CS6370 : Natural Language Processing
Report for Final Project

Varun Gumma CS21M070
Harsha Vardhan Gudivada CS21M021

1 *Abstract*

This project involves building an Information Retrieval system employing Natural Language Processing techniques such as Vector Space Model, Latent Semantic Analysis(LSA). The objective of the IR system is to return the most relevant documents from the collection to answer the query. The dataset which has been provided is the Cranfield Dataset. We compared the performance of Vector Space Model and Latent Semantic Analysis (LSA) along with spell check and understand why LSA performs better.

2 Vector Space Model

The primitive/baseline model we consider is the Vector Space Model (VSM). In VSM, the unique-terms in the documents form the basis and the documents are represented as vectors (linear combinations of these basis words) as shown in the figure below.

Is the figure correct?

VSM is a very naive representation of the documents as it assumes that the terms are independent of one another while this is not true in reality. There will be a lot of dependencies between the words. For example, the words `fluffy` and `soft` are synonyms, but VSM represents them as orthogonal basis, i.e. they are not related in any way.

VSM becomes computationally intensive as the number of terms (number of basis vectors) increases. This is a classic example of the *Curse of Dimensionality*. The documents are retrieved only if the query terms *exactly* match the terms in the documents, even synonyms won't work. This representation of the documents is also oblivious to the sequence in which the words appear.

**How does sequence matter in a query?
So, is VSM is a Bag of words model without any consideration for the order of words?**

The weights (or coefficients) are described by the TF-IDF scores which are given as $tf \cdot \ln\left(\frac{N}{n}\right)$. To *smoothen* the values, we use the formula $tf \cdot (1 + \ln(\frac{N+1}{n+1}))$. The TF-IDF matrix mentioned before can be rather sparse and does not give us a proper relation between words & topics (term-topic matrix) and topics & documents (topic-document matrix).

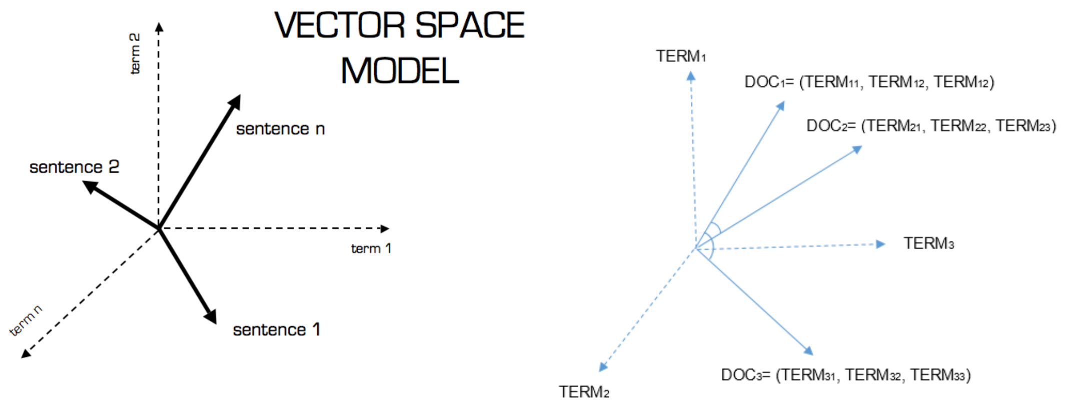
VECTOR SPACE MODEL
term 2
sentence n
sentence 2
term 1
term n
sentence 1
TERM1
DOC1= (TERM11, TERM12, TERM12)
DOC2= (TERM21, TERM22, TERM23)
TERM3
DOC3= (TERM31, TERM32, TERM33)
TERM2

Figure 1: Pictorial representation of VSM

3 Latent Semantic Analysis

As seen with the Vector Space Model, the largest challenges associated with information retrieval are synonymy and high dimensionality. An elegant and accurate technique to solve both problems has been presented in the form of Latent Semantic Analysis (LSA).

Latent Semantic Analysis (LSA) 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.

So, LSA is a topic modeling technique that uncovers these latent topics. Any topic modeling algorithm is built around the idea that the semantics of our document is actually being governed by some hidden, or *latent* variables that we are not observing directly after seeing the textual material.

LSA tries to leverage the context around the words to capture the hidden or latent concepts, which are called *topics*. So, if we simply mapped the words to documents, like in Vector Space Modelling, then it won't really helpful for us. So, what we really require is to extract the hidden concepts or topics behind the words.

LSA uses the term-document TF-IDF matrix. However, as seen in the Vector Space Model, the document term matrix is very sparse, noisy and is redundant across its many dimensions. As a result, to find the few latent topics that capture the relationships among the words and documents, we want to perform dimensionality reduction on document term matrix. Then, we have to reduce the dimensions of the above document term matrix to k, which specifies the number of desired topics) dimensions.

So, this matrix is decomposed into three other matrices of very special form, U, Σ, and

V matrices, using SVD decomposition. These original matrices show a breakdown of the original relationships into linearly independent components or factors. 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 matrix contains document-concept weights. Σ matrix contains all the singular values which are used to decide significant and insignificant dimensions.

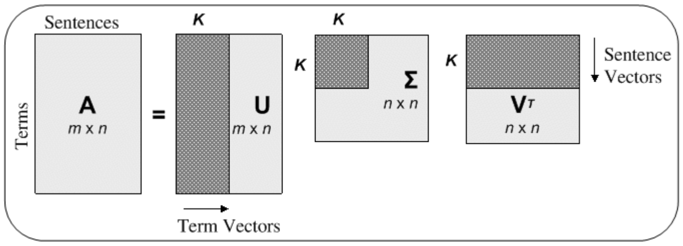

Figure 2: Pictorial representation of LSA with SVD

4 Comparison of VSM and LSA

In addition to LSA, we also implemented spellcheck as well and performed an ablation study. Applying spellcheck always degrades the performance (metrics) by 2% . We reason this may be because the spellcheck model corrects the word, but does not consider the context, i.e. it just corrects to the closest word alphabetically. For example, the query `How does aeodyamics resove wnds` was corrected to `How does aerodynamics remove ends` instead of the true query `How does aerodynamics resolve winds`.

We ran a gridsearch over multiple values of `n_comp`, i.e. the number of components to use for SVD and we observe the trend as shown below. Further, we modified the stop word removal program to remove strings (words) with only one character. This boosted the performance by 2%. We observe a decaying trend as we increase the `n_comp` values, i.e., more singular values, the more we approach the the primitive VSM (which uses the whole TF-IDF matrix without reducing the dimension). LSA outperforms VSM by 2% in terms of nDCG. The trends of LSA and VSM are almost the same (relatively) with and without spellcheck. We reason that spellcheck in this setting (with Cranfield Dataset and checking for every query) will not make sense as they queries provided are already well-formed with no errors and occasionally the spellcheck module may "uncorrect" correct spellings in the query in an attempt to correct them. Further, using stemming or lemmatization did not affect the performance much, they both produced almost similar results.

We also tested the position of spellcheck, i.e. the optimal step at which spellcheck should be performed. We found that when using lemmatization, the position of spellcheck did not matter, and it could be performed anywhere after tokenization. But for stemming, if we perform spellcheck anywhere after inflection-reduction we see a drop of 11% and we only upon performing spellcheck after tokenization and before stemming, we see a performance of 0.47 (nDCG).

This is because lemmatization produces a *proper* reduction unlike stemming which is *rough*. For example, `surfaces` is reduced to `surface` with lemmatization whereas with stemming it is reduced to `surfac`. In this case, spellcheck corrects it appropriately. But the word `curving` is reduced to `curv` with stemming and it is corrected to `cure` which is total incorrect (original context was `curving wings`). Similarly, `turbulances` was stemmed to `turbul` and corrected to `turbo`. These cause the query to lose its meaning and many irrelavant documents are retrieved and hence the low recall, precision and nDCG.

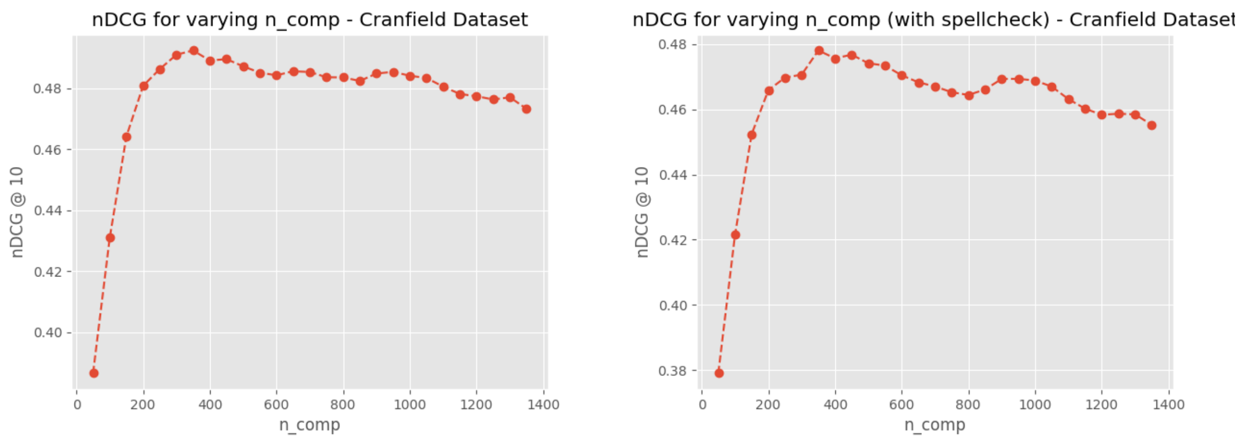

Figure 3: `nDCG @ 10 vs n_comp` with and without spellcheck

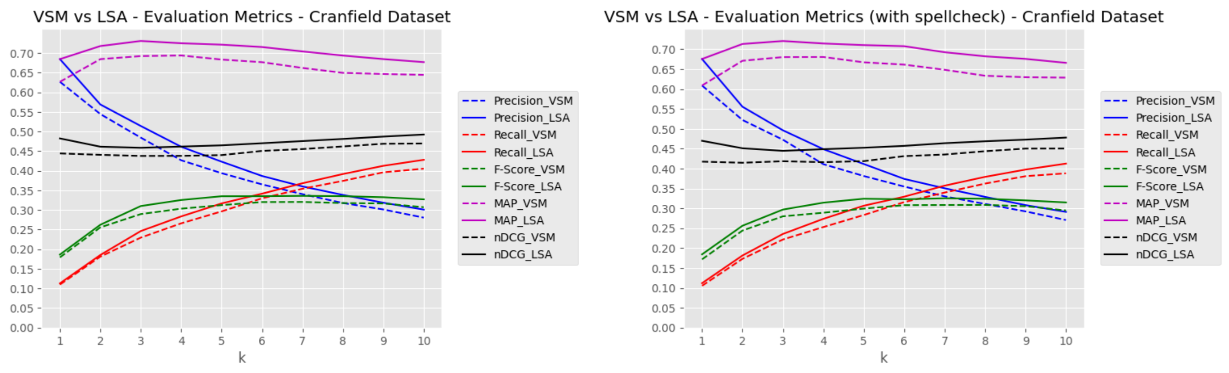

Figure 4: Evaluation Metrics with and without spellcheck

5 References

1. Step by Step Guide to Master NLP – Topic Modelling using LSA

2. Latent Semantic Analysis — Deduce the hidden topic from the document
3. Dimensionality Reduction by Random Projection and Latent Semantic Indexing
4. pyspellchecker