Assignment 2: Latent Semantic Indexing (LSI)
CS430 - Information Retrieval

Michael Metral (mdm257)

October 2, 2007

1 How To Run

1. Unpack the zipped file a2.zip

Command Line:
2. Direct the command line to the a2/Code directory
3. Type: *python lsi.py*

Directory Icon:
2. Open the directory a2/Code
3. Double-Click lsi.py

Searching:
4. After the index is built, enter a query to search for. i.e. "control," "solar system," "nasa" etc.

Note:
-The program is written in Python 2.5
-The program uses NumPy, a matrix package for Python. The download can be found at http://numpy.scipy.org/ under the side option, "Download NumPy"

2 Mathematical Basis

2.1 Singular Value Decomposition (SVD) on Index Matrix

As the latent structure used to produce the LSI, the SVD decomposes a *Term x Document* matrix, extracted from the corpus' index, into the product of 3 matrices: T, S and D_t (transposed), where T and D_t are respectively the left and right singular vectors and S is the diagonal matrix of singular values (accordingly, the eigenvectors and eigenvalues of a *M x N* matrix).

With these matrices in hand and a given k dimensions used for reduction, we then trim the matrices to the desired dimensions:

T becomes T_k, a matrix of size *Terms x k*
S becomes S_k, a matrix of size *k x k*
D_t becomes D_{tk}, a matrix of size *k x Documents*

2.2 D_k*S_k Matrix for Document Referral

Once the 3 matrices have been decomposed from the *Term x Document* matrix, we can use the D_k*S_k matrix, where D_k is the transposition of D_{tk}, to create a special space. This space allows for the rows of newly formed matrix D_k*S_k to serve as a set of coordinates for the documents; thus, each row of the D_k*S_k matrix correlates to a document in the corpus and each column represents a single dimension in the total amount of dimensions used to reduce the original *Term x Document* matrix.

2.3 Cosine Similarity between Query and D_k*S_k Matrix

With D_k*S_k defined, we can use each cell of this particular matrix in the dot products of a cosine angle computation to produce the similarity between a document and the query.

However, we cannot directly compute the dot-products with the query as a list of terms so it must be constructed into a vector. Once a vector is formed we can use the matrices returned from the SVD to compute the query in the reduced amount of k dimensions used in the similarity comparison.

The transformation equation:

$$Q_k = Q_v * T_k * {S_k}^{-1} \tag{1}$$

It is worthy to note that the use of the D_k*S_k matrix, in this particular assignment, allows a reduction of the original *Term x Document* matrix from a size of 3037 x 20 to a incredibly smaller matrix of size 20 x k, where k is at most 20 dimensions; thus, reducing the similarity computations by using $< 1\%$ of the original *Term x Document* matrix.

3 NumPy Package of Matrix Functions for Python

3.1 Singular Value Decomposition (SVD)

Performing the SVD on a matrix allows the creation of the 3 matrices T, S and D_t used through out the production of the LSI.

$$T,\ S,\ D_t = numpy.linalg.svd(X)$$

X is a matrix of size Term x Document

3.2 Matrix Multiplication

Matrix multiplication is used in two instances through out this program. Initially it is used to produce the D_k*S_k matrix of reduced document dimensions and lastly to when calculating the dot products between the query and a cell from the D_k*S_k matrix.

$$Z = numpy.dot(X,\ Y)$$

X and Y are matrices that follow the necessary rules for matrix multiplication

3.3 Transpose and Inverse

To compute the D_k*S_k matrix we must transpose the matrix returned from the SVD function of the original *Term x Document* after trimming the matrix down to the reduced k dimensions: D_{tk}.

$$D_k = numpy.transpose(D_{tk})$$

When computing the reduced query Q_k through the use of the transformation equation, Eq.1, we must use the inverse of S_k to produce ${S_k}^{-1}$.

$${S_k}^{-1} = numpy.linalg.inv(S_k)$$

3.4 Zeroing Matrices

When working with the query vector and the index, matrices need to be created. With zeroing, when a matrix is created it initially has all its cells filled with 0's. This allows us to only fill in the frequencies for the terms in a query vector and for certain documents that a term appears in within the index.

$$X = numpy.zeros.((M,\ N),\ int)$$

M and N are respectively the sizes of the rows and columns of the newly created matrix, X

4 Self-Implemented Functions

4.1 Eigenvalue Lists to Diagonal Matrix

When numpy is used to compute the SVD, the matrix of singular values (eigenvalues) are placed into a list. This list is then taken and transformed into a matrix of size *Documents x Documents.*

4.2 Matrix Trimming

In order to reduce the amount of computing when performing the similarity comparisons, we must trim the matrices, T, S and D_t down to the sizes previously listed using k dimensions appropriately.

4.3 Query Vector Creation

When a query is given, it is taken in by the program as a list. This list must be converted to a vector of size *Terms x 1* in order to perform the necessary functions through out the LSI process.

4.4 Cosine Similarity

Finally, when Q_k and D_k*S_k are calculated the cosine angle must be computed to compare their similarities.

In order to make the program efficient the concept space D_k*S_k is run through a function to calculate each documents (a row in the matrix) magnitude (length) to aid in the normalizing when computing the similarity. This is stored in a dictionary for future reference.

When actually computing the cosine angle, all that is left to do is compute the magnitude of the reduced query Q_k and perform the normalized dot products with each row in the concept space, D_k*S_k.

5 Tests and Results

5.1 Synonymy

Synonymy: There are many ways to refer to the same object. For example, the words "immense" and "huge" are synonyms of each other referring to the definition of an object that is extremely big.

When LSI is implemented, synonymy is said to improve.

In the following test, the words "immense" and "huge" were used as separate queries and tested in 3 different k dimensions, 2, 5 and 15, with a cosine similarity threshold $\geq$ than .85. A hit of 1 means the document was returned as a hit, a 0 means otherwise.

As seen from Figure 1., we can compare both synonyms by the results produced for a certain dimension, i.e. compare Immense-2 with Huge-2 to see that both produced a hit in Document 1 when 2 dimensions were used.

In this specific example with 2 dimensions, "immense" and "huge" produce relevant results for Documents 1, 9 and 13, when each only appear in Document 1 and 17 respectively. This shows us that synonymy in fact is working when LSI is implemented and that precision is very likely to increase.

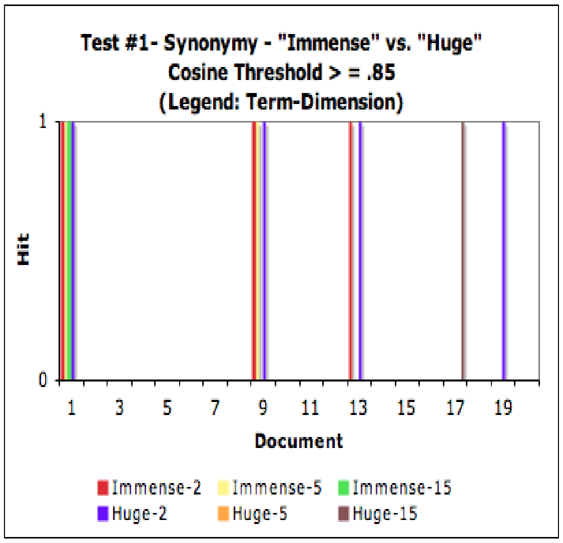

Figure 1: Synonymy Graph of the terms "Immense" and "Huge."

5.2 Polysemy

Polysemy: Most words have more than one distinct meaning. For example, the term "down" in the corpus has 4 different meanings: to stop, decrease, position and pinpoint.

In the following test, the word "down" was used as a query and tested in 3 different k dimensions, 2, 5 and 15, with a cosine similarity threshold $\geq$ than .85. A hit of 1 means the document was returned as a hit, a 0 means otherwise.

As seen from Figure 2., we can compare the various meanings by the results produced for a certain dimension, i.e. compare To Stop - Index and To Stop - Returned. In this test we are attempting to see if we were returned a hit with the LSI implementation over all 3 dimensions.

We can see from the instance of "down", to stop,

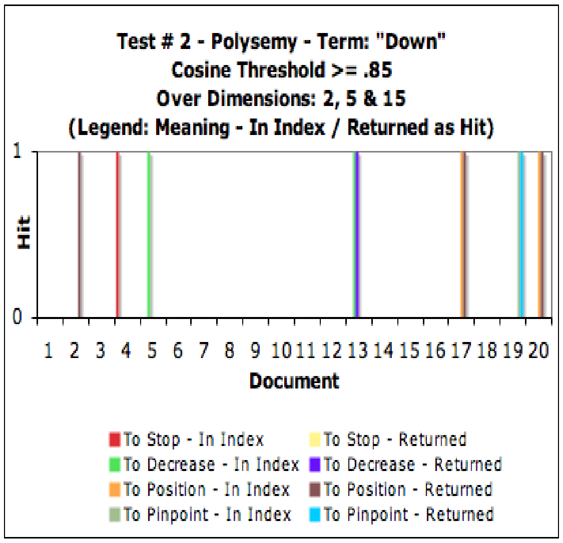

Figure 2: Polysemy Graph of the term "Down".

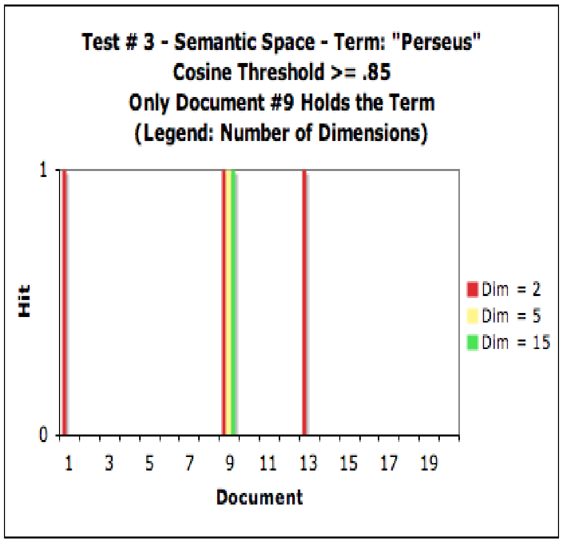

Figure 3: Semantic Space of the term "Perseus".

located in the index, To Stop - In Index, which lies in Document 4, that none of the dimensions returned a hit for To Stop - Returned.

Some of the meanings do in fact return similar document hits, but the average for this word and its 4 meanings was 50%. Thus, polysemy is not as successful as users would like, because it lowers precision while increasing recall, which is a known fact of LSI.

5.3 Semantic Space

Semantic Space: A vector space where terms and documents that are closely associated are placed near one another but not necessarily including the term. For example, a document that discusses the effects of coffee and soda is highly associated with the term "beverage" since they are both beverages, regardless of the document not including the term itself.

In the following test, the word "perseus" was used as a query and tested in 3 different k dimensions, 2, 5 and 15, with a cosine similarity threshold $\geq$ than .85. A hit of 1 means the document was returned as a hit, a 0 means otherwise.

As seen from Figure 3., we can compare the relevant documents returned as hits for the term "perseus," which only appears in one location in Document 9.

Despite "perseus" only appearing once in one document, the 3 dimensions returned other documents, i.e. 1 and 3. This demonstrates that Documents 1 and 3 have a strong association with the term regardless of term not being located within either one of them and LSI is capable of associating the two.

We should point out that at times the documents might have nothing to do with the term itself, but this is a slight problem that LSI faces.

5.4 Co-Occurring Words

Co-Ocurring Words: Two terms that almost always occur together. For example "black hole," or "solar system".

In the following test, the words "black hole," "black" and "hole" were used as a query and tested in 3 different k dimensions, 2, 5 and 15, with a cosine similarity threshold $\geq$ than .85. A hit of 1 means the document was returned as a hit, a 0 means otherwise.

As seen from Figure 4., we can compare the rele-

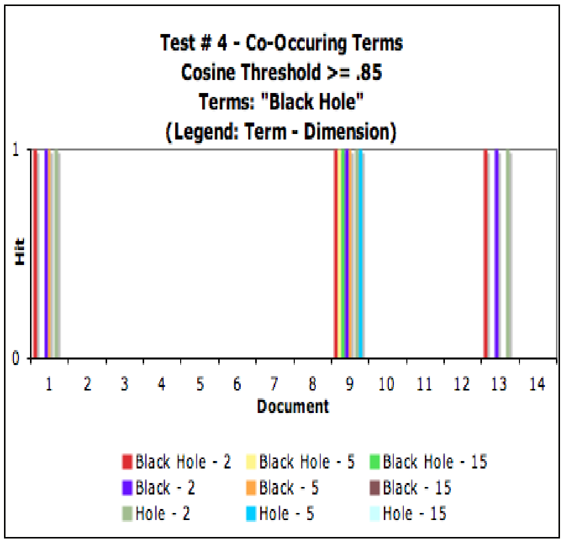

Figure 4: Co-Occuring Terms Graph of "Black Hole," "Black" and "Hole"

vant documents returned as hits for the 3 terms by dimension, i.e. compare "Black Hole - 2," "Black - 2" and "Hole - 2" to see the returned document hits for 2 dimensions. By comparing the 3 terms in the 2 dimensions, we can observer that all 3 terms return Documents 1, 9 and 13. This example along with the other columns in the graph demonstrate that co-occurring terms and the individual, independent terms return similar results; thus, demonstrating that LSI is capable of associating all 3 terms with each other.

6 Dimension Selection

After viewing the various results from all 4 tests and from searching queries it is evident that the greater the amount of dimensions, the less that similarities between queries and documents occurred. Also, the higher the cosine similarity threshold was set, the less the documents retrieved were displayed as hits.

As to achieve a balance of the two factors, a setting of 2 dimensions and a cosine threshold of .75 were chosen in order to return enough similar and relevant documents to the user.

7 Extra Features

As extras, the results returned have a limit of 10 hits. If a document has ≥ 10 hits, only 10 will be displayed; however, if a document has ≤ 10 documents, whatever the total may be, is displayed. These hits are also ordered by rank. In other words, the most similar/relevant documents are placed at the beginning of the results, leaving the dissimilar results towards the bottom of the results.