A Project Report on

Latent Semantic Indexing

Mridul Gupta (2019PCP5037)
Harshal Patel (2019PCP5048)
Shabnam Ali (2019PCP5333)

Under Guidance of
Dr. Namita Mittal

Computer Science & Engineering
Malaviya National Institute of Technology
Jaipur, Rajasthan

August 20, 2020

Abstract

Latent semantic indexing (LSI) is a successful technology in information retrieval (IR) which attempts to explore the latent semantics implied by a query or a document through representing them in a dimension-reduced space. It is a mathematical method used to determine the relationship between a set of documents and the terms they contain by producing a set of concepts related to the documents and terms.The measurement of similarity between documents is usually influenced by sparseness of term-document matrix. LSI transforms the original data in a different space so that two documents/words about the same concept are mapped close (so that they have higher cosine similarity). LSI achieves this by Singular Value Decomposition (SVD) of term-document matrix. In this project we intended to focus on comparing similarity between documents with LSI technique and Topic modelling. All this is application is in NLP domain.

Chapter 1

Introduction

All languages have their own intricacies and nuances which are quite difficult for a machine to capture (sometimes they're even misunderstood by us humans!).This can include different words that mean the same thing, and also the words which have the same spelling but different meanings.

For example, consider the following two sentences:

1 I liked his last novel quite a lot.

2 We would like to go for a novel marketing campaign.

In the first sentence, the word 'novel' refers to a book, and in the second sentence it means new or fresh.

We can easily distinguish between these words because we are able to understand the context behind these words. However, a machine would not be able to capture this concept as it cannot understand the context in which the words have been used. This is where **Latent Semantic Analysis (LSA)** comes into play as it attempts to leverage the context around the words to capture the hidden concepts, also known as topics.

So, simply mapping words to documents won't really help. What we really need is to figure out the hidden concepts or topics behind the words. LSA is one such technique that can find these hidden topics.

Chapter 2

Algorithms Used

We have implemented three Latent Semantic Indexing algorithms - **Word Frequency, TF-IDF and SVD**.

2.1 Word Frequency

1. Get input document and number of sentences in a summary(n)
2. Find the word tokens
3. Remove the stopwords
4. Calculate frequency of each word and create a word frequency table
5. Find the sentence tokens
6. Score the sentences by calculating term frequency
7. Generate summary - Display top n sentences based of sentence scores

2.2 TF-IDF

TF-IDF algorithm is made of 2 algorithms multiplied together. Term Frequency and Inverse document frequency Term frequency is how common a word is and inverse document frequency (IDF) is how unique or rare a word is.
Here we considered, number of documents as number of sentences.

- **Term frequency (TF)**

$$TF(t) = \frac{Number\ of\ times\ term\ t\ appears\ in\ a\ document}{Total\ number\ of\ terms\ in\ the\ document}$$

- **Inverse document frequency (IDF)**

$$IDF(t) = log_e(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ t\ in\ it})$$

- **TF-IDF**

$$TF - IDF(t) = TF(t) * IDF(t)$$

1. Get input document and number of sentences in a summary(n)
2. Tokenize sentences
3. Create the Frequency matrix of the words in each sentence
4. Calculate Term Frequency and generate a matrix
5. Create a table for documents per words
6. Calculate IDF and generate a matrix
7. Calculate TF-IDF and generate a matrix
8. Score the sentences
9. Generate summary - Display top n sentences based of sentence scores

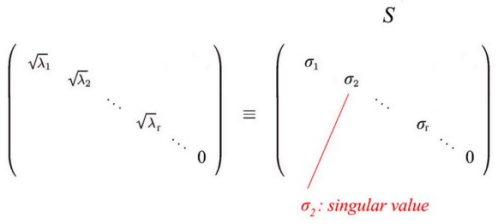
2.3 Singular Value Decomposition (SVD)

SVD is the problem of finding eigen values and vectors of a matrix of size term x term and it cannot be done analytically, thus it is done iteratively each time reducing the error, training the svd-transformer.
With all large set of raw data collected, how can we discover structures in it? This becomes even harder for high-dimensional raw data. It is like finding a needle in a haystack. SVD allows us to extract and untangle information.
SVD states that any matrix A can be factorized as:

$$A = USV^{\mathrm{T}}$$

where U and V are orthogonal matrices with orthonormal eigenvectors chosen from AA^{T} and $A^{\mathrm{T}}A$ respectively.

S is a diagonal matrix with r elements equal to the root of the positive eigenvalues of AA^{T} or $A^{\mathrm{T}}A$ (both metrics have the same positive eigenvalues anyway). The diagonal elements are composed of singular values.

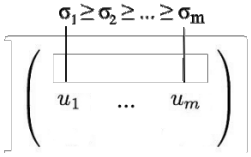


i.e. an $m \times n$ matrix can be factorized as:

$$A_{m\times n} = U_{m\times m} S_m \times_n {V^{\mathrm{T}}}_{n\times n}$$

$$\underset{m\times n}{A} = \underset{m\times m}{U}\ \underset{m\times n}{S}\ \underset{n\times n}{V^{\mathrm{T}}}$$

We can arrange eigenvectors in different orders to produce U and V. To standardize the solution, we order the eigenvectors such that vectors with higher eigenvalues come before those with smaller values.



Comparing to eigen-decomposition, SVD works on non-square matrices. U and V are invertible for any matrix in SVD and they are orthonormal.
After svd, the term document vectors are mapped to a reduced size vector such that co-occuring terms all represent one dimension of the vector. This is called a topic.

Chapter 3

Implementation

3.1 Steps involved in the implementation of LSI

Let's say we have m number of text documents with n number of total unique terms (words). We wish to extract k topics from all the text data in the documents. The number of topics, k, has to be specified by the user.

- Generate a document-term matrix of shape m x n having TF-IDF scores.
- Then, we will reduce the dimensions of the above matrix to k (no. of desired topics) dimensions, using singular-value decomposition (SVD).
- SVD decomposes a matrix into three other matrices. Suppose we want to decompose a matrix A using SVD. It will be decomposed into matrix U, matrix S, and V^T (transpose of matrix V). Each row of the matrix U_k (document-term matrix) is the vector representation of the corresponding document. The length of these vectors is k, which is the number of desired topics. Vector representation for the terms in our data can be found in the matrix V_k (term-topic matrix).

$$A = USV^T$$

- So, SVD gives us vectors for every document and term in our data. The length of each vector would be k. We can then use these vectors to find similar words and similar documents using the cosine similarity method

3.1.1 Python Implementation

1. **Data Preprocessing:** To start with, we will try to clean our text data as much as possible. The idea is to remove the punctuations, numbers, and special characters all in one step using the regex replace(\[^{a}−zA − Z]","\), which will replace everything, except alphabets with space. Then we will remove shorter words because they usually don't contain useful information. Finally, we will make all the text lowercase to nullify case sensitivity.
   It's good practice to remove the stop-words from the text data as they are mostly clutter and hardly carry any information. Stop-words are terms like 'it', 'they', 'am', 'been', 'about',

'because', 'while', etc.

To remove stop-words from the documents, we will have to tokenize the text, i.e., split the string of text into individual tokens or words. We will stitch the tokens back together once we have removed the stop-words.

2. **Document-Term Matrix:** We will use sklearn's TfidfVectorizer to create a document-term matrix with 1,000 terms. We can also use all the terms to create the matrix but that would need quite a lot of computation time and resources. Hence, we have restricted the number of features to 1,000. If we have the computational power,We can try out all the terms.

3. **Model Construction:** The next step is to represent each and every term and document as a vector. We will use the document-term matrix and decompose it into multiple matrices. We will use sklearn's TruncatedSVD to perform the task of matrix decomposition.
The components of *svd_model* are our topics

Chapter 4

Technologies Used

- **Python3** - Python is an interpreted, high-level, general-purpose programming language. Python's large standard library, commonly cited as one of its greatest strengths provides tools suited to many tasks. It can serve as a scripting language for web applications.
- **NLTK** - The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing for English written in the Python programming language.
- **FPDF** - PyFPDF is a library for PDF document generation under Python, ported from PHP
- **Numpy** - NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- **Sklearn** - Scikit-learn is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy
- **Matplotlib** - Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

Chapter 5

Project Screenshots

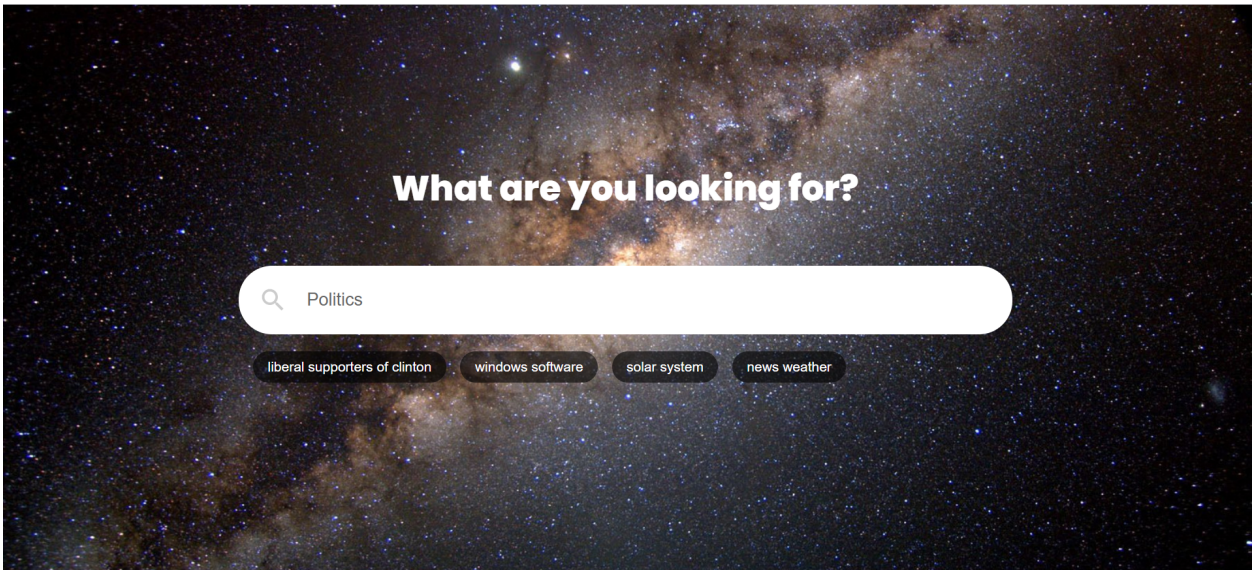


Figure 5.1: First Page

This is the first page(index page) of our project. Users can type their query in the search bar and after pressing enter all the documents related to the query will shown on the next page along with their cosine similarity. We have also mentioned some of the suggestions below the search bar for the user.

The next page are the results for our query. A number of documents are shown as a result and are rank by using cosine similarity. The document having the highest similarity will appear at the top and the same fashion is followed for all the documents. Users can also download or print the documents if they wished to.

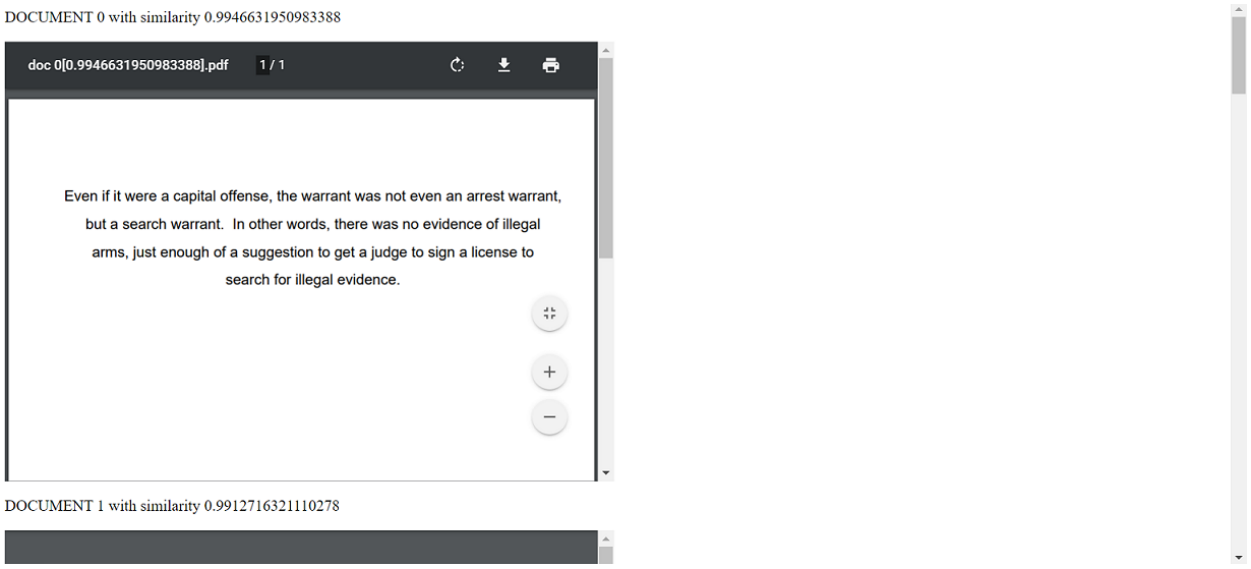


Figure 5.2: Results for the query

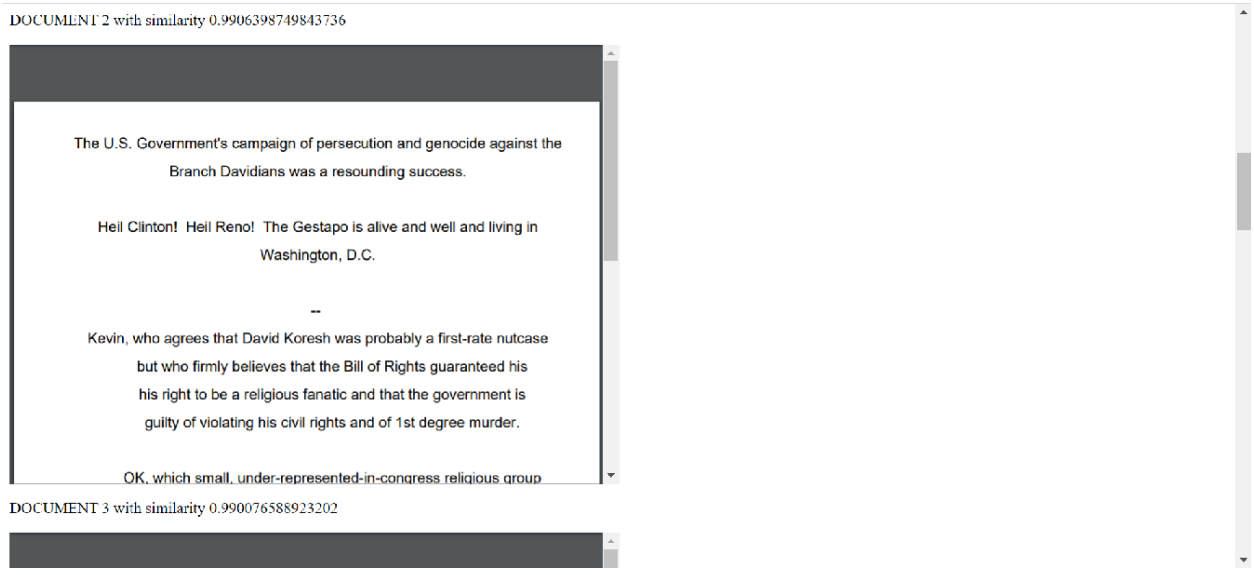


Figure 5.3: Results for the query

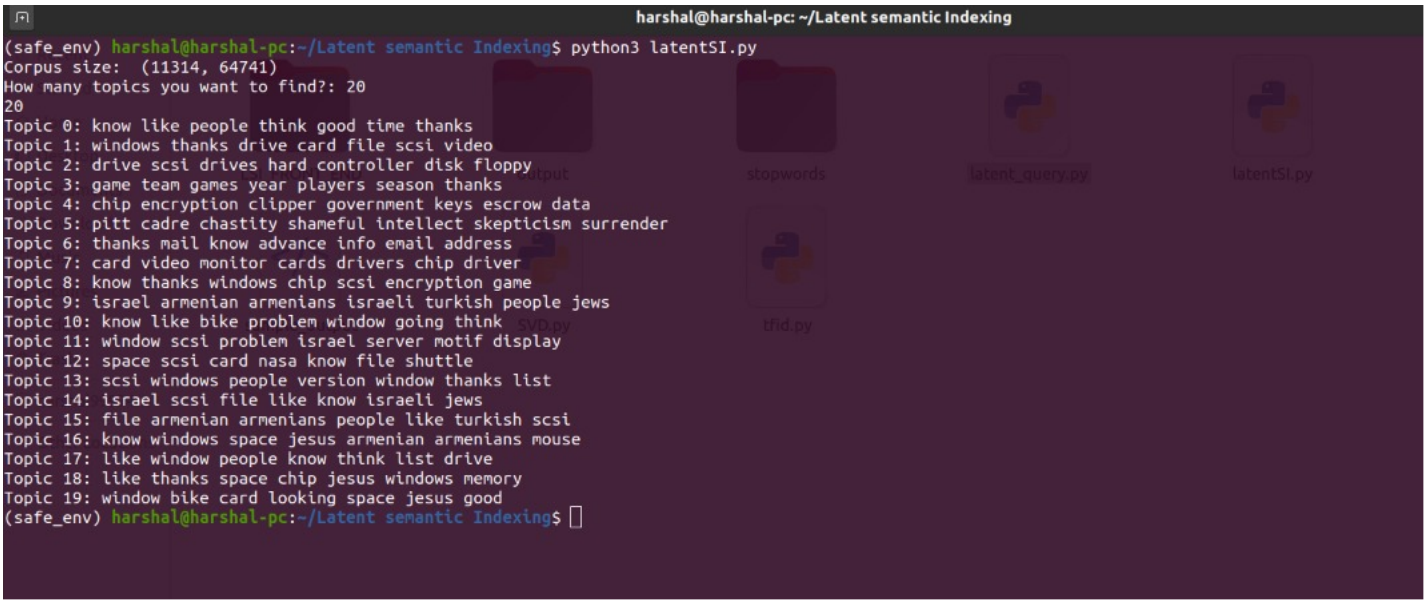


Figure 5.4: Topic Modelling

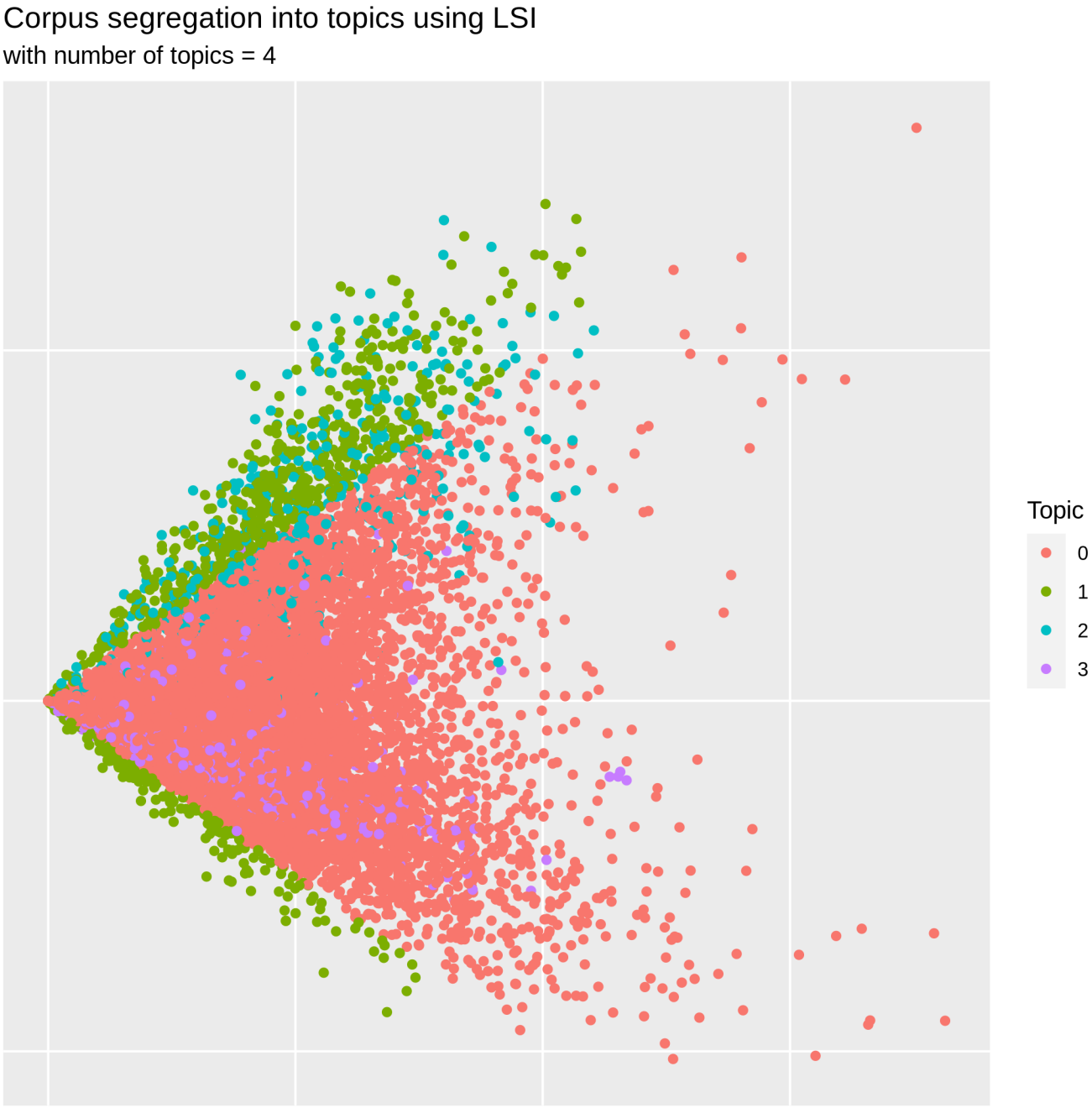


Figure 5.5: Corpus Segregation into topics using LSI

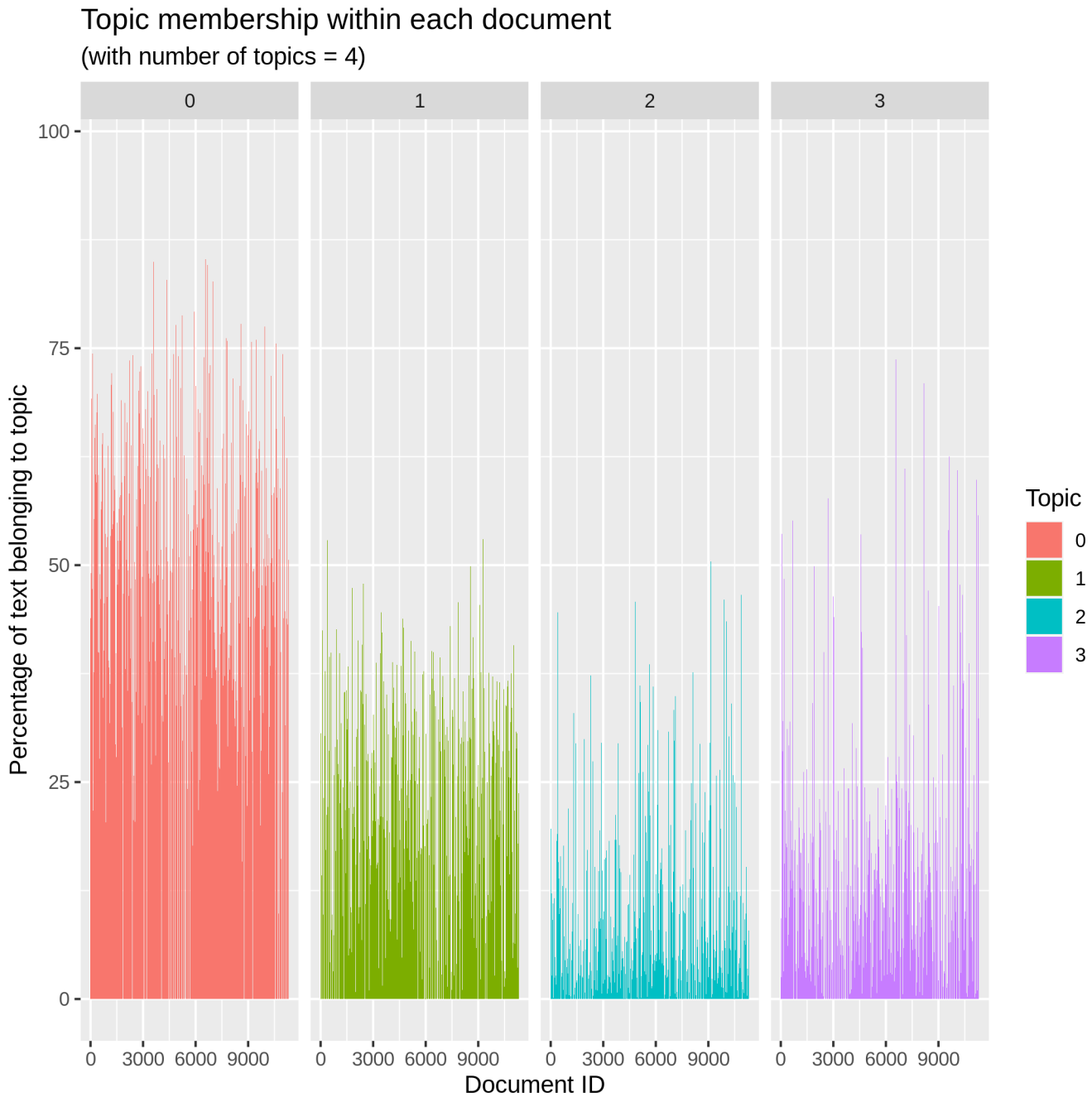


Figure 5.6: Topic membership with each document

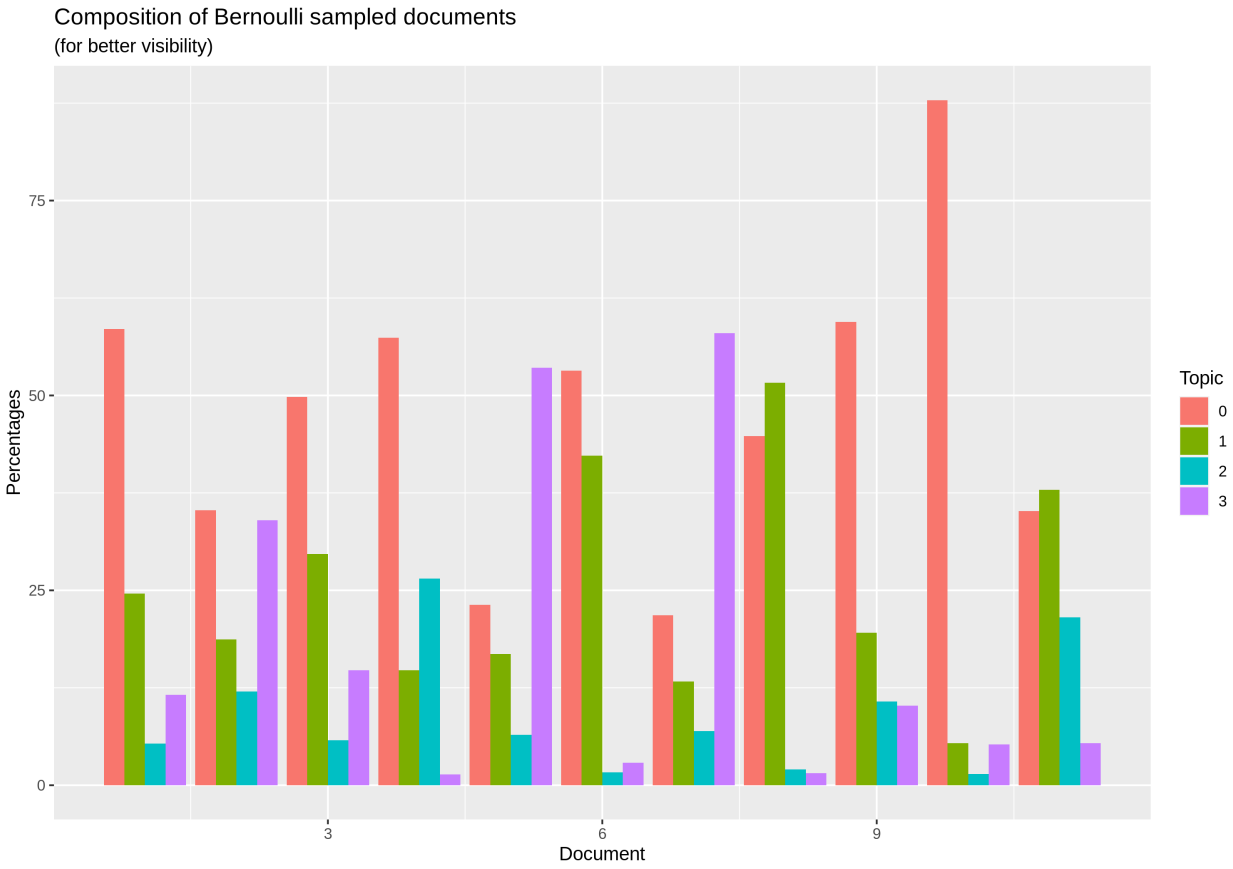


Figure 5.7: Composition of Bernoulli sampled document

Chapter 6

Conclusion

LSI is fast and easy to implement. It gives decent results, much better than a plain vector space model. LSI involves SVD, which is computationally intensive and hard to update as new data comes up. We have used algorithms like TF-IDF, SVD for topic modelling and comparing similarity between documents with LSI technique.

There is always a scope for improvement and in this project there is scope to improve it by implementing stemming/lamentization.

SVD is quite complex for implementing on large metrices. It also needs a fixed size of corpus that's another limitation.