Addressing Limitations of Vector Space Model using Latent Semantic Indexing

Harish Iniyarajan [1] [CH18B005] and Vishal Rishi [2] [CH18B013]

[1,2] *Indian Institute of Technology, Madras*

Abstract: Compare the performance of Vector Space Model and Latent Semantic Indexing (LSI) and understand why LSI performs better.

Keywords: *Information Retrieval · Vector Space Model · Latent Semantic Indexing*

1 Introduction

Vector Space Model (VSM) is the first non-trivial algebraic model encountered for building an information retrieval system. While VSM does offer an innovating and interesting approach – representing documents and terms as vectors of identifiers – it does suffer some important issues. It fails to understand the context of the query and the documents, thereby failing to mimic humans in retrieving relevant documents. Latent Semantic Indexing (LSI), on the other hand, works under the assumption that words that are similar in meaning occur in similar pieces of text. It attempts to capture the underlying or latent structure in word usage that is partially obscured by variability in word choice. Capturing of this latent structure improves the performance of the information retrieval system significantly.

2 Problem Definition and Motivation

Vector Space Model has the following limitations:

- It is semantically sensitive. Documents with similar context but with different vocabulary will not be associated. This also implies that synonymy and polysemy are not captured.
- The resulting identifier vectors that span the vector space, usually the term vectors, are not orthogonal. This again is indicative of similar terms not being associated together.

- This model does not capture co-occurrence of terms. Similar terms appearing together in similar documents indicates similarity in meaning of these words. This is not captured by VSM.

The nDCG@10 score of Vector Space Model on Cranfield dataset was 0.27. This is not a satisfactory performance and leaves room for great improvements. This served as a motivation to seek models that perform better and address the above-mentioned shortcomings of VSM.

Most information retrieval systems perform poorly when queries are erroneous and meaningless. Introducing a spell check algorithm would help in improving the quality of queries and thereby improve the performance of the information retrieval system.

3 Background and Related Work

Both Vector Space Model and Latent Semantic Indexing rely on term-document matrix. This matrix could be filled with binary values – 0 indicating absence of the term in the document and 1 indicating the presence of the term in that document. However, tf-idf is a more popular and effective weighting methodology.

3.1 TF-IDF Representation

The values in term-document matrix are replaced by *term frequency – inverse document frequency (tf-idf)* score instead of primitive binary values. This addresses two important issues faced in binary value representation:

- Terms that occur in every document contain little information and add no value.
- Terms that rarely occur in documents do not contain sufficient information and add little value.

Such terms must be given smaller values in the term-document matrix. These issues are addressed by *tf-idf* score in the following way:

- *Term frequency (tf)* – This is a count of the number of times a term appears in a particular document. Terms that do not occur frequently will

have a smaller *tf* score, thereby reducing the importance given to such terms.

- *Inverse document frequency (idf)* – It diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely. It is calculated as follows:

$$idf = \log\left(\frac{N}{n}\right)$$

$$where, N = total\ number\ of\ documents$$

$$n = number\ of\ documents\ in\ which\ a\ particular\ term\ occurs$$

The *tf-idf* score is the product of term frequency value and inverse document frequency value. This is calculated for every term occurring in the document set and the term-document matrix is populated with these values. Combination of term frequency and inverse document frequency values ensure that frequently occurring terms as well as sparsely occurring terms are not given excess importance.

Table 1. Sample tf-idf matrix for the following documents

Documents	
S_1	Herbivores are typically plant eaters and not meat eaters.
S_2	Carnivores are typically meat eaters and not plant eaters.
S_3	Deers eat grass and leaves.

Terms	Documents		
	S_1	S_2	S_3
Herbivore	0.4771	0	0
Typical	0.1761	0.1761	0
Plant	0.1761	0.1761	0
Eat	0	0	0
Meat	0.1761	0.1761	0
Carnivore	0	0.4771	0
Deer	0	0	0.4771
Grass	0	0	0.4771
Leaf	0	0	0.4771

3.2 Vector Space Model

In VSM, documents in the document set are represented as vectors in a vector space spanned by all the terms. Each term is a separate dimension and terms are assumed to be orthogonal. The weight of each document vector along a particular dimension is the corresponding *tf-idf* score.

For queries, *idf* scores are taken from the term-document matrix and are multiplied with the corresponding *tf* score from the queries. Using this, queries can be represented as vectors in the space spanned by all the terms. As documents and queries are projected in the same vector space, similarity between queries and documents can be calculated to retrieve most relevant documents for each query. Similarity measure are discussed in section 3.4.

3.3 Latent Semantic Indexing

LSI also uses the term-document *tf-idf* matrix. This matrix is decomposed into three other matrices of very special form, U, $\sum$, and V matrices, using SVD decomposition [1]. These original matrices show a breakdown of the original relationships into linearly independent components or factors [2]. Many of these components are very small and are ignored, leading to an approximate model that contains fewer dimensions.

These independent components or factors, often referred to as concepts, are assumed to capture the latent structure and are orthogonal to each other. U matrix contains term-concept weights and V matric contains document-concept weights. $\sum$ matrix contains all the singular values which are used to decide significant and insignificant dimensions.

Let d be the number of documents, t be the number of terms, n be the number of concepts and k be the reduced number of concepts. V matrix, which initially of size $d \times n$ is truncated to include only first k dimension. Multiplying this truncated V matrix $(d \times k)$ with truncated $\sum$ matrix $(k \times k)$, we get a new matric which contains all document vectors in the new k-dimensional space. All the documents in this matrix can be projected in a vector space spanned by these k concepts.

For every query, it's *tf-idf* representation is calculated and is then projected to the k-dimensional space by multiplying this query vector with the truncated U matrix $(t \times k)$. After projecting all the queries in this k-dimensional space, similarity between queries and documents can be calculated to retrieve the most relevant documents for a every query.

3.4 Similarity Measures

There are several similarity measures which find their applications in different scientific and engineering fields. Most popular among them are Euclidean distance [3], Jaccard similarity [4] and cosine similarity. For our purposes, cosine similarity is best suited. Cosine similarity between any two vectors A and B can be calculates as:

$$sim(A, B) = \cos\theta = \frac{A \cdot B}{|A||B|}$$

This yields a value between -1 and 1 and documents can be ranked based on these values.

4 Proposed Methodology

We propose that Latent Semantic Indexing performs better than Vector Space Model in information retrieval task, on the evaluation measure nDCG, on Cranfield dataset under the assumption that terms that are similar in meaning occur in similar documents. This addresses the following issues faced by VSM:

- It is not semantically sensitive. Documents with similar context but with different vocabulary will be associated. This helps it capture synonymy.
- The resulting identifier vectors that span the vector space are orthogonal.
- LSI can capture higher order co-occurrences.

This model does not capture polysemy. As Cranfield is a highly domain specific dataset, we are assuming that presence of polysemous words will be rare and the performance of LSI will not be limited to a great extent. Moreover, the presence of a latent structure will ensure that the performance of LSI will be better than VSM.

As mentioned in the problem definition, poor quality of queries limit the performance of IR systems. As we are working with Cranfield dataset which is highly domain specific, it is unlikely to encounter homophones. Therefore, detecting and correcting non-words in queries should greatly improve the quality of queries along with other common pre-processing methods. To achieve this, we propose to use a spell checker which uses Bayesian model to choose the potentially correct word among many other possibilities [5].

5 Experiments

5.1 Pre-processing

Documents and queries are likely to contain noise in forms of stop-words, undesired characters, punctuations etc. Non-removal of these noise will hinder the performance of information retrieval methods. Some pre-processing techniques that were used are discussed below.

Tokenization

Tokenization is breaking the raw text into small chunks called tokens. Various tokenization techniques like white space tokenization, rule based tokenization, dictionary based tokenization etc could be used. For our purpose, we used Regular Expression Tokenizer [6] and Penn Treebank Tokenizer [7]. Numbers are also removed separately in this process.

Stop-words removal

Stop words are commonly used words in a language which do not add any additional information to the information retrieval system. A curated set of stop-words, that match the domain of the dataset could be used. We have used NLTK's list of stop-words [8] and have added additional tokens which were encountered in the dataset. Tokens that are present in the stop-words set are removed from the documents and queries.

Lemmatization

Inflection reduction is a necessary pre-processing technique to reduce words to their base forms. Out of lemmatization and stemming [9], we have used lemmatization in order to preserve the meaning of the base word. However, there was no significant difference in performance when stemming was use instead of lemmatization. After stop-word removal, each token is reduced to it *lemma*, which is its base form.

Spell-checker

As noted earlier, quality of queries affects the performance of information retrieval system. TextBlob [10] provides a simple API for diving into common Natural Language Processing tasks, which include Bayesian model for spelling correction. Calculating prior probabilities in Bayesian model demands knowing how many times each term occur on some reference corpus. We have trained this model on Cranfield dataset and the file named *train.txt* contains the frequency of terms encountered in the dataset. Note that, this needs to be trained only once. If *train.txt* is lost or needs to be generated for a different dataset, consider running the function that we have provided in the code.

For a given word which is erroneous, TextBlob's *suggest* method returns a list of (word, confidence) tuples which are sorted based on confidence values. The word with highest confidence is chosen to replace the erroneous word.

5.2 Latent Semantic Analysis

After the documents and queries are pre-processed, the *tf-idf* matrix is generated using Scikit-learn's *Tfidfvectorizer* [11]. After attaining the *tf-idf* score for queries and documents, the *tf-idf* matrix is decomposed using SVD decomposition into U, $\sum$ and V matrices. The top *k*-components are selected and the corresponding document-concept matrix is found over this *k*-dimensional space. By considering only the first *k* dimensions, the corresponding representation for queries are found in the *k*-dimensional space.

The value of *k* was tuned to give the best performance. nDCG@10 vs *k* was plotted and the most optimal value of *k* was found (see Fig. 1).

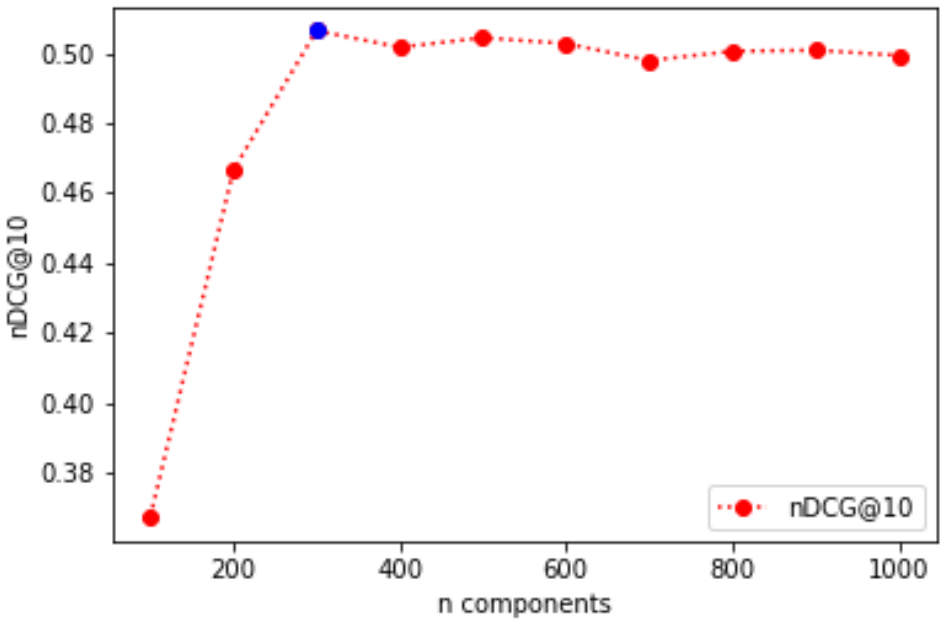

Fig. 1. nDCG@10 vs k. It can be seen that optimal performance is achieved for k = 300.

Hence, *k* was set to 300 and the documents and queries were projected onto this 300-dimensional space. Cosine similarity between the queries and documents were computed and the documents were ranked accordingly.

6 Results

nDCG score was our preferred metric for evaluation. The nDCG@10 score for VSM was 0.27 while that for LSA was 0.51. We observed that nDCG score monotonically increases with increased number of positions. That is, nDCG@k increases monotonically as k increases (see Fig. 2). Considering this, the best nDCG score achieved at k = 1400 and this value in VSM was 0.48 and in LSA was 0.65.

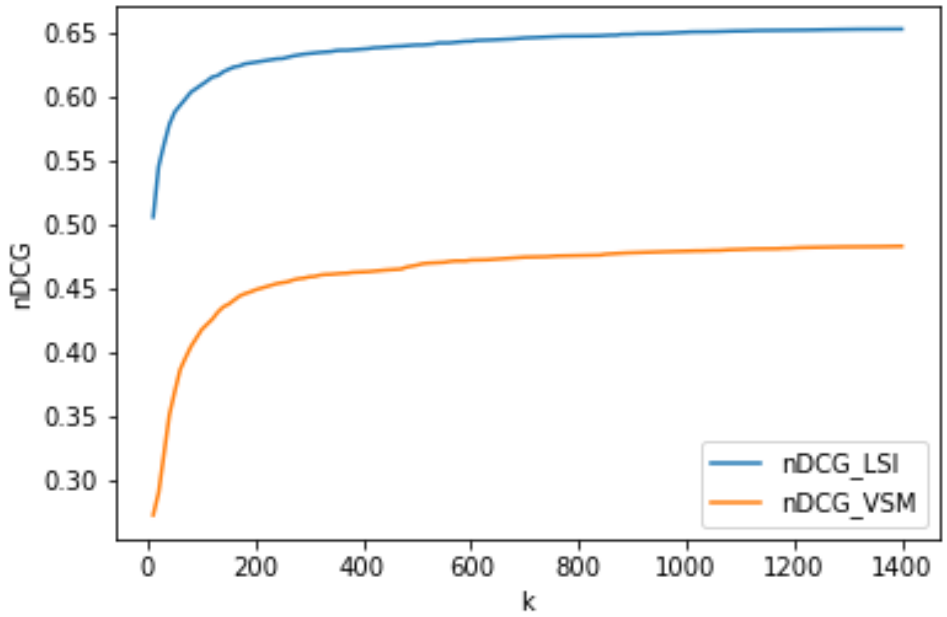

Fig. 2. nDCG@k vs k. It can be seen that the value of nDCG keeps increasing and achieves it maximum at k = 1400.

There is a significant increase in the nDCG score for LSA when compared to VSM. This validates our assumption that terms that are similar in meaning occur in similar documents and hence, there is indeed a latent structure which is being captured by LSA.

To understand why LSA performs better than VSM, we could query-wise nDCG score of both models. LSA and VSM do perform similarly for many queries (see Fig. 3) but also vary significantly for few queries (see Fig. 4). On analysing the responses of VSM and LSA over queries where their performances are significantly different, we can notice the following:

- Tokens of these queries, when considered individually, do not capture the context of the queries well. For example, in query 209, the tokens would be [‘physical’, ’characteristic’, ‘significance’, ‘separate’, ‘laminar’, ‘layer’, ‘turbulent’, ‘boundary’, ‘flow’]. If the interaction between these tokens is not considered and if they are modelled as though they were independent, we miss the context of the query. Different combination of these tokens given rise to different contexts and assuming that these tokens are independent implies that the context is independent of the interaction between these tokens. This is a reason why VSM performs poorly for this query and LSA performs better.

- When queries and documents are long, VSM fails to capture short-term and long-term dependency. Such dependencies can be partially modelled by the latent structure and LSA performs better. For example, query 206 is "how do subsonic and transonic flutter data measured in the new langley transonic dynamics tunnel compare with similar data obtained in other facilities." Here, the word "data" refers to the subsonic and transonic flutter data which VSM doesn't model at all. As the words "subsonic", "transonic", "flutter" occurs together with the word "data" in many documents, LSA understands that these words are related. This leads to a significant improvement in the performance of LSA over VSM.

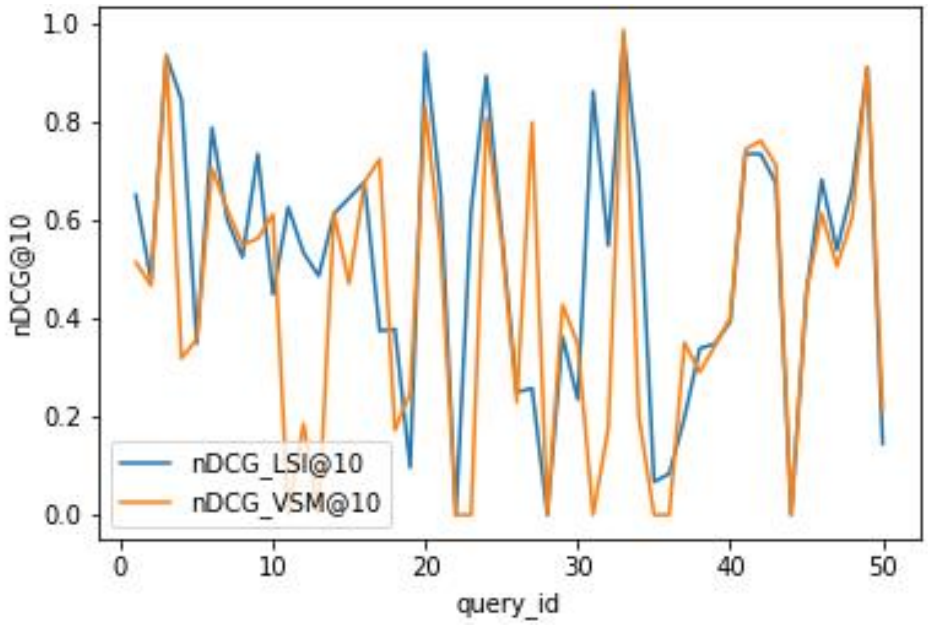

Fig. 3. Comparison of nDCG scores of VSM and LSA over queries where they perform similarly

The queries over which both the models perform similarly are fairly simple queries. They don't exhibit high word interactions and word dependencies. For such queries and documents, terms can be assumed to be independent and modelled accordingly.

The reasons behind "moderate" performance of LSA are yet to be explored. While LSA did offer a significant improvement, it's best nDCG score was only 0.65, which leaves room of more improvement. One possibility could be the presence of polysemous words. While we were unable to spot any polysemous words and

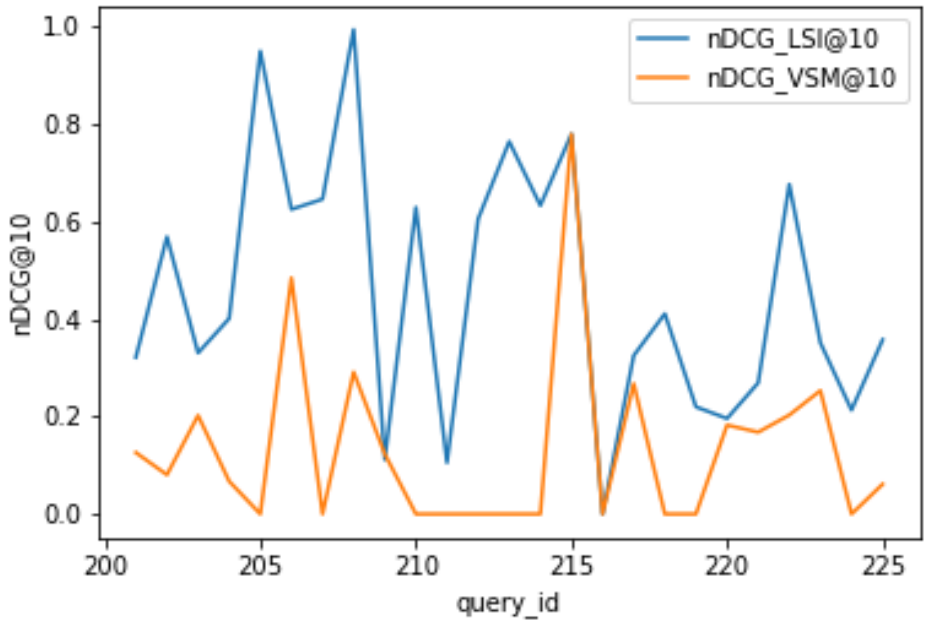

Fig. 4. Comparison of nDCG scores of VSM and LSA over queries where they perform significantly different.

assumed that they occur rarely, our assumption could be wrong and integrating WordNet [12] would give us a better idea. If we represent each sense of every term as a separate dimension, we could perfectly capture polysemy.

We also have to consider the effects of pre-processing techniques. While we chose a specific combination of inflection reduction and tokenization, a different combination might result in better performance. Penn Treebank Tokenizer is dependent on the corpus it is trained on. Using a different tokenizer which is specially designed considering the domain of the dataset would improve the performance.

There are several other pre-processing techniques, which could affect the performance. While we removed numbers and other insignificant characters, they do play a very important role in scientific literature. This could be detrimental to performance of LSA on Cranfield dataset specifically. We also curated a special set of stop-words, which is not perfect. Improvements along these areas can improve the performance of LSA significantly.

7 Conclusion

Latent Semantic Indexing does offer superior performance compared to Vector Space Model. Failure of VSM to capture the similarities between words and documents has limited its performance significantly. LSA, on the other hand, assumes the presence of a latent structure and captures it using SVD decomposition and low-rank approximation. This allows LSA to understand the context of queries and documents to a certain extent and this boosts its performance significantly.

Another important issue to be taken care of is the noise present in the documents and queries. Although perfect pre-processing is seldom possible, there should be significant efforts made to reduce the noise present in documents and queries. This is should also ensure that documents and queries do not lose their context and parts which do not contribute any information must be removed. Such techniques are highly dependent on the purpose of the experiment and methods must be chosen accordingly. These noise can severely detriment the performance of information retrieval system.

There is still scope for improvement. LSA, even though not a perfect technique, offered significant improvements in performance when compared to the standard Vector Space Model.

References

1. 'Singular value decomposition' (2021). Wikipedia. https://en.wikipedia.org/wiki/Singular_value_decomposition
2. Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer & Richard Harshman (1990). Indexing by Latent Semantic Analysis. In *Journal of the American Society for Information Science*.
3. 'Euclidean distance' (2021). Wikipedia. https://en.wikipedia.org/wiki/Euclidean_distance
4. 'Jaccard Index' (2021). Wikipedia. https://en.wikipedia.org/wiki/Jaccard_index
5. Peter Norvig (2007). 'How to Write a Spelling Corrector'. http://norvig.com/spell-correct.html
6. NLTK 3.6 documentation, Regexp tokenizer (2020). https://www.nltk.org/_modules/nltk/tokenize/regexp.html

7. NLTK 3.6 documentation, Penn Treebank Tokenizer (2020). https://www.nltk.org/_modules/nltk/tokenize/treebank.html
8. NLTK Stop word corpus. https://www.nltk.org/nltk_data/
9. 'Stemming and Lemmatization' (2008). Stanford NLP. *Cambridge University Press*. https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html
10. TextBlob - https://textblob.readthedocs.io/en/dev/#
11. Scikit-learn, Tfidfvectorizer - https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
12. Wordnet (2021). Princeton University. https://wordnet.princeton.edu/