set of stored essays on a given topic, LSI tools can analyze submitted assignments and highlight content areas that the student essay didn't cover. This can be used as a kind of automated grading system, where the assignment is compared to a pool of essays of known quality, and given the closest matching grade. We believe a more appropriate use of the technology is a feedback tool to guide the student in revising his essay, and

suggest directions for further study.
{ More info and demo: http://www-psych.nmsu.edu/essay/ }

- **Textual Coherence:**

  LSI can look at the semantic relationships within a text to calculate the degree of topical coherence between its constituent parts. This kind of coherence correlates well with readability and comprehension, which suggests that LSI might be a useful feedback tool in writing instruction (along the lines of existing readability metrics).
  { source: http://www.knowledge-technologies.com/papers/abs-dp2.foltz.html }

- **Information Filtering:**

  LSI is potentially a powerful customizable technology for filtering spam (unsolicited electronic mail). By training a latent semantic algorithm on your mailbox and known spam messages, and adjusting a user-determined threshold, it might be possible to flag junk mail much more efficiently than with current keyword based approaches. The same may apply to common Microsoft Outlook computer viruses, which tend to share a basic structure.
  LSI could also be used to filter newsgroup and bulletin board messages. { source: http://www-psych.nmsu.edu/~pfoltz/cois/filtering-cois.html }

MULTI-DIMENSIONAL SCALING

The Big Picture

In our discussion of information retrieval, we have been talking about searches that give textual results - we enter queries and phrases as text, and look at results in the form of a list. In this section, we will be looking at an annex technology to LSI called multi-dimensional scaling, or MDS, which lets us visualize complex data and interact with it in a graphical environment. This approach lets us use the advanced pattern-recognition software in our visual cortex to notice things in our data collection that might not be apparent from a text search.

Multi-dimensional scaling, or MDS, is a method for taking a two- or three-dimensional 'snapshot' of a many-dimensional term space, so that dimensionally-challenged human beings can see it. Where before we used singular value decomposition to compress a large term space into a few hundred dimensions, here we will be using MDS to project our term space all the way down to two or three dimensions, where we can see it.

The reason for using a different method (and another three-letter acronym) has to do with how the two algorithms work. SVD is a perfectionist algorithm that finds the best projection onto a given space. Finding this projection requires a lot of computing horsepower, and consequently a good deal of time. MDS is a much faster, iterative approach that gives a 'good-enough' result close to the optimum one we would get from SVD. Instead of deriving the best possible projection through matrix decomposition, the MDS algorithm starts with a random arrangement of data, and then incrementally moves it around, calculating a stress function after each perturbation to see if the projection has grown more or less accurate. The algorithm keeps nudging the data points until it can no longer find lower values for the stress function. What this produces is a scatter plot of the data, with dissimilar documents far apart, and similar ones closer together. Some of the actual distances may be a little bit off (since we are using an approximate method), but the general distribution will show good agreement with the actual similarity data.

To show what this kind of projection looks like, here is an example of an MDS graph of approximately 3000 AP wire stories from February, 2002. Each square in the graph represents a one-page news story indexed using the procedures described earlier:

As you can see, the stories cluster towards the center of the graph, and thin out towards the sides. This is not surprising given the nature of the document collection - there tend to be many news stories about a few main topics, with a smaller number of stories about a much wider variety of topics.

The actual axes of the graph don't mean anything - all that matters is the relative distance between any two stories, which reflects the semantic distance LSI calculates from word co-occurence. We can estimate the similarity of any two news stories by eyeballing their position in the MDS scatter graph. The further apart the stories are, the less likely they are to be about the same topic.

Note that our distribution is not perfectly smooth, and contains 'clumps' of documents in certain regions. Two of these clumps are highlighted in red and blue on the main MDS graph. If we look at the articles within the clumps, we find out that these document clusters consist of a set of closely related articles. For example, here is the result of zooming in on the blue cluster from Figure 1:

As you can see, nearly all of the articles in this cluster have to do with the Israeli-Palestinian conflict. Because the articles share a great many keywords, they are bunched

together in the same semantic space, and form a cluster in our two-dimensional projection. If we examine the red cluster, we find a similar set of related stories, this time about the Enron fiasco:

Notice that these topic clusters have formed spontaneously from word co-occurence patterns in our original set of data. Without any guidance from us, the unstructured data collection has partially organized itself into categories that are conceptually meaningful. At this point, we could apply more refined mathematical techniques to automatically detect boundaries between clusters, and try to sort our data into a set of self-defined categories. We could even try to map different clusters to specific categories in a taxonomy, so that in a very real sense unstructured data would be organizing itself to fit an existing framework.

This phenomenon of clustering is a visual expression in two dimensions of what LSI does for us in a higher number of dimensions - reveals preexisting patterns in our data. By graphing the relative semantic distance between documents, MDS reveals similarities on the conceptual level, and takes the first step towards organizing our data for us in a useful way.

PUTTING MDS AND LSI TO WORK

Creative Approaches

Much of the work in applied MDS has come from the fields of advertising and cognitive psychology (where it is also known as perceptual mapping). Researchers in both fields use the technique to transform questionnaires about relative preferences and similarities into a visual representation using the scaling techniques we have outlined. These techniques do not appear to have been applied to linguistic data until relatively recently.

This illustrates a common theme in latent semantic research - combining familiar techniques from different disciplines in a novel way to tackle problems in data retrieval. This kind of creative juxtaposition is one of the things that makes LSI interesting to work on, and levels the playing field between major research institutions and liberal arts colleges. One does not need an enormous supercomputer or advanced mathematical knowledge to do interesting work with these techniques. In fact, because LSI research draws on pure and applied mathematics, linguistics, computer science, psychology, information retrieval, and the social sciences, what really matters is breadth of knowledge. There are likely to be connections further afield that remain to be discovered.

With this eclectic background in mind, here are some potential applications of semantic indexing coupled with MDS data visualization:

1. **Archive Management Tools:**

   We already mentioned the potential use of LSI as an archivist's assistant, using LSI to highlight content patterns in a data collection, and more traditional taxonomies to formalize and heighten those patterns. One intuitive method for creating such tools is to display data visually using MDS, and allow for human feedback. An interactive program using multi-dimensional scaling would allow an archivist to graphically manipulate data, draw boundaries between clusters, examine content relationships and add classifiers using an intuitive, click-and-drag type interface. What's more, different expert users would be able to use MDS to generate their own personal view of a data set, and then reconcile or combine those views.

2. **Concept Maps:**

   Concept maps take this notion of interactivity and classification further, letting users manipulate and edit LSI-generated views of a data collection to produce a spatial map of topics and concepts. By drawing connections between items and moving them around, users can create their own view of a data collection. These views can be 'untangled' using mathematical techniques to create clear, visually direct concept maps. These maps can be shared, combined, and compared with others, making a unique pedagogical or research tool.

3. **Bioinformatics:**

   The same LSI techniques we use to find similarities in language have enormous potential in the field of bioinformatics. Both DNA and protein molecules consist of long strings of biochemical 'words'. Finding and understanding patterns in those words is one of the major research problems in modern biology. Using the tools we describe would make it possible to detect and visualize such patterns, and conduct important basic research in this nascent field.

Further Reading

LSI Journal Articles

The following are journal articles relevant to the material covered in this overview. We welcome suggestions for expanding this list

.

1. Dumais, S. T., Landauer, T. K. and Littman, M. L. (1996) "Automatic cross-linguistic information retrieval using Latent Semantic Indexing." In SIGIR'96 -

Workshop on Cross-Linguistic Information Retrieval, pp. 16-23, August 1996.
Available online
2. Foltz, P. W., Kintsch, W., and Landauer, T. K. (1998). The Measurement of Textual Coherence with Latent Semantic Analysis. Discourse Processes, 25, 285-307.
Available online
3. Foltz, P. W. (1990) "Using Latent Semantic Indexing for Information Filtering". In R. B. Allen (Ed.) Proceedings of the Conference on Office Information Systems, Cambridge, MA, 40-47.
Available online
4. Landauer, T. K., Foltz, P. W., and Laham, D. (1998). Introduction to Latent Semantic Analysis. Discourse Processes, 25, 259-284.
Available online
5. Landauer, T. K. and Dumais, S. T. (1977) - html only, "Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction and Representation of Knowledge." Psychological Review, 1997, 104 (2), 211-240.
Available online
6. Person, N. K., Graesser, A. C., Bautista, L., Mathews, E. C., & the Tutoring Research Group (2001). Evaluating Student Learning Gains in Two Versions of AutoTutor. In J. D. Moore, C. L. Redfield, & W. L. Johnson (Eds.) Artificial intelligence in education: AI-ED in the wired and wireless future (pp. 286-293). Amsterdam, IOS Press.
Available online
7. Rehder, B., Schreiner, M. E., Wolfe, M. B., Laham, D., Landauer, T. K., & Kintsch, W. (1998). Using Latent Semantic Analysis to assess knowledge: Some technical considerations. Discourse Processes, 25, 337-354.
8. Robinson, C.S. & Burgess, C. Vocabulary Performance of HAL and LSA Using a Standardized Performance Measure. Society for Computers in Psychology, November 15, 2001.
Available online

LSI Websites

Links to websites with comprehensive bibliographies on LSI and related techniques

.

1. Latent Semantic Indexing Website (University of Tennessee). Includes links to sites by Michael Berry and Susan Dumais, both of whom have done extensive work in LSI.
2. Telcordia Technologies LSI links page. Comprehensive list of articles. Telcordia holds a patent in LSI.
3. Knowledge Analysis Technologies. Company that makes an LSI product for automated essay assessment.
4. AuthorTutor. Papers related to AuthorTutor.