# Text Mining and Information Retrieval using SVD in LSI (Latent Semantic Indexing)

## Introduction

Latent semantic indexing (LSI) or Latent Semantic Analysis(LSA) is an indexing and information retrieval method. It is one of the major analysis approaches in the field of text mining. It helps in finding out the documents which are most relative with the specified keyword. Similarly it also helps the search engines to give most appropriate results for the search query.

## Linear Algebra Concepts Used:

### Singular Value Decomposition

SVD provides the statistical basis for text mining and classification methods usually referred to as latent semantic indexing in conjunction with text mining. In SVD, matrix A is normally a word x document matrix, which represents the paper and textual as a high-dimensional vector space model, also called a hyperspace document representation.

- Documents are shown as V rows.
- The similarity of documents can be calculated by analysis of the VS rows.
- Words are shown as U rows.
- The similarity of terms can be defined by analysing the rows of the US matrix.

### Frobenius norm

The Frobenius norm of a matrix is nothing but the square root of the sum of squares of its elements. It is also known as the Euclidean norm. In the case of LSI, it is used to compare how different two matrices are. Let's say that the matrices we want to compare are A1 and A2. This can be done so by computing the Frobenius norm of (A1 - A2) - (F1). Then we compute the Frobenius norm of A1 (F2). We divide both of them (F1/F2) and according to the answer we get, we can say how different those two matrices are. This answer will always be between 0 and 1. The lesser the answer, the lesser the difference between A1 and A2. (For better understanding: as if A1 = A2, our answer will be 0 since F1 will be equal to 0)

## Jacobi eigenvalue theorem

- It is an iterative method to find eigenvalues and eigenvectors of symmetric matrices.It is based on the series of rotations.Here we apply similarity transformations on a matrix such that the given matrix gets converted into a diagonal matrix. Diagonal elements of the final diagonal matrix will be the approximation of eigenvalues of the original given matrix.
  Find maximum element of matrix let it have index i,j then we have to make another matrix
  → Givens Rotation:

Here c = cos(theta) and s = sin(theta)

$$G(i,j,\text{theta})= \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & c & -s & 0 \\ 0 & s & c & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

g*ii*=c

g*jj*=-s

g*ij*=s if(i>j)

$$\tan(2*\text{theta})= \frac{2 S_{ij}}{S_{jj}-S_{ii}}$$

**Step 1**. Symmetric matrix A.
**Step 2.** Initialize D = A and S = I, an identity matrix.

**Step 3.**To find largest non-diagonal matrix from D matrix.
$D=[d_{ij}]$

**Step 4 .**Find the rotational angle .
If dii = djj then
if dij > 0 → θ =π/4 → because- denominator of tan(2theta) will be 0 → theta=pi/2
If $d_{ij}$<0 → θ = −π/4→ -pi/2

Else

$\theta = \dfrac{\tan^{-1}(2d_{ij})}{d_{ii}-d_{jj}}$

Endif;

– **Step 5.** Compute the transformation matrix S1 = [$s_{pq}$]
Set $s_{pq}$ = 0 for all p, q = 1, 2, ..., n
$s_{kk}$ = 1, k = 1, 2, ..., n(To take all the diagonal elements as 1)
and $s_{ii}$ = $s_{jj}$ = cos , $s_{ij}$ = sin , $s_{ji}$ = sin.
– **Step 6.** : $D=S_1^TDS_1$ perform $S=SS_1$
– **Step 7** Repeat steps 3 to 6 until D becomes diagonal.
– **Step 8**. Diagonal elements of D are the approximation of eigenvalues and the columns of S are the corresponding eigenvectors.

## Approach:

To perform LSI, we first load and preprocess the text data. We remove all punctuations from it and store it in a python list. Then we tokenized each string in this list. After that, we found unique words in each string and stored all of these words in a wordset. This wordset is a set of unique words of all documents.

Now, we will find the frequency of all words (in the wordset) in each document one by one. We measured term frequency by (count of word)/(total words in respective document) for all the documents. After that, we stored the IDF value of all words in a dictionary called idfs. Then, we multiplied the term frequency of words in the respective documents with the IDF value of each word. This way, we measured the significance of each word in that document.

We stored these values in a dataframe called X (you can see the second image in STEP 2 for a better understanding). Rows of X transpose are words and columns are document numbers. We converted this dataframe into a matrix and performed SVD on it.

We iterated from 1 to min(m,n) where (m = number of rows, n = number of columns) singular values and chose the first i (iterator) number of singular values. Then, we reconstructed the l_2d matrix by using the first i singular values. As we reduce the number of singular values, the reconstructed matrix we get gets more and more compressed.

After deciding a threshold for the Frobenius norm, we compared the original matrix (l_2d) and the reconstructed matrix. If the value returned by the frobenius function of two matrices is less than this threshold, we will use the reconstructed matrix for the search.

For the search function, we simply ask for an input. If any word of that input is not present in any of the documents, we say that no matching documents were found. If it is found, we calculate the score of each document (a higher score means that the document is more relevant). This is done so by using the reconstructed matrix we get from running the functions above.

## Coding and Simulation:

Types of modules and libraries used.

1. Matplotlib
1. Numpy
2. Pandas
3. NLTK

**NOTE:**

For implementation of this program, we have minimized the use of libraries. We use our own function for transpose of a matrix, multiplication of a matrix and SVD of a matrix as well. The SVD function takes O(n^4) time to return the decomposed matrices in the worst case. Due to this when we pass a m*n matrix, if m is greater than 100, the program takes a lot of time to get executed. That is why we created our own documents containing fewer words.

However, upon using functions like np.linalg.svd() and np.dot(), we were able to successfully compare and rank 50-60 documents (each containing approximately 75 words). This is a drastic boost to the program which we made from scratch.
So, we have prepared two programs, **one entirely from scratch** and **one in which we have used libraries**.

## STEP-1: Loading and Preprocessing data in text format

We loaded the documents from SciKit learn fetch_20newsgroup dataset.
Through the fetch_20newsgroups() function itself, we removed all footers and quotes.
We made two functions to remove punctuations and numbers respectively from the data.

These two functions take a string as input. Then they traverse through the string and if s[i] is not in punctuation (something imported from the string library) or not s[i].isdigit(), we didn't add s[i] to the output string
Else add s[i] to the output string

Using nltk.tokenize.wordpunct_tokenize(), we tokenized the first 'n_docs' number of documents and stored them in all_docs

all_docs is sort of a 2D list in whose row a list of words of ith document are stored.

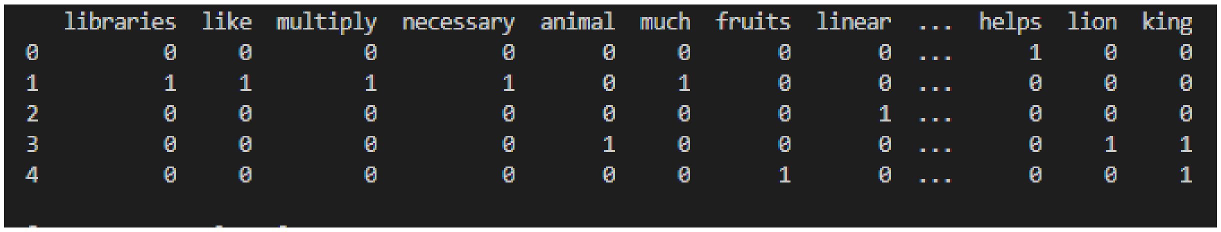
'Bow' is an abbreviation of 'Bag of Words'. We traverse through all docs and remove all words which are 'stopwords'. Then we remove all the null values using filter()

Now, we made all words inside bow unique by passing them through the unique1() function

Then we made a wordset (a set of unique words) of bow. Worddict is a list of dictionaries. It is initialized in such a way that all words of wordset are the keys of each dictionary and each key has value 0

## STEP-2: Creating TF-IDF matrix

We use a function called term_document_matrix() to find the number of times a particular word appears in each document. It returns a pandas DataFrame of worddict after carrying out the necessary operations. It consists of words as columns and document numbers as rows. See the image below for better understanding.



```
   libraries  like  multiply  necessary  animal  much  fruits  linear  ...  helps  lion  king
0          0     0         0          0       0     0       0       0  ...      1     0     0
1          1     1         1          1       0     1       0       0  ...      0     0     0
2          0     0         0          0       0     0       0       1  ...      0     0     0
3          0     0         0          0       1     0       0       0  ...      0     1     1
4          0     0         0          0       0     0       1       0  ...      0     0     1
```

Figure 1. Pandas dataframe containing count of words in each document

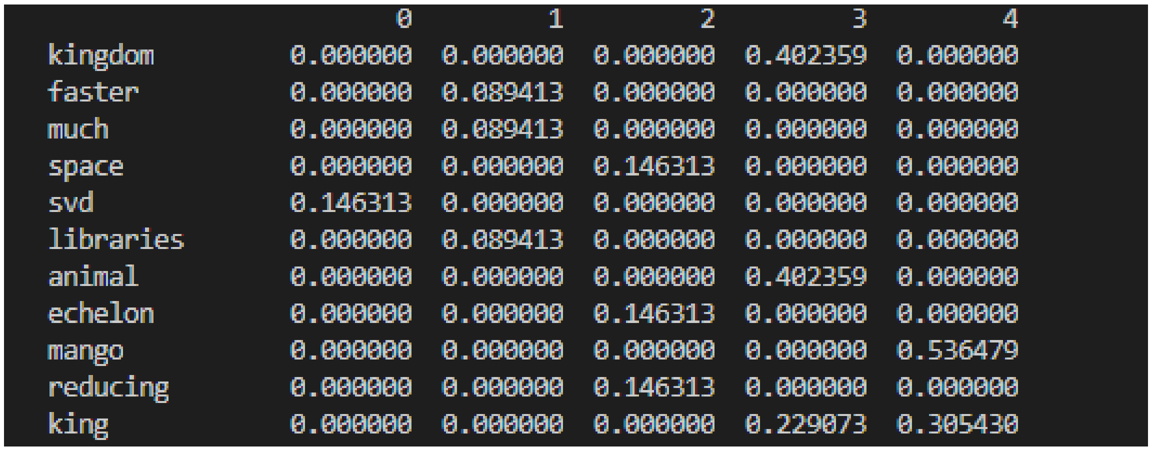
Next, we use the term_freq() function to find the importance of a particular word in the document it occurs in. It returns a dictionary containing key value pairs of word and its importance respectively.   (Here importance means: (count of word)/(total words in a document))

We computed term_freq() on all documents and stored them in a list called tfbow (term frequency of bag of words)

Next, we made a function called idf() to make a tf-idf matrix (term frequency inverse document frequency). To calculate the IDF value we use the formula: math.log(n/float(val)) Where n is the number of documents and val is the number of different documents a word occurs in.

We used the previously calculated worddict and implemented this formula. Here N is the total number of documents.
The next function tfidf() multiplies term frequency with idf of each term.
X is a pandas dataframe of tf idf of each document.

X transpose looks like this:



```
                    0         1         2         3         4
kingdom      0.000000  0.000000  0.000000  0.402359  0.000000
faster       0.000000  0.089413  0.000000  0.000000  0.000000
much         0.000000  0.089413  0.000000  0.000000  0.000000
space        0.000000  0.000000  0.146313  0.000000  0.000000
svd          0.146313  0.000000  0.000000  0.000000  0.000000
libraries    0.000000  0.089413  0.000000  0.000000  0.000000
animal       0.000000  0.000000  0.000000  0.402359  0.000000
echelon      0.000000  0.000000  0.146313  0.000000  0.000000
mango        0.000000  0.000000  0.000000  0.000000  0.536479
reducing     0.000000  0.000000  0.146313  0.000000  0.000000
king         0.000000  0.000000  0.000000  0.229073  0.305430
```

Figure 2. X transpose

Here columns are the document numbers and rows are the concepts. As mentioned above, the values are the product of term frequency with idf of each term.

## Singular Value Decomposition

We convert X into a 2D array so that we can perform SVD on it.

In order to perform SVD, first we find A^T * A and its eigenvalues and eigenvectors. This is done by using Jacobi (since Jacobi only works on symmetric matrices and A^T*A is always a symmetric matrix). Please see the steps of Jacobi method in the  to understand the steps we implemented. After getting the eigenvalues and vectors, we found the singular values and stored them in the list S. Furthermore, we found the U matrix by using the formula U_i = (A*V_i)/(S_i)

The function SvD() returns U,S, VT

## STEP-3: Finding best low-rank approximation of the TF-IDF matrix

The zero_padding function is used to return the Sigma matrix such that Sigma has len(U) rows and len(VT) columns. It stores only the first n singular values of the matrix.

The reconstruct() function simply multiplies the parameters given to it. Before multiplying them, it pads using the zero_padding() function.

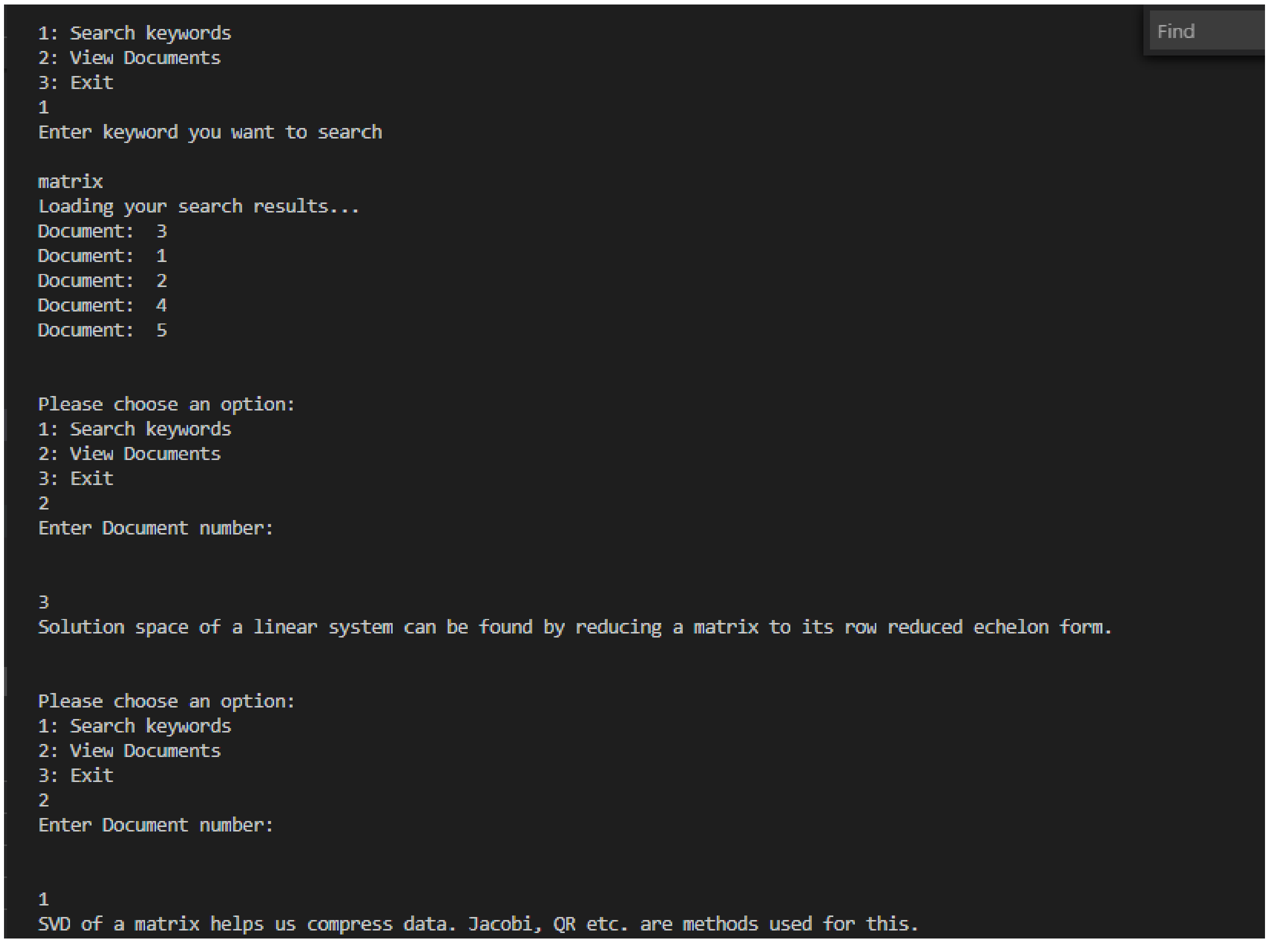
In the frobenius function, we use frobenius norm in our program to compare two matrices, A1 and A2 with respect to A1. We find the frobenius norm of A1 - A2 and the frobenius norm of A1. The value which we get after dividing the two matrices is used to determine how much A2 varies compared to A1. This function always returns a value between 0 and 1.

The find_k() function iterated through number of singular values -- len(S) and checks the frobenius norm between the original matrix passed as input into the SVD (l_2d) and the reconstructed matrix which has number of singular values equal to i (the iterator). It will return the value of i as soon as the frobenius function returns $f < 0.35$. (We set this threshold arbitrarily after doing a few tests on the input)

## STEP-4: OUTPUT:

The search function takes a string q as input. It removes all punctuations from q, converts it into a lower form and splits it into different words using the split() method. This makes q a list of strings. We store all the unique terms in ‘terms’. After that we traverse through q and see whether its elements are there in ‘term’. If none of its elements are there in term, we print “Could not find any documents”. We do this by using a list called ‘query’.We append 0 to query if any term from q is not present in term, else, we append 1 to it.

Next, we find the minimum number of singular values required to create a matrix which gives output of the frobenius function less than 0.35 (the threshold which was mentioned earlier).Using this value, we find the reconstructed matrix called reconstructed_A. Then, we make a list called ‘score’ by multiplying the query and reconstructed_A. The list ‘score’ contains the relevance scores of each document. Using this list, we print the relevance ranking.



```
1: Search keywords
2: View Documents
3: Exit
1
Enter keyword you want to search

matrix
Loading your search results...
Document:  3
Document:  1
Document:  2
Document:  4
Document:  5


Please choose an option:
1: Search keywords
2: View Documents
3: Exit
2
Enter Document number:


3
Solution space of a linear system can be found by reducing a matrix to its row reduced echelon form.


Please choose an option:
1: Search keywords
2: View Documents
3: Exit
2
Enter Document number:


1
SVD of a matrix helps us compress data. Jacobi, QR etc. are methods used for this.
```

## End results

The program which compares 5 documents takes about 1 minute time to compute U, Sigma and V transpose (dimensions of the input matrix are approximately 40*40). After these three are computed, the program proceeds to run smoothly.

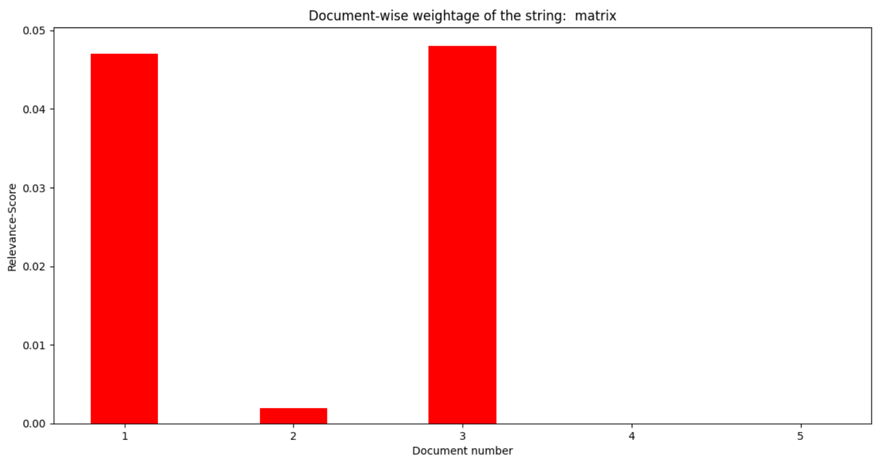
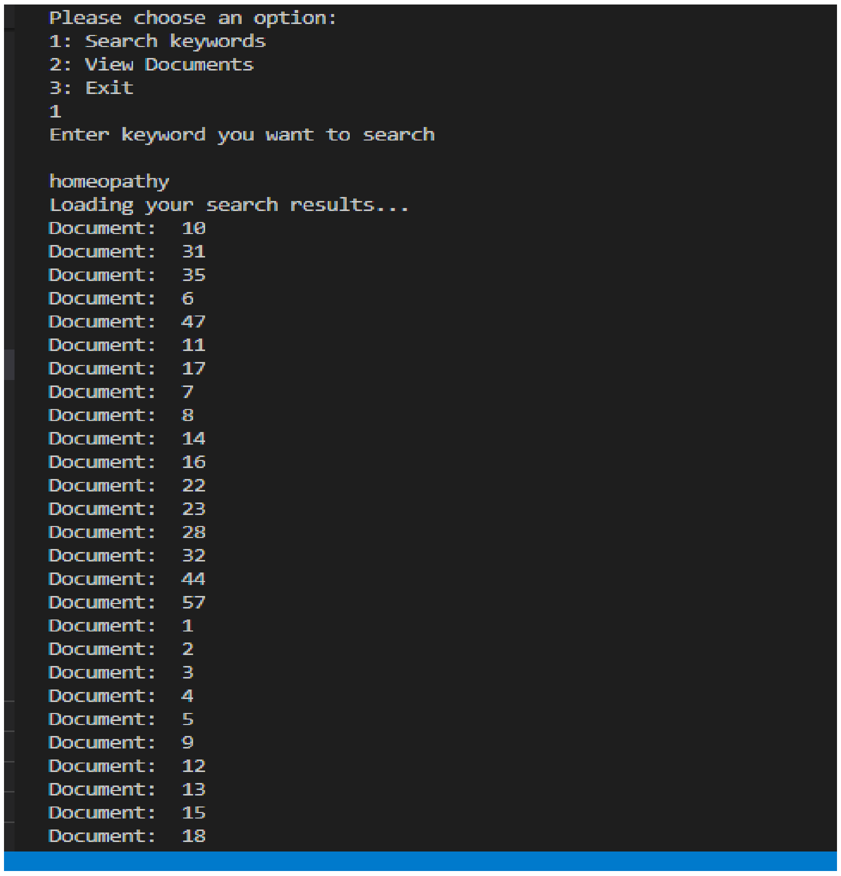


Figure 4. Graph of Relevance score Vs Document number (scratch)

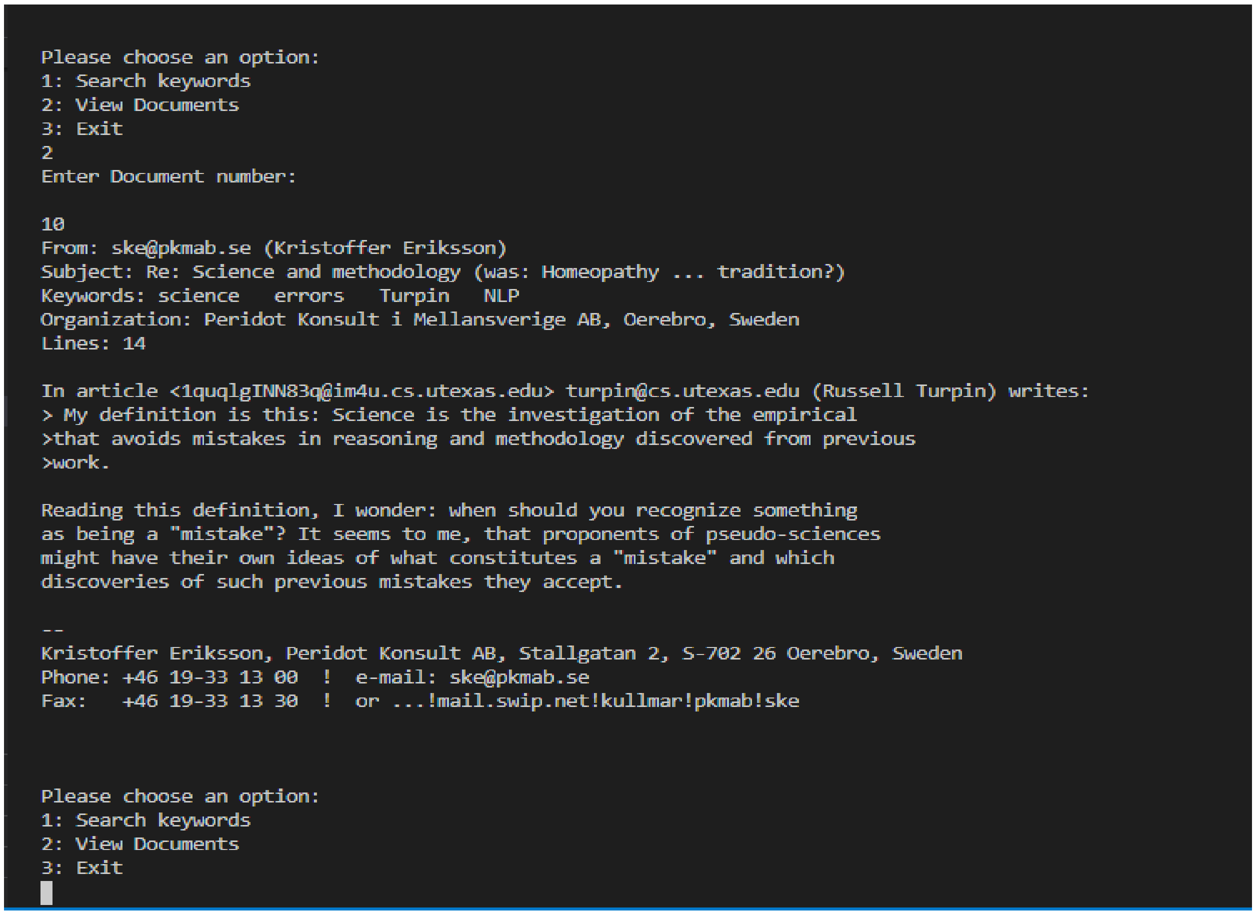


```
Please choose an option:
1: Search keywords
2: View Documents
3: Exit
1
Enter keyword you want to search

homeopathy
Loading your search results...
Document:  10
Document:  31
Document:  35
Document:  6
Document:  47
Document:  11
Document:  17
Document:  7
Document:  8
Document:  14
Document:  16
Document:  22
Document:  23
Document:  28
Document:  32
Document:  44
Document:  57
Document:  1
Document:  2
Document:  3
Document:  4
Document:  5
Document:  9
Document:  12
Document:  13
Document:  15
Document:  18
```

Figure 5.

Search result of the word "homeopathy"



```
Please choose an option:
1: Search keywords
2: View Documents
3: Exit
2
Enter Document number:

10
From: ske@pkmab.se (Kristoffer Eriksson)
Subject: Re: Science and methodology (was: Homeopathy ... tradition?)
Keywords: science   errors   Turpin   NLP
Organization: Peridot Konsult i Mellansverige AB, Oerebro, Sweden
Lines: 14

In article <1quqlgINN83q@im4u.cs.utexas.edu> turpin@cs.utexas.edu (Russell Turpin) writes:
> My definition is this: Science is the investigation of the empirical
>that avoids mistakes in reasoning and methodology discovered from previous
>work.

Reading this definition, I wonder: when should you recognize something
as being a "mistake"? It seems to me, that proponents of pseudo-sciences
might have their own ideas of what constitutes a "mistake" and which
discoveries of such previous mistakes they accept.

--
Kristoffer Eriksson, Peridot Konsult AB, Stallgatan 2, S-702 26 Oerebro, Sweden
Phone: +46 19-33 13 00  !  e-mail: ske@pkmab.se
Fax:   +46 19-33 13 30  !  or ...!mail.swip.net!kullmar!pkmab!ske



Please choose an option:
1: Search keywords
2: View Documents
3: Exit
```

Figure 6. Text in document 10 (with library)

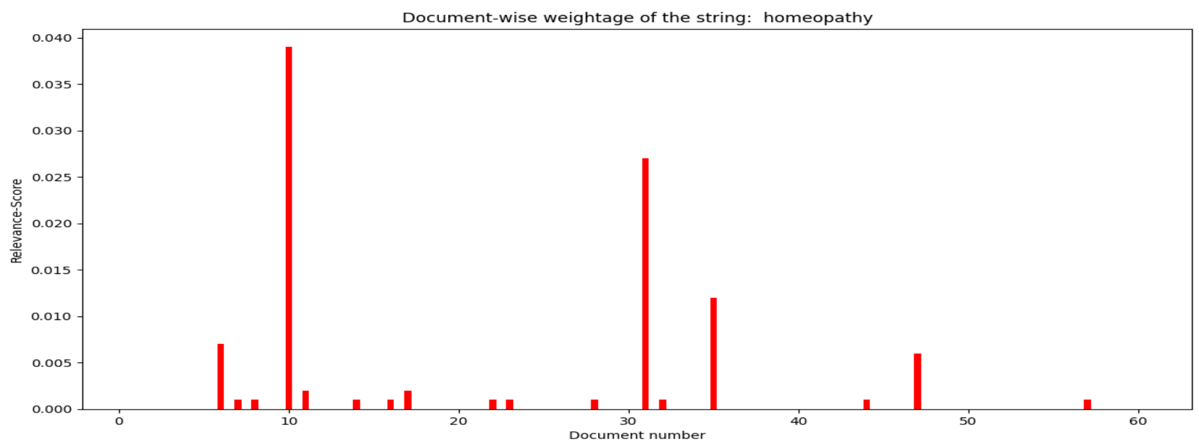


Figure 7. Graph of Relevance Score vs Document Number

## Inferences:

- We got an insight on how SVD helps us compress data.
- SVD would provide a lot more justification to LSI if we were working with more documents than the ones we are (5 documents for the program done from scratch and 60 for the ones using libraries).
- LSI is much more useful when the number of documents is much more than the number of words. We were able to use LSI on only 5 documents because of the inefficiency of the code. However, we saw how well data could be compressed by using SVD on a matrix containing unique words of 60+ documents!
- Libraries can make a program several times more efficient.

## References

Lzakharov, “lzakharov/lsi,” *GitHub*. [Online]. Available: https://github.com/lzakharov/lsi/tree/master/src [Accessed: 06-Nov-2020].

“Databricks Academy,” *YouTube*. [Online]. Available: https://www.youtube.com/channel/UCTrcvusxQ1Hy6qTE9uTxafQ/featured . [Accessed: 06-Nov-2020].

“1.3.3 The Frobenius norm,” *YouTube*, 13-Sep-2019. [Online]. Available: https://www.youtube.com/watch?v=yiSKsLcniGw . [Accessed: 06-Nov-2020].

“JACOBI'S METHOD TO FIND EIGEN VALUES FOR A SYMMETRIC MATRIX,” *YouTube*, 08-Apr-2020. [Online]. Available: https://youtu.be/6R1kVlIrtZE . [Accessed: 06-Nov-2020].

“Iterative methods for eigenvalue problem,” *NLA2013*, 2013. [Online]. Available: https://www.cmi.ac.in/~ksutar/NLA2013/iterativemethods.pdf . [Accessed: 2020].