University of Colorado Boulder

Final Project Report

APPM 3310 - Matrix Methods and Applications

Reverse Engineering PubMed Similar Articles Recommendations Using Latent Semantic Indexing

Authors:	Professors & Section Numbers:
Kai Drumm	Ian Grooms (Section: 001)
Joshua Park	Jim Curry (Section: 002)
Taryn Welch	Jim Curry (Section: 002)

April 28th, 2021

Abstract

Latent Semantic Indexing (LSI) is a language processing technique that helps distinguish the relationship between a set of documents and the words within those documents. In this report, we collected fifty-five article abstracts from PubMed and compared them to a different set of twenty-five article abstracts to calculate the similarity between each set of abstracts dealing with COVID-19. After completing this process, we were able to mimic the behavior of the PubMed "Similar Articles" feature, demonstrating that LSI is an effective method to group articles together in order to find similarity between them.

Attribution

Throughout this project, Kai was the main contributor for the Python code. The three of us equally contributed to the data collection as well as writing up the analysis of our numerical results.

Introduction

Latent Semantic Indexing is a natural language processing technique that analyzes the relationships between a set of documents and the words in those documents. Given a set of documents, we can construct an MxN term-document matrix A where each row represents a unique word found in our set of the documents and each column represents a document in our set. The entries of the matrix contain weights. These may be assigned in different ways, which we will discuss later in this paper, but the simplest way to assign weights is to use the frequency of a word in a particular document. LSI will take the term-document matrix and project it into a lower dimensional space using a truncated singular value decomposition. In this latent semantic space we can compare the similarities of queries and documents and observe a high similarity even if the query and document do not share like terms.

Two common problems in natural language processing are synonymy and polysemy. Synonymy refers to the idea that different terms can refer to the same concept and polysemy refers to the idea that certain terms that have multiple meanings. To deal with this problem, instead of using individual words for analysis, LSI uses statistical indices to analyze and derive similarities. To do this, LSI assumes that word choice has some basic, underlying arrangement, which can vary due to the disparity in word usage. LSI uses truncated singular value decomposition (SVD) to score each document on a set of component axes, which are meant to capture inherent linguistic meaning through assigning weights to each vocabulary word.

For this project, we will take fifty-five medical paper abstracts from PubMed dealing with COVID-19 and use them as our training set. Using these abstracts, we will construct a term-document matrix and perform LSI. We will then select twenty-five abstracts that were deemed “similar” by PubMed’s recommendation algorithm to our test set while keeping track of the new abstract’s “parent” paper that was in our training set. We will query the new abstracts and determine the document in our training set with the highest similarity score. We can evaluate the accuracy of LSI by comparing the document with the highest similarity score to the query and the “parent” paper.

In the following sections of this report, we will explain the mathematical calculations behind our project, present our results through graphs, and analyze our findings. In the “Mathematical Formulation” section, we will dive into the mathematical process and theory behind latent semantic indexing. In the “Examples and Numerical Results” section, we will look at the results gathered through our data processing in Python. To conclude our report, we will share our analysis and final conclusions in the “Discussion and Conclusions” section.

Mathematical Formulation

In this section, we will dive into the mathematical theory behind LSI, including singular value decomposition, low-rank approximation, query vectors, and term weighting scheme.

Linear Algebra setup for SVD:

Given an NxN square matrix S with N linearly independent eigenvectors, there exists a decomposition known as the *eigen decomposition* such that:

$$S = U\Lambda U^{-1} \tag{1}$$

where Λ is a diagonal matrix with the diagonal entries equal to the eigenvalues of S in decreasing order ($\lambda_{i,i} \geq \lambda_{i+1,i+1}$) and the columns of U are the eigenvectors of S. Taking the decomposition above one step further we can say for anNxN symmetric matrix $S$with N linearly independent eigenvectors, there exists a decomposition known as the *symmetric diagonal decomposition* or the *spectral decomposition* such that:

$$S = Q\Lambda Q^{T} \tag{2}$$

where Λ is still the diagonal matrix whose entries are eigenvalues of S in decreasing order and the columns of Q are orthonormal eigenvectors of S. However, most of the time our term-document matrix will not be a square matrix. We can extend the symmetric diagonal decomposition to rectangular matrices using *singular value decomposition*.

SVD and low-rank approximation:

The singular values of matrix A are defined as the square root of the eigenvalues of matrix A: $\sigma_i = \lambda_i^{1/2}$ for $1 \le i \le R$. The singular value decomposition of an MxNterm-document matrix A with rank R has the form:

$$A = U\Sigma V^T \tag{3}$$

where U is an MxR matrix with orthonormal columns, Σ is an RxR diagonal matrix with the *singular values* of A in decreasing order as its diagonal entries, and V is an RxN matrix with orthonormal rows. Once we have obtained a singular value decomposition for our term-document matrix we can find a low rank approximation of rank k (where $k << r$) by replacing Σ with Σ_k where $\Sigma_k$has the same first k singular values as Σ in its diagonal entries but the remaining $r - k$ diagonal entries are set to zero. Similarly, U_k and V are the original matrices with the same first k columns of the original matrices and 0 columns for the remaining columns. The matrix A has M documents and N terms, therefore, $U_k$represents the terms and their component values in the latent semantic space, while V_k represents the documents and their component values in the latent semantic space. Finally, we can compute A_k by

$$A_k = U_k\Sigma_k V^T_k \tag{4}$$

where $A_k$is the rank k approximation of A. Setting the remaining diagonal entries equal to zero guarantees the best rank k approximation because the singular values on the diagonal matrix Σ are in decreasing order. The smaller the singular value is, the less of an impact it has on matrix products. The theorem due to Eckart and Young tells us that that finding A_k using the method above will minimize the *Euclidean matrix norm* given by the formula below:

$$||A - A_k|| \tag{5}$$

Query vectors:

Once we have a low rank approximation of our term document matrix we can cast queries and compute the query-document similarity scores. We can map a query vector onto the new k-dimensional space by:

$$q_k = \Sigma_k^{-1} U_k^T q \qquad (6)$$

It is important to note $q$is not necessarily a document but rather a vector of words multiplied by their term weights. This representation is known as a "bag-of-words" representation of a word vector. This allows us to query anything from a single term to an entirely new document.

To determine the similarity between two documents, we can compute their cosine similarity using the documents' vector representation:

$$sim(d_1, d_2) = (d_1 \cdot d_2)/||d_1|| \, ||d_2|| \qquad (7)$$

If the word vectors for d_1, d_2 are normalized then the cosine similarity simplifies to the dot product between the two unit word vectors. The absolute value of a cosine similarity near 1 indicates high similarity while a cosine similarity near 0 indicates little to no similarity.

TF-IDF:

A common term weighting scheme when creating a term document matrix is the Term Frequency Inverse Document Frequency (TF-IDF) method. A term's weight will increase as its frequency increases (term frequency) . However, TF-IDF also penalizes terms that appear in many documents (inverse document frequency). This prevents the LSI from incorrectly emphasizing very common words in the set of documents. In our set of documents, the term "covid" would be penalized for showing up in many if not all the documents in our set. The TF-IDF weighting scheme follows the formula:

$$tfidf(t, d, D) = tf(t, d) \; * \; idf(t, D) \qquad (7)$$

where t is a term, d is a document from our set, and D is the set of all documents. The formulas for term frequency and inverse document frequency are shown below:

$$tf(t, d) = t_f / t_{tot} \qquad (8)$$

where t_f is the term frequency of some term t and t_{tot} is the total number of terms.

$$idf(d, D) = log(N_D / N_{doc}) \qquad (9)$$

where N_D is the total number of documents in our set and N_{doc} is the number of documents that the term t appears in (Note: N_{doc} is nonzero, otherwise, we would not have the term t in our term document matrix).

<u>Examples and Numerical Results</u>

We first collected a training data set made up of fifty-five scientific paper abstracts on PubMed related to the search term “COVID-19”. These were manually collected into text files consisting of the abstract, title, and DOI. For the sake of simplicity we used as a naming convention the first four significant words of the title and the first author’s surname.

Later, we collected a test set of twenty-five additional abstracts, stored in a similar format. These were selected from the PubMed “Similar Articles” section of articles in our training set, with the stipulation that these should not be duplicates of papers already in our training set. For these articles we recorded both the article’s own DOI and the DOI of the article from which it was suggested on PubMed.

For processing data and performing analysis we used Python with Pandas, Scipy, and Numpy and Google Colaboratory notebook. We used Scikit-Learn's CountVectorizer and TF-IDF methods to generate document vectors with either method from the text of the abstracts. CountVectorizer computes term weights by determining their term frequency in a given document while TF-IDF uses both term frequency and inverse document frequency to give more weight to words which appear in only a few documents. For both we used the default English stopword list to remove common articles, prepositions, and so forth which do not contain much inherent meaning. We then used Numpy’s singular value decomposition method to factor the term-document matrix and simple matrix operations to truncate the resulting matrices down to the desired number of components. Of the resulting matrices, we consider the left-side U matrix to be the “document by component” matrix, the middle S matrix to be the components, and the right-side V^T matrix to be the “component by term” matrix.

To query the database, we generated vectors in the same way from the test set of abstracts. Instead of performing SVD on this matrix, we transformed it into the existing k-space using equation (6). We then calculated the cosine similarity between each training set document vector and each query document vector. To do so we multiplied the U matrix by the matrix of query vectors, after first normalizing the component vectors of each.

We then computed the top n document(s) in our training set with the highest cosine similarity score(s). We scored our performance by counting the number of test documents which, when used as a query, returned the target article among the top n matches. We calculated scores for various settings including: (1) how many components we preserve from the singular value decomposition; (2) how many documents we include in our set of the top n matches; and (3) whether the document vectors are calculated using simple word counts or TF-IDF. We graph the resulting scores as they change with parameters in Figure 4.

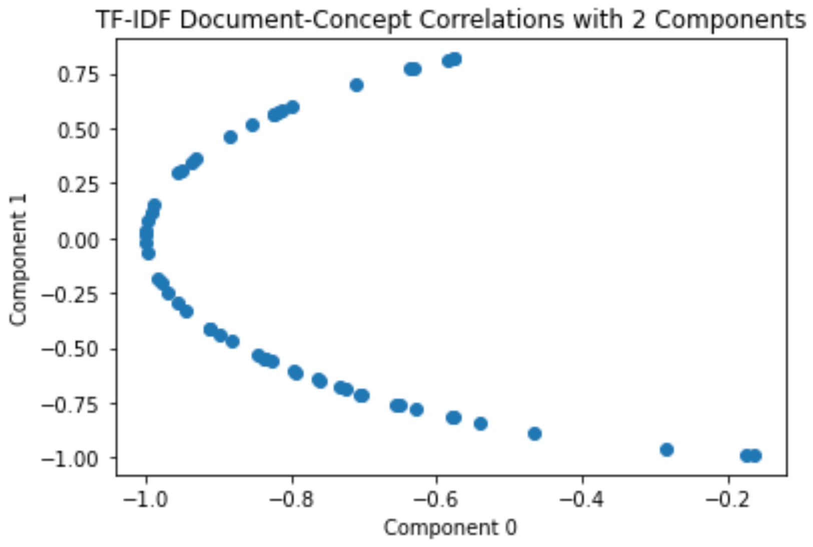
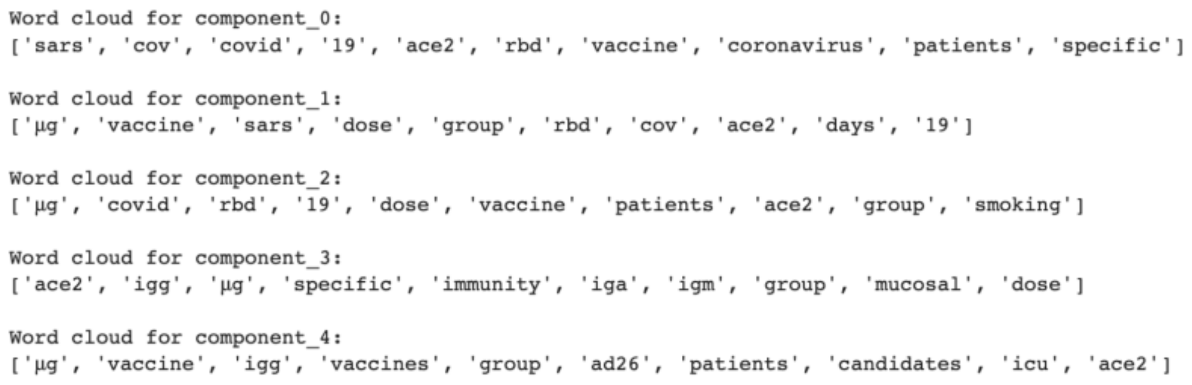


Figure 1: The document similarities found from the TF-IDF term-document matrix with two components (2-dimensional)

```
Word cloud for component_0:
['sars', 'cov', 'covid', '19', 'ace2', 'rbd', 'vaccine', 'coronavirus', 'patients', 'specific']

Word cloud for component_1:
['µg', 'vaccine', 'sars', 'dose', 'group', 'rbd', 'cov', 'ace2', 'days', '19']
```

Figure 1a: The word cloud associated with the two factors each contain the top ten terms. The terms are ordered from greatest similarity to least.

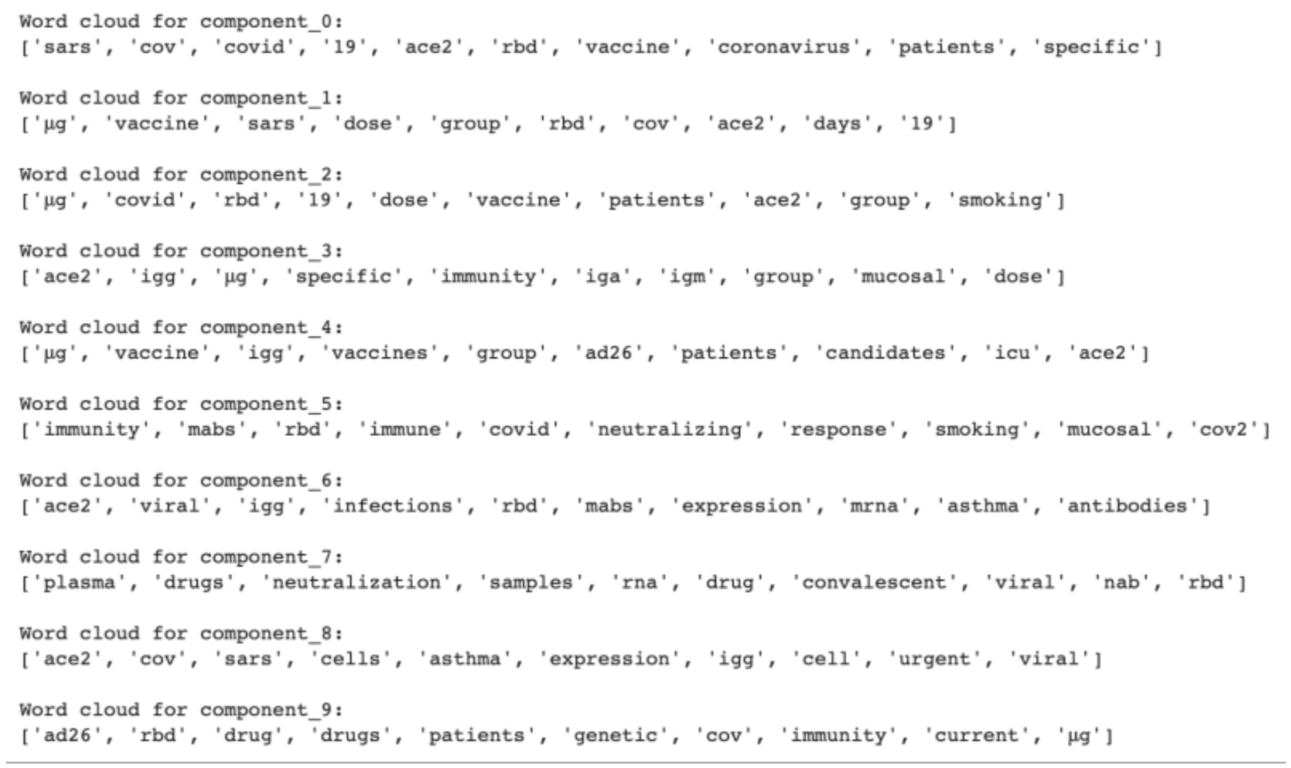
Figure 1 shows the document-component relationship for two components. Figure 1a shows the word clouds for the terms that are highly associated with the components in decreasing order of similarity. The interesting thing to note is that terms like “covid” and “coronavirus” are still appearing in the word clouds despite using TF-IDF to penalize terms that would show up in many documents. However, the significant overlap of terms in each component is to be expected with only two components because reducing the rank of the term-document matrix significantly has caused some loss of information. This is likely the reason why so many mutual terms are showing up in our word clouds.



```
Word cloud for component_0:
['sars', 'cov', 'covid', '19', 'ace2', 'rbd', 'vaccine', 'coronavirus', 'patients', 'specific']

Word cloud for component_1:
['µg', 'vaccine', 'sars', 'dose', 'group', 'rbd', 'cov', 'ace2', 'days', '19']

Word cloud for component_2:
['µg', 'covid', 'rbd', '19', 'dose', 'vaccine', 'patients', 'ace2', 'group', 'smoking']

Word cloud for component_3:
['ace2', 'igg', 'µg', 'specific', 'immunity', 'iga', 'igm', 'group', 'mucosal', 'dose']

Word cloud for component_4:
['µg', 'vaccine', 'igg', 'vaccines', 'group', 'ad26', 'patients', 'candidates', 'icu', 'ace2']
```

Figure 2: The word cloud for LSI from a TF-IDF term-document matrix with five components. The top ten terms associated with each component are shown in order from greatest to least similarity.

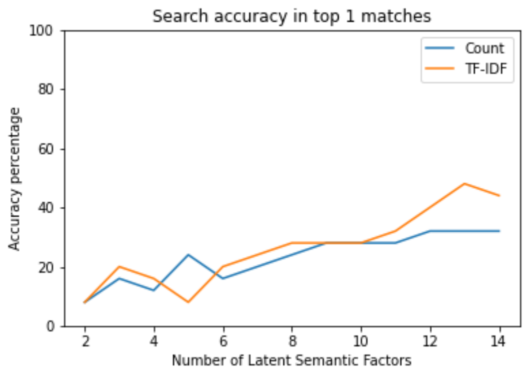
Although the word cloud above for five components shows a greater variety of terms than the word clouds for two components, we still observe significant overlap of terms in each component. There are unique terms for some components such as “smoking” in component 2. This could indicate documents that are related to the effects of covid-19 on smokers would have a high similarity score on component 2.

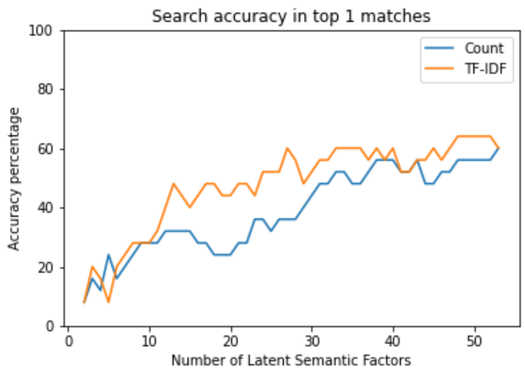


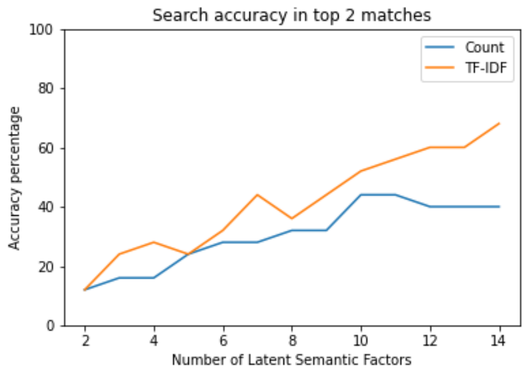
```
Word cloud for component_0:
['sars', 'cov', 'covid', '19', 'ace2', 'rbd', 'vaccine', 'coronavirus', 'patients', 'specific']

Word cloud for component_1:
['µg', 'vaccine', 'sars', 'dose', 'group', 'rbd', 'cov', 'ace2', 'days', '19']

Word cloud for component_2:
['µg', 'covid', 'rbd', '19', 'dose', 'vaccine', 'patients', 'ace2', 'group', 'smoking']

Word cloud for component_3:
['ace2', 'igg', 'µg', 'specific', 'immunity', 'iga', 'igm', 'group', 'mucosal', 'dose']

Word cloud for component_4:
['µg', 'vaccine', 'igg', 'vaccines', 'group', 'ad26', 'patients', 'candidates', 'icu', 'ace2']

Word cloud for component_5:
['immunity', 'mabs', 'rbd', 'immune', 'covid', 'neutralizing', 'response', 'smoking', 'mucosal', 'cov2']

Word cloud for component_6:
['ace2', 'viral', 'igg', 'infections', 'rbd', 'mabs', 'expression', 'mrna', 'asthma', 'antibodies']

Word cloud for component_7:
['plasma', 'drugs', 'neutralization', 'samples', 'rna', 'drug', 'convalescent', 'viral', 'nab', 'rbd']

Word cloud for component_8:
['ace2', 'cov', 'sars', 'cells', 'asthma', 'expression', 'igg', 'cell', 'urgent', 'viral']

Word cloud for component_9:
['ad26', 'rbd', 'drug', 'drugs', 'patients', 'genetic', 'cov', 'immunity', 'current', 'µg']
```

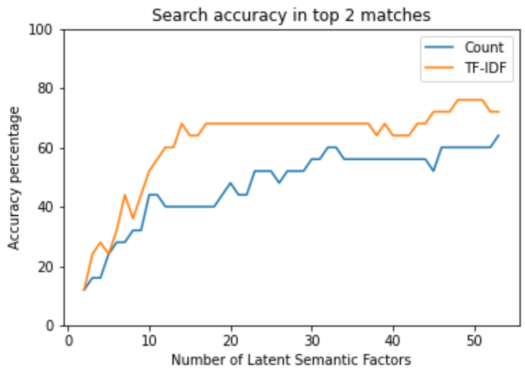
Figure 3: The word cloud for LSI from a TF-IDF term-document matrix with ten components. The top ten terms associated with each component are shown in order from greatest to least similarity.

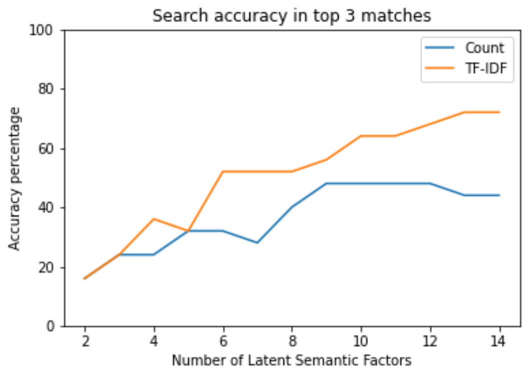
The word clouds for ten components shows more unique words in a majority of the components than the word clouds of two and five components. For example, component 7 is the first component we've seen with a unique term, "plasma", that has the highest similarity to a component. The terms like "covid" and "coronavirus" still appear in multiple components, however, there are now components that do not contain that term. This is likely due to TF-IDF adjusting the weight of that term to prevent overemphasis.

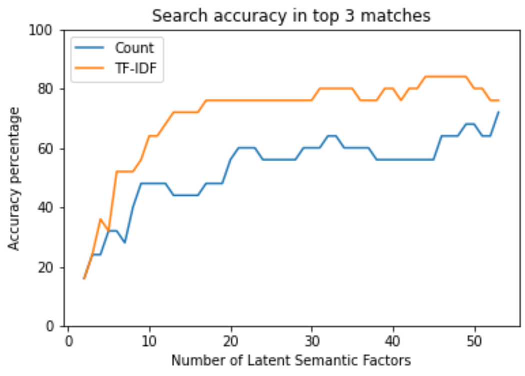

Search accuracy in top 1 matches
Count
TF-IDF
Accuracy percentage
Number of Latent Semantic Factors


Search accuracy in top 1 matches
Count
TF-IDF
Accuracy percentage
Number of Latent Semantic Factors


Search accuracy in top 2 matches
Count
TF-IDF
Accuracy percentage
Number of Latent Semantic Factors


Search accuracy in top 2 matches
Count
TF-IDF
Accuracy percentage
Number of Latent Semantic Factors


Search accuracy in top 3 matches
Count
TF-IDF
Accuracy percentage
Number of Latent Semantic Factors


Search accuracy in top 3 matches
Count
TF-IDF
Accuracy percentage
Number of Latent Semantic Factors

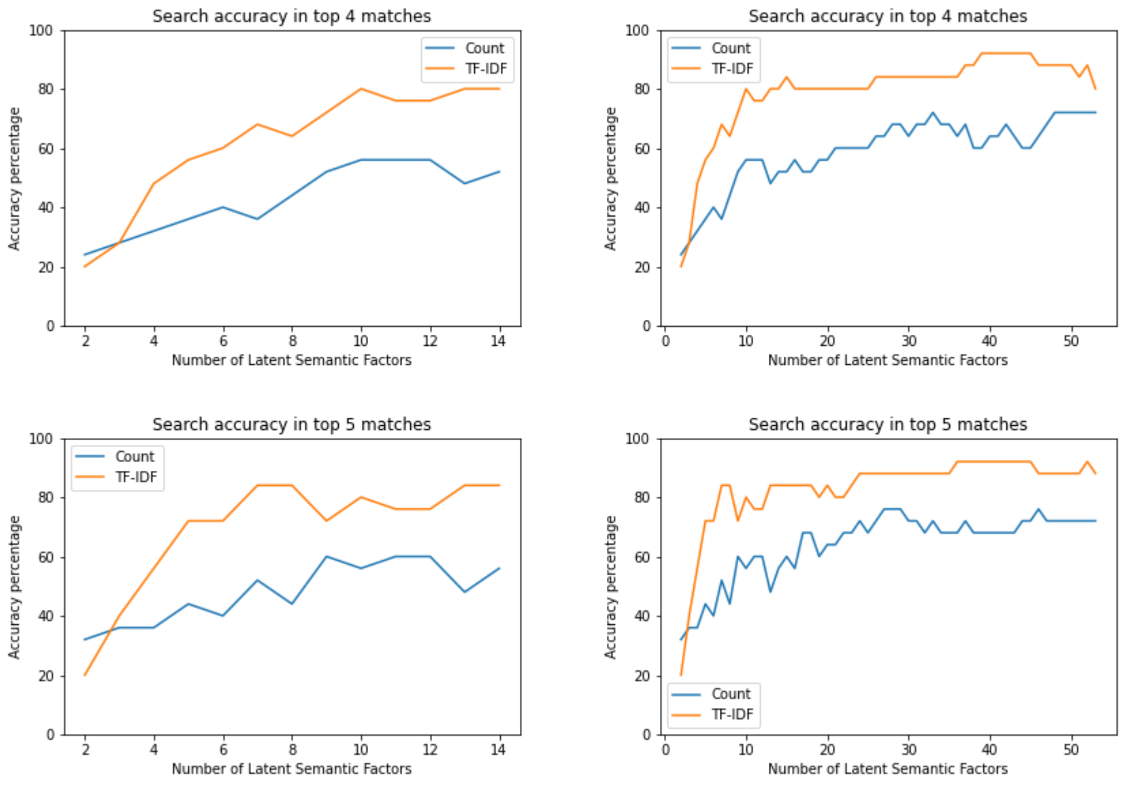


Figure 4: The graphs show the accuracy of both the TF-IDF and CountVectorizer models while varying the number of components. The different graphs score the accuracy based on if the true parent document is in the top 1-5 matches.

Figure 4 shows the TF-IDF model outperforms the CountVectorizer model overall which is to be expected. We initially calculated the search accuracy by returning the top similar document from our training set and comparing it with the true parent document. We noticed the accuracy was very low and this is likely due to the fact that many articles in our data set are similar and in fact appear on each other's Similar Articles lists. Therefore, if we query a document similar to one in our training set, it is likely similar to multiple documents in our training set; allowing for this, our algorithm works well we we consider the top 4-5 matches. The TF-IDF model appears to plateau after about 10 components while the CountVectorizer model continues to increase as the number of components increases. The search accuracy in the top 5 matches shows the TF-IDF model at above 80% after roughly 10 components indicating the LSI method using TF-IDF is an effective way to compare document similarities.

Discussion and Conclusions

We set out to test if the Latent Semantic Indexing technique can be used to replicate the "similar articles" recommendation system already present in the PubMed browsing interface, and we succeeded. We proved to ourselves that the LSI technique can be useful in linking between related written documents. This could be used in any textual recommendation-based browsing environment, such as recommendations of books to read based on a favorites list, or topics on a discussion board similar to the one being viewed. It is likely that the technique might be relevant in audiovisual based similarities as well.

A few directions to investigate beyond this would be to explore the performance of LSI on large datasets, to apply it to other specific use cases, to identify rules of thumb for optimal number of components based on database size, to use n-gram text representations to improve on the bag-of-words model, and to attempt to group and label document clusters rather than simply find the best matches to a query. These are active areas of research which reveal that LSI has continued potential in the field of natural language processing.

Our dataset was rather small, given the limitations of manually collecting abstracts as text files instead of writing a web crawler to scrape them or making use of a preexisting plaintext article repository. Automatic data collection or a larger database should absolutely be used to expand this work and generate more interesting analysis. With a larger dataset, we will need to pay more attention to algorithmic efficiency and data storage techniques. Luckly, highly efficient matrix multiplication algorithms are available which should make the SVD calculation scalable over larger datasets.

While measuring performance based on our ability to reverse engineer the PubMed similar articles section is an artificial metric, it helped us to validate that our algorithm was working as expected in the absence of a more pressing research question or set of human-labelled similarity scores. We would be interested to expand the work by applying it to a novel use case. For example, such a method might be useful within an electronic version of our textbook to suggest related sections when one chapter references the concepts from another.

With larger datasets in play, we could perform more analysis on the optimal parameters for the number of components to keep and the best stopword set to use for a specific application. We were able to vary the number of components in our analysis as shown in Figure 4. In most scenarios we saw the success rate plateau as the number of components increased, indicating a point of diminishing returns. We would be interested to see if we can predict this inflection point. In general, choosing a number of components that is too low will lose valuable information but keeping too many components will bring in noise and increase computational requirements. Exploring further on how to choose an appropriate number of components will help us understand LSI and create better models.

In creating our vector representations of the documents, we used either a simple count of each word in the vocabulary, or an adjusted weight for counting those words based on their tf-idf ratio. These are both examples of a ‘bag-of-words’ representation of text; that is, they do not preserve any information about word order. To create a more sophisticated LSI analysis, we can further process the text to include information about n-grams - that is, sequences of 2, 3, or 4 words linked together - in the vectors. This would preserve meaning based on multi-term phrases.

Finally, one area of analysis that we considered but have not included is to group the materials into topic clusters based on the cosine similarity between each document vector and every other one. We would be able to calculate these as the Gram matrix of $U_{kn}U_{kn}^{T}$, where U_{kn} is the normed version of the truncated document-by-component matrix. Currently we can output document clusters that lie directly along the component axes - for example, we can return the top 5 documents most strongly correlated with Component 1. However, strategies to group the entire collection would require some additional analysis, using algorithms from the topic area of correlation clustering.

References

Cook, Joshua. *A Second LSA (3/5)*. *YouTube*, Databricks Academy, 22 Apr. 2019, www.youtube.com/watch?v=NWb_4O3ssbA.

Cook, Joshua. *A Trivial Implementation of LSA Using Scikit Learn (2/5)*. *YouTube*, Databricks Academy, 16 May 2019, www.youtube.com/watch?v=Fy0bF7u6W20.

Cook, Joshua. *Improving the LSA with a TFIDF (4/5)*. *YouTube*, Databricks Academy, 22 Apr. 2019, www.youtube.com/watch?v=YX4xRIQ84Z0.

Cook, Joshua. *Introduction to Latent Semantic Analysis (1/5)*. *YouTube*, Databricks Academy, 22 Apr. 2019, www.youtube.com/watch?v=hB51kkus-Rc.

Cook, Joshua. *Latent Semantic Analysis with Apache Spark (5/5)*. *YouTube*, Databricks Academy, 16 May 2019, www.youtube.com/watch?v=OXJdfpEFmF8.

"Correlation Clustering." *Wikipedia*, Wikimedia Foundation, 13 Dec. 2020, en.wikipedia.org/wiki/Correlation_clustering.

Deerwester, Scott, et al. "Indexing by Latent Semantic Analysis." *Journal of the American Society For Information Science*, 1990, pp. 391–407., lsa.colorado.edu/papers/JASIS.lsi.90.pdf.

Homayouni, Ramin, et al. "Gene Clustering by Latent Semantic Indexing of MEDLINE Abstracts." *Bioinformatics*, vol. 21, no. 1, 2004, pp. 104–115., doi:10.1093/bioinformatics/bth464.

Manning, Christopher D., et al. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

Stanton, William G. "Latent Semantic Analysis." www.datascienceassn.org/sites/default/files/users/user1/lsa_presentation_final.pdf.

Articles Used In LSI Calculations

- "A clade of SARS-CoV-2 viruses associated with lower viral loads in patient upper airways" by Ramon Lorenzo-Redondo and others (DOI: 10.1016/j.ebiom.2020.103112)
- "A Longitudinal Study of Immune Cells in Severe COVID-19 Patients" by Didier Payen and others (DOI: 10.3389/fimmu.2020.580250)
- "A neutralizing human antibody binds to the N-terminal domain of the Spike protein of SARS-CoV-2" by Xiangyang Chi and others (DOI: 10.1126/science.abc6952)
- "A systematic review of asymptomatic infections with COVID-19" by Zhiru Gao and others (DOI: 10.1016/j.jmii.2020.05.001)
- "ACE2 receptor polymorphism: Susceptibility to SARS-CoV-2, hypertension, multi-organ failure, and COVID-19 disease outcome" by Christian A Devaux and others (DOI: 10.1016/j.jmii.2020.04.015)
- "Angiotensin-converting enzyme 2 (ACE2), SARS-CoV-2 and the pathophysiology of coronavirus disease 2019 (COVID-19)" by Arno R Bourgonje and others (DOI: 10.1002/path.5471)
- "Cell entry mechanisms of SARS-CoV-2" by Jian Shang and others (DOI: 10.1073/pnas.2003138117)

- “Clinical Characteristics and Outcome of Hospitalized COVID-19 Patients in a MERS-CoV Endemic Area” by Mazin Barry and others (DOI: 10.2991/jegh.k.200806.002)
- “Comparative analyses of SARS-CoV-2 binding (IgG, IgM, IgA) and neutralizing antibodies from human serum samples” by Livia Mazzini and others (DOI: 10.1016/j.jim.2020.112937)
- “Comparison of the immunogenicity & protective efficacy of various SARS-CoV-2 vaccine candidates in non-human primates” by Labanya Mukhopadhyay and others (DOI: 10.4103/ijmr.IJMR_4431_20)
- “COVID-19 and cancer: From basic mechanisms to vaccine development using nanotechnology” by Hyun Jee Han and others (DOI: 10.1016/j.intimp.2020.107247)
- “COVID-19 and immunomodulation treatment for women with reproductive failures” by Lenka Sedláčková (PMID: 33823602)
- "COVID-19 and Smoking: What Evidence Needs Our Attention?" by Jianghua Xie and others (DOI: 10.3389/fphys.2021.603850)
- “COVID-19 Susceptibility in chronic obstructive pulmonary disease” by Jordi Olloquequi (DOI: 10.1111/eci.13382)
- “COVID-19 vaccine BNT162b1 elicits human antibody and T(H)1 T cell responses” by Ugur Sahin and others (DOI: 10.1038/s41586-020-2814-7)
- “Decline of Humoral Responses against SARS-CoV-2 Spike in Convalescent Individuals” by Guillaume Beaudoin-Bussières and others (DOI: 10.1128/mBio.02590-20)
- “Diabetes and COVID-19” by Zohair Jamil Gazzaz (DOI: 10.1515/biol-2021-0034)

- “Different Innate and Adaptive Immune Responses to SARS-CoV-2 Infection of Asymptomatic, Mild, and Severe Cases” by Rita Carsetti and others (DOI: 10.3389/fimmu.2020.610300)
- “DNA vaccines against COVID-19: Perspectives and challenges” by Marcelle Moura Silveira, Gustavo Marçal Schmidt Garcia Moreira, and Marcelo Mendonça (DOI: 10.1016/j.lfs.2020.118919)
- “Do patients with rheumatoid arthritis show a different course of COVID-19 compared to patients with spondyloarthritis?“ by Rebecca Hasseli and others (PMID: 33822706)
- “Eating behaviors and weight outcomes in bariatric surgery patients amidst COVID-19” by Eva Conceição and others (DOI: 10.1016/j.soard.2021.02.025)
- “Evaluating SARS-CoV-2 spike and nucleocapsid proteins as targets for antibody detection in severe and mild COVID-19 cases using a Luminex bead-based assay” by Joachim Mariën and others (DOI: 10.1016/j.jviromet.2020.114025)
- “Expression of the SARS-CoV-2 cell receptor gene ACE2 in a wide variety of human tissues” by Meng-Yuan Li and others (DOI: 10.1186/s40249-020-00662-x)
- “Genomics in Patient Care and Workforce Decisions in High-Level Isolation Units: A Survey of Healthcare Workers” by Jennifer E Gerber and others (DOI: 10.1089/hs.2020.0182)
- “New insights into genetic susceptibility of COVID-19: an ACE2 and TMPRSS2 polymorphism analysis” by Yuan Hou and others (DOI: 10.1186/s12916-020-01673-z)
- “Immunity to SARS-CoV-2: Lessons Learned” by Jaime Fergie and Amit Srivastava (DOI: 10.3389/fimmu.2021.654165)

- “Human genetic factors associated with susceptibility to SARS-CoV-2 infection and COVID-19 disease severity” by Cleo Anastassopoulou and others (DOI: 10.1186/s40246-020-00290-4)
- “Immunogenicity and safety of a recombinant adenovirus type-5-vectored COVID-19 vaccine in healthy adults aged 18 years or older: a randomised, double-blind, placebo-controlled, phase 2 trial” by Feng-Cai Zhu and others (DOI: 10.1016/S0140-6736(20)31605-6)
- “Immunological Aspects Related to Viral Infections in Severe Asthma and the Role of Omalizumab” by Francesco Menzella and others (DOI: 10.3390/biomedicines9040348)
- “In silico inquest reveals the efficacy of Cannabis in the treatment of post-Covid-19 related neurodegeneration” by Indrani Sarkar and others (DOI: 10.1080/07391102.2021.1905556)
- “Insights into neutralizing antibody responses in individuals exposed to SARS-CoV-2 in Chile” by Carolina Beltrán-Pavez and others (DOI: 10.1126/sciadv.abe6855)
- “iPSC screening for drug repurposing identifies anti-RNA virus agents modulating host cell susceptibility” by Keiko Imamura and others (DOI: 10.1002/2211-5463.13153)
- “Kinetics of SARS-CoV-2 specific IgM and IgG responses in COVID-19 patients” by Baoqing Sun and others (DOI: 10.1080/22221751.2020.1762515)
- “Mechanisms of SARS-CoV-2 Transmission and Pathogenesis” by Andrew G Harrison, Tao Lin, and Penghua Wang (DOI: 10.1016/j.it.2020.10.004)
- “Methodological evaluation of bias in observational COVID-19 studies on drug effectiveness” by Oksana Martinuka, Maja von Cube, and Martin Wolkewitz (DOI: 10.1016/j.cmi.2021.03.003)

- “Mini Review Immunological Consequences of Immunization With COVID-19 mRNA Vaccines: Preliminary Results” by Andrea Lombardi and others (DOI: 10.3389/fimmu.2021.657711)
- “Molecular characterization, pathogen-host interaction pathway and in silico approaches for vaccine design against COVID-19” by Nidhi Singh and others (DOI: 10.1016/j.jchemneu.2020.101874)
- “Molecular Mechanism of Evolution and Human Infection with SARS-CoV-2” by Jiahua He and others (DOI: 10.3390/v12040428)
- “Mucosal Immunity in COVID-19: A Neglected but Critical Aspect of SARS-CoV-2 Infection” by Michael W Russell and others (DOI: 10.3389/fimmu.2020.611337)
- “Overview of Immune Response During SARS-CoV-2 Infection: Lessons From the Past” by Vibhuti Kumar Shah and others (DOI: 10.3389/fimmu.2020.01949)
- “Potently neutralizing and protective human antibodies against SARS-CoV-2” by Seth J Zost and others (DOI: 10.1038/s41586-020-2548-6)
- “Receptor-binding domain-specific human neutralizing monoclonal antibodies against SARS-CoV and SARS-CoV-2” by Fei Yu and others (DOI: 10.1038/s41392-020-00318-0)
- “Safety and immunogenicity of an inactivated SARS-CoV-2 vaccine, BBIBP-CorV: a randomised, double-blind, placebo-controlled, phase 1/2 trial” by Shengli Xia and others (DOI: 10.1016/S1473-3099(20)30831-8)
- “Safety and immunogenicity of a recombinant tandem-repeat dimeric RBD-based protein subunit vaccine (ZF2001) against COVID-19 in adults: two randomised, double-blind,

placebo-controlled, phase 1 and 2 trials” by Shilong Yang and others (DOI: 10.1016/S1473-3099(21)00127-4)

- “SARS-CoV-2 Neutralizing Antibodies: A Network Meta-Analysis across Vaccines” by Paola Rogliani and others (DOI: 10.3390/vaccines9030227)
- “Severe vitamin D deficiency is not related to SARS-CoV-2 infection but may increase mortality risk in hospitalized adults: a retrospective case-control study in an Arab Gulf country” by Abdullah M Alguwaihes and others (DOI: 10.1007/s40520-021-01831-0)
- “Single-shot Ad26 vaccine protects against SARS-CoV-2 in rhesus macaques” by Noe B Mercado and others (DOI: 10.1038/s41586-020-2607-z)
- “S Protein-Reactive IgG and Memory B Cell Production after Human SARS-CoV-2 Infection Includes Broad Reactivity to the S2 Subunit” by Phuong Nguyen-Contant and others (DOI: 10.1128/mBio.01991-20)
- “Structural basis of receptor recognition by SARS-CoV-2” by Jian Shang and others (DOI: 10.1038/s41586-020-2179-y)
- “Structural basis for the neutralization of SARS-CoV-2 by an antibody from a convalescent patient” by Daming Zhou and others (DOI: 10.1038/s41594-020-0480-y)
- “Systemic and mucosal antibody responses specific to SARS-CoV-2 during mild versus severe COVID-19” by Carlo Cervia and others (DOI: 10.1016/j.jaci.2020.10.040)
- “Targeted design of drug binding sites in the main protease of SARS-CoV-2 reveals potential signatures of adaptation” by Aditya K Padhi and Timir Tripathi (DOI: 10.1016/j.bbrc.2021.03.118)
- “Targeting SARS-CoV2 Spike Protein Receptor Binding Domain by Therapeutic Antibodies” by Arif Hussain and others (DOI: 10.1016/j.biopha.2020.110559)

- “The cytokine storm and COVID-19” by Biying Hu, Shaoying Huang, and Lianghong Yin (DOI: 10.1002/jmv.26232)
- “The immunology of SARS-CoV-2 infections and vaccines” by Lilit Grigoryan and Bali Pulendran (DOI: 10.1016/j.smim.2020.101422)
- “The novel coronavirus Disease-2019 (COVID-19): Mechanism of action, detection and recent therapeutic strategies” by Elahe Seyed Hosseini and others (DOI: 10.1016/j.virol.2020.08.011)
- “What are the roles of antibodies versus a durable, high quality T-cell response in protective immunity against SARS-CoV-2?” by Marc Hellerstein (DOI: 10.1016/j.jvacx.2020.100076)

```
from google.colab import drive

import os
import re
from collections import Counter

import pandas as pd
import numpy as np

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import Normalizer

from scipy.sparse import csc_matrix
from scipy.sparse.linalg import svds, eigs
from scipy.linalg import lu

drive.mount("/content/gdrive")
```

```
    Mounted at /content/gdrive
```

```
# Example from Stanford online textbook (Manning)
# nlp.stanford.edu/IR-book/html/htmledition/latent-semantic-indexing-1.html
C = [[1,0,1,0,0,0],[0,1,0,0,0,0],[1,1,0,0,0,0],[1,0,0,1,1,0],[0,0,0,1,0,1]]

# Round numbers when printing
np.set_printoptions(precision=2)
np.set_printoptions(suppress=True)

# Factor the original matrix
u, s, vh = np.linalg.svd(C)
print("\nMatrix U:\n",u)
print("\nSingular values:\n",s)
print("\nMatrix V^T:\n",vh)

# Store sigma as diagonal matrix
sigma = np.diag(s)

# Store vT as v
# Delete column associated with 0 singular value
v = np.transpose(vh[:-1, :])

# Reconstitute C2
C2 = u @ sigma @ np.transpose(v)
print("\nOriginal matrix recreated from factors\n", C2)

# Zeroing out sigma
sigma[2:] = 0
print("\nSigma zeroed to 2 components:\n", sigma)
```

```
C3 = sigma @ np.transpose(v)
print("\nC2 is U times truncated sigma\n", C3)

# # Truncate matrices
uprime = u[:, :2]
sprime = sigma[:2, :2]
vprime = v[:, :2]
print("\n U truncated:\n", uprime)
print("\nSigma truncated:\n", sprime)
print("\nVT truncated:\n", np.transpose(vprime))

# print(uprime.shape)
# print(sprime.shape)
# print(vprime.shape)
```

```
    Matrix U:
     [[ 0.44 -0.3  -0.57  0.58 -0.25]
     [ 0.13 -0.33  0.59  0.   -0.73]
     [ 0.48 -0.51  0.37  0.    0.61]
     [ 0.7   0.35 -0.15 -0.58 -0.16]
     [ 0.26  0.65  0.41  0.58  0.09]]

    Singular values:
     [2.16 1.59 1.28 1.   0.39]

    Matrix V^T:
     [[ 0.75  0.28  0.2   0.45  0.33  0.12]
     [-0.29 -0.53 -0.19  0.63  0.22  0.41]
     [-0.28  0.75 -0.45  0.2  -0.12  0.33]
     [-0.    0.    0.58  0.   -0.58  0.58]
     [ 0.53 -0.29 -0.63 -0.19 -0.41  0.22]
     [-0.   -0.    0.   -0.58  0.58  0.58]]

    Original matrix recreated from factors
     [[ 1.  0.  1. -0.  0. -0.]
     [ 0.  1. -0. -0.  0. -0.]
     [ 1.  1. -0.  0. -0.  0.]
     [ 1.  0. -0.  1.  1.  0.]
     [-0. -0. -0.  1.  0.  1.]]

    Sigma zeroed to 2 components:
     [[2.16 0.   0.   0.   0.  ]
     [0.   1.59 0.   0.   0.  ]
     [0.   0.   0.   0.   0.  ]
     [0.   0.   0.   0.   0.  ]
     [0.   0.   0.   0.   0.  ]]

    C2 is U times truncated sigma
     [[ 1.62  0.6   0.44  0.97  0.7   0.26]
     [-0.46 -0.84 -0.3   1.    0.35  0.65]
     [ 0.    0.    0.    0.    0.    0.  ]
     [ 0.    0.    0.    0.    0.    0.  ]
     [ 0.    0.    0.    0.    0.    0.  ]]
```

```
      U truncated:
      [[ 0.44 -0.3 ]
      [ 0.13 -0.33]
      [ 0.48 -0.51]
      [ 0.7   0.35]
      [ 0.26  0.65]]

     Sigma truncated:
      [[2.16 0.  ]
      [0.   1.59]]

     VT truncated:
      [[ 0.75  0.28  0.2   0.45  0.33  0.12]
      [-0.29 -0.53 -0.19  0.63  0.22  0.41]]
```

```
# Deerwester example
C = [[1,0,0,1,0,0,0,0,0],[1,0,1,0,0,0,0,0,0],[1,1,0,0,0,0,0,0,0],
     [0,1,1,0,1,0,0,0,0],[0,1,1,2,0,0,0,0,0],[0,1,0,0,1,0,0,0,0],
     [0,1,0,0,1,0,0,0,0],[0,0,1,1,0,0,0,0,0],[0,1,0,0,0,0,0,0,1],
     [0,0,0,0,0,1,1,1,0],[0,0,0,0,0,0,1,1,1],[0,0,0,0,0,0,0,1,1]]

# Print with rounding
np.set_printoptions(precision=2)
np.set_printoptions(suppress=True)

# Factor the original matrix
# Matrix naming convention matches example paper
# Full_matrices = False returns the factorization we are familiar with
T0, S0, D0 = np.linalg.svd(C, full_matrices = False)

print("\nLeft side factor, T:\n", T0)
print("\nSingular values:\n", S0)
print("\nRight side factor, D:\n", D0)
C_valid = T0 @ np.diag(S0) @ D0
print("\nOriginal matrix reconstructed from factors: \n", C_valid)

# Truncate matrices
T = T0[:, :2]
S = np.diag(S0[:2])
D = D0[:2, :]
print("\nLeft side truncated:\n", T)
print("\nSigma truncated:\n", S)
print("\nRight side truncated:\n", D)
X = T@S@D
print("\nThe closest rank-2 approximation to C is:\n", X)

# What about the query vector?
query_vector = [1,2,3,4,0,0,0,0,0,0,0,0]
print("\nAn example query vector, original:\n", query_vector)
sigma_k_inverse = np.linalg.inv(S)
U_k_transposed = np.transpose(T)
qk = sigma_k_inverse @ U_k_transposed @ query_vector
qk = sigma_k_inverse @ U_k_transposed @ query_vector
print("\nAn example query vector, in k-space:\n", qk)
```

```
      [ 0.11 -0.5   0.21  0.57 -0.51  0.1   0.19  0.25  0.08]
      [-0.95 -0.03  0.04  0.27  0.15  0.02  0.02  0.01 -0.02]
      [ 0.05 -0.21  0.38 -0.21  0.33  0.39  0.35  0.15 -0.6 ]
      [-0.08 -0.26  0.72 -0.37  0.03 -0.3  -0.21  0.    0.36]
      [-0.18  0.43  0.24 -0.26 -0.67  0.34  0.15 -0.25 -0.04]
      [ 0.01 -0.05 -0.01  0.02  0.06 -0.45  0.76 -0.45  0.07]
      [ 0.06 -0.24 -0.02  0.08  0.26  0.62 -0.02 -0.52  0.45]]

     Original matrix reconstructed from factors:
      [[ 1.  0.  0.  1. -0.  0.  0. -0.  0.]
      [ 1.  0.  1.  0.  0. -0.  0.  0. -0.]
      [ 1.  1.  0. -0.  0.  0. -0. -0.  0.]
      [-0.  1.  1.  0.  1.  0. -0. -0. -0.]
      [-0.  1.  1.  2. -0. -0.  0.  0.  0.]
      [ 0.  1.  0. -0.  1. -0. -0. -0.  0.]
      [ 0.  1.  0. -0.  1.  0. -0. -0. -0.]
      [-0.  0.  1.  1.  0.  0.  0.  0. -0.]
      [ 0.  1. -0.  0. -0.  0.  0. -0.  1.]
      [-0. -0.  0.  0. -0.  1.  1.  1.  0.]
      [-0. -0.  0.  0. -0. -0.  1.  1.  1.]
      [-0. -0. -0.  0. -0.  0.  0.  1.  1.]]

     Left side truncated:
      [[-0.22 -0.11]
      [-0.2  -0.07]
      [-0.24  0.04]
      [-0.4   0.06]
      [-0.64 -0.17]
      [-0.27  0.11]
      [-0.27  0.11]
      [-0.3  -0.14]
      [-0.21  0.27]

      [-0.01  0.49]
      [-0.04  0.62]
      [-0.03  0.45]]

     Sigma truncated:
      [[3.34 0.  ]
      [0.   2.54]]

     Right side truncated:
      [[-0.2  -0.61 -0.46 -0.54 -0.28 -0.   -0.01 -0.02 -0.08]
      [-0.06  0.17 -0.13 -0.23  0.11  0.19  0.44  0.62  0.53]]

     The closest rank-2 approximation to C is:
      [[ 0.16  0.4   0.38  0.47  0.18 -0.05 -0.12 -0.16 -0.09]
      [ 0.14  0.37  0.33  0.4   0.16 -0.03 -0.07 -0.1  -0.04]
      [ 0.15  0.51  0.36  0.41  0.24  0.02  0.06  0.09  0.12]
      [ 0.26  0.84  0.61  0.7   0.39  0.03  0.08  0.12  0.19]
      [ 0.45  1.23  1.05  1.27  0.56 -0.07 -0.15 -0.21 -0.05]
      [ 0.16  0.58  0.38  0.42  0.28  0.06  0.13  0.19  0.22]
      [ 0.16  0.58  0.38  0.42  0.28  0.06  0.13  0.19  0.22]
      [ 0.22  0.55  0.51  0.63  0.24 -0.07 -0.14 -0.2  -0.11]
```

```
 [ 0.1   0.53  0.23  0.21  0.27  0.14  0.31  0.44  0.42]
 [-0.06  0.23 -0.14 -0.27  0.14  0.24  0.55  0.77  0.66]
 [-0.06  0.34 -0.15 -0.3   0.2   0.31  0.69  0.98  0.85]
 [-0.04  0.25 -0.1  -0.21  0.15  0.22  0.5   0.71  0.62]]

An example query vector, original:
```

```
class DataReader:

  # Stores words used as features
  dictionary = []

  # Data Frame holding training data
  df_train = pd.DataFrame()

  # Data Frame holding training data
  df_test = pd.DataFrame()

  # Data Frame where we keep track of hits and misses
  df_compare = pd.DataFrame()

  # Data Frame containing the transformed and normalized query vectors
  qvs = pd.DataFrame

  # Count of documents in training and test sets
  n_docs = 0
  n_queries = 0

  # Settings
  k = 0
  component_names = []
  tf = False

  # SVD Factorization
  Uk = [] # Left matrix; document by factor
  Uk_normed = [] # Right matrix, document by factor, normed
  Sk = [] # Singular values
  Vk = [] # Right matrix; term by factor

  ###
  # Init
  ###
  def __init__(self):
    return


  ###
  # Train
  # Initializes df_compare
  ###
  def train(self, k, tfidf=False):
    self.k = k
    self.component_names = [f'component_{i}' for i in range(self.k)]
    self.component_names = [f'component_{i}' for i in range(self.k)]
    self.tf = tfidf
    # Collect data
    self.training_inputs()
    # Document Term Matrix
    tdm_counts = self.vectorize(self.df_train.abstract)
    # Singular Value Decomposition
    svd = self.svd(tdm_counts.toarray(), k)
    # Setup df_compare
    df_doi = pd.DataFrame(self.df_train['doi'])
    df_doi.index.name = "n"
    df_uk = pd.DataFrame(self.Uk_normed)
    df_uk.index.name = 'n'
    self.df_compare = pd.merge(df_doi, df_uk, how='left', on='n')


  ###
  # Test
  # Input: n = number of top results to match on
  ###
  def test(self, n):
    # Collect data
    self.testing_inputs()
    # Generate query vectors
    queries = self.vectorize(self.df_test['abstract'], query=True)
    qn = self.normalize_query_vectors(np.transpose(queries.toarray()))
    qvs = pd.DataFrame(qn, columns=self.df_test['test_doi'])
    qvs.index = self.component_names
    # Calculate cosine similarity
    self.cos_sim(qvs, n)


  ###
  # Read training abstracts
  # Input: File
  # Output: Abstract and DOI as strings
  ###
  def read_training_abstract(self, f):
    abstract = ""
    doi = ""
    regexp = re.compile(r'(10.(\d)+/([^(\s\>\"\<)])+)')
    metadata=True

    # Skip past the metadata at the top
    line = "."
    while metadata and line:
      line = f.readline()
      m = regexp.search(line)
      if m:
        doi = m[0]
      if line.strip() =="Abstract":
        metadata=False
    # Now read the abstract
    while line:
      line = f.readline()
      abstract += line.strip()
    return(abstract, doi)


  ###
  # Read test abstracts
  # Input: File
  # Output: Abstract and DOIS as strings
  ###
  def read_test_abstract(self, f):
    abstract = ""
    test_doi = ""
    train_doi = ""
    regexp = re.compile(r'(10.(\d)+/([^(\s\>\"\<)])+)')
    metadata=True

    # Skip past the metadata at the top
    line = "."
    while metadata and line:
      line = f.readline()
      m = regexp.search(line)
      if m and line.startswith("TrainDOI"):
        train_doi = m[0]
      elif m and line.startswith("TestDOI"):
        test_doi = m[0]
      elif m:
        print("Found unlabelled DOI")
      if line.strip()=="Abstract:":
        metadata=False

    # Now read the abstract
    while line:
      line = f.readline()
      abstract += line.strip()
    return(test_doi, train_doi, abstract)


  ###
  # Fetch abstracts from training folder
  # Side effects: Fills in df_train and n_docs
  ###
  def training_inputs(self):
    n_docs = 0
    abstracts = []
    dois = []
    titles = []

    directory = os.fsencode("/content/gdrive/MyDrive/LSIProject/dataset/papers/")
    for file in os.listdir(directory):
      fn = os.fsdecode(file)
      if fn.endswith(".txt"):
        with open(f"/content/gdrive/MyDrive/LSIProject/dataset/papers/{fn}",
                  'r') as f:
          abstract, doi = self.read_training_abstract(f)
          if not doi:
            print(f'{fn} has no doi')
          n_docs += 1
          abstracts.append(abstract)
          dois.append(doi)
          titles.append(fn)
    self.df_train = pd.DataFrame.from_dict({'doi': dois, 'title': titles,
                                            'abstract': abstracts})
    self.n_docs = n_docs


  ###
  # Fetch abstracts from test folder
  # Side effects: Fills in df_test and n_queries
  ###
  def testing_inputs(self):
    n_docs = 0
    abstracts = []
    test_dois = []
    train_dois = []
    titles = []

    directory = os.fsencode("/content/gdrive/MyDrive/LSIProject/dataset/testset")
    for file in os.listdir(directory):
      fn = os.fsdecode(file)
      if fn.endswith(".txt"):
        with open(f"/content/gdrive/MyDrive/LSIProject/dataset/testset/{fn}",
                  'r') as f:
          testdoi, traindoi, abstract = self.read_test_abstract(f)
          abstracts.append(abstract)
          test_dois.append(testdoi)
          train_dois.append(traindoi)
          titles.append(fn[:-4])
          n_docs += 1
    self.df_test = pd.DataFrame.from_dict({'test_doi': test_dois,
                                           'target_doi': train_dois,
                                           'title': titles,
                                           'abstract': abstracts})
    self.n_queries = n_docs


  ###
  # Vectorize with either counts or TF-IDF
  # Input: Column of abstracts, query or training flag,
  # threshhold is number of times a word should appear to be counted
  # Output: Document-term matrix of shape (samples, features)
  # Side effect: Stores dictionary of words used, if one is not present
  ###
  def vectorize(self, data, query=False, threshhold = 1):
    if self.tf:
      if query:
        vectorizer = TfidfVectorizer(min_df=threshhold, stop_words='english',
                                     vocabulary=self.dictionary)
      else:
       vectorizer = TfidfVectorizer(min_df=threshhold, stop_words='english')
    else:
      if query:
        vectorizer = CountVectorizer(min_df=threshhold, stop_words="english",
                                     vocabulary=self.dictionary)
      else:
       vectorizer = CountVectorizer(min_df=threshhold, stop_words="english")
    tdmatrix = vectorizer.fit_transform(data).astype(float)
    if not self.dictionary:
      self.dictionary = vectorizer.get_feature_names()
    return tdmatrix


  ###
  # Normalize Query Vector
  # Input: Matrix where each column is a raw query vector
  # Output: Matrix where each column is a query vector transformed
  # into k-space and normalized
  ###
  def normalize_query_vectors(self, query):

    # Transform into k-space
    Sk_inverse = np.linalg.inv(self.Sk)
    Vk_transposed = self.Vk
    qk = Sk_inverse @ Vk_transposed @ query

    # Normalize
    normalizer = Normalizer()
    qk_normed = normalizer.transform(np.transpose(qk))
    return(np.transpose(qk_normed))


  ###
  # SVD
  # Input: Term-document matrix, k = number of factors to reduce to
  # Side effects: Store left-matrix U, sigma matrix S, and right-matrix V
  ###
  def svd(self, tdmatrix, k):

    U0, S0, V0 = np.linalg.svd(tdmatrix, full_matrices=False)
    np.set_printoptions(precision=2)
    np.set_printoptions(suppress=True)
    Uk = U0[:, :k]
    Sk = np.diag(S0[:k])
    Vk = V0[:k, :]
    self.Uk = Uk
    self.Sk = Sk
    self.Vk = Vk

    # Normalize all vectors in Uk
    normalizer = Normalizer()
    Uk_normed = normalizer.transform(Uk)
    self.Uk_normed = Uk_normed


  ###
  # Similarity
  # If all vectors are normed, you just need Vq - a matrix vector multiplication
  # When you have many q vectors this becomes a matrix matrix multiplication
  # Input: Matrix where each column is a Query vector normalized in k-space
  # Side effects: df_test is a matrix where each column is the comparison of
  # cosine similarities between query vector and all database vectors
  ###
  def cos_sim(self, qks_normed, n):
    cossim = self.Uk_normed @ qks_normed
    cossim_abs = cossim.abs()
    answers = []

    # For top n...
    for column in cossim_abs:
      topn = cossim_abs[column].nlargest(n, keep='all')
      topn_dois = []
      for t in topn:
        idx = cossim_abs.loc[cossim_abs[column] == t].index[0]
        #print(f'Test doi {column} had max cossim of {t} at position {idx} matching t
        topn_dois.append(self.df_train.doi[idx])
      answers.append(tuple(topn_dois))
    self.df_test['answer_doi'] = answers


  ###
  # Score
  # Given a list of articles and a list of articles that they were linked to,
  # score what % returned us back to the linked "answer"
  # Input: DataFrame containing test DOI, target DOI, and DOI output
  # Output: Count of correct answers
  ###
  def score(self):
    score = 0
    count = 0
    for index, row in self.df_test.iterrows():
      #print(f'Target: #{row["target_doi"]}, Answer: #{row["answer_doi"]}\n')
      if row['target_doi'] in row['answer_doi']:
        score += 1
      count += 1
    if count == 0:
      return 0
    else:
      return score/count


  # Generate word clouds per component
  # Uses the V matrix
  # For each component, what are the top n words?
  def word_clouds(self, n):
    df = pd.DataFrame(np.transpose(self.Vk)).abs()
    df.index = self.dictionary
    display(df)

    clouds = {} # List of list of strings

    for column in df:
      cloud = []
      topn = df[column].nlargest(n, keep='all')
      for i, t in enumerate(topn):
        idx = df.loc[df[column] == t].index[0]
        #print(f'Word with significance {t} is {idx}')
        cloud.append(idx)
        clouds[self.component_names[column]] = cloud

    for c in self.component_names:
      print(f'\nWord cloud for {c}:')
      print(clouds[c])


  # Display top 5 documents per component
  # Uses the U matrix
  def document_clouds(self):
    df = pd.DataFrame(self.Uk).abs()
    df.index = self.df_train.title
    df.columns = self.component_names
    for c in self.component_names:
      display(df.sort_values(by=c, ascending=False).head())


  ###
  # Graph
  # Plot first two dimensions from document-factor matrix
  # Output: Images of 2d vectors or scatterplot
  ###
  def graph(self):
    data = self.Uk_normed
    xs = [w[0] for w in data]
    ys = [w[1] for w in data]
    xo = [0 for w in data]
```

```
    yo = [0 for w in data]

    %pylab inline
    import matplotlib.pyplot as plt
    figure()
    plt.scatter(xs, ys)
    xlabel('Component 0')
    ylabel('Component 1')
    title('TF-IDF Document-Concept Correlations over First Two Components')
    # Can label points with DOIs to identify outliers
    # for i, doi in self.df_train['doi'].iteritems():
    #   plt.annotate(doi, (xs[i], ys[i]))
    show()

    %pylab inline
    plt.figure()
    ax = plt.gca()
    ax.quiver(xo, yo, xs, ys, angles="xy", scale_units='xy', scale=1.)
    ax.set_xlim([-1,1])
    ax.set_ylim([-1,1])
    xlabel('First Component')
    ylabel('Second Component')
    title('TF-IDF Document-Concept Correlations over First Two Components')
    plt.draw()
    plt.show()


###
# Main
###
reader = DataReader()


abstracts = reader.train(2, True)


#display(reader.df_train.head())


reader.document_clouds()
```

	component_0	component_1
title		
Molecular_Mechanism_of_Evolution_He_Jiahua.txt	0.223214	0.221483
Systemic-Mucosal-Antibody-Responses_Cervia.txt	0.207835	0.015998
Structural_basis_of_receptor_Shang_Jian.txt	0.206248	0.292478
Different-Innate-Adaptive-Immune_Carsetti.txt	0.197183	0.050524
Cell_entry_mechanisms_Shang_Jian.txt	0.195851	0.240576

```
reader.word_clouds(10)
```

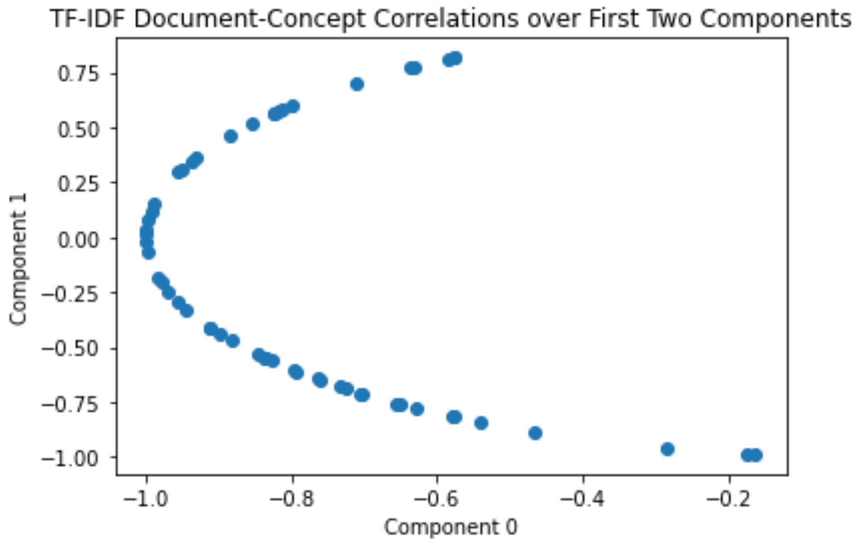
	0	1
000	0.019012	0.006573
001	0.000597	0.006115
005	0.000984	0.003135
008	0.002756	0.003512
009	0.000984	0.003135
...	...	...
î²	0.005035	0.001558
î³	0.005357	0.005907
î¼g	0.016071	0.017722
η2p	0.003934	0.012539
µg	0.034276	0.340644

2423 rows × 2 columns

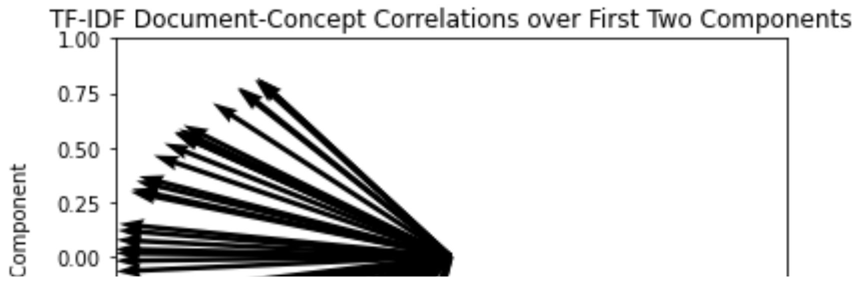
```
Word cloud for component_0:
['sars', 'cov', 'covid', '19', 'ace2', 'rbd', 'vaccine', 'coronavirus', 'patient

Word cloud for component_1:
['µg', 'vaccine', 'sars', 'dose', 'group', 'rbd', 'cov', 'ace2', 'days', '19']
```

```
reader.graph()
```

```
Populating the interactive namespace from numpy and matplotlib
```



```
Populating the interactive namespace from numpy and matplotlib
```



```
# print(reader.dictionary)
reader.test(5)
```

```
reader.n_docs
```

```
55
```

```
reader.n_queries
```

```
25
```

```
results_cv = [[],[],[],[],[]]
results_tv = [[],[],[],[],[]]
a = 54
for n in range (1, 6):
  results_cv[n-1].append(NaN)
  results_cv[n-1].append(NaN)
  results_tv[n-1].append(NaN)
  results_tv[n-1].append(NaN)
  for k in range(2, a):
    reader = DataReader()
    reader.train(k, False)
    reader.test(n)
    results_cv[n-1].append(reader.score()*100)
    reader2 = DataReader()
```

```
      reader2.train(k, True)
      reader2.test(n)
      results_tv[n-1].append(reader2.score()*100)
    chart = pd.DataFrame.from_dict({'Count': results_cv[n-1], 'TF-IDF': results_tv[n-1]
    fig = chart.plot(xlabel='Number of Latent Semantic Factors', ylabel='Accuracy perce

    fig.savefig(f'plot{n}-{a}.png')

  # print(results_cv)
  # print(results_tv)
```