Project Report

CS6370: Natural Language Processing

Submitted by:
R Rishaab Karthik (MM19B046),
B Aditi (MM19B022),

Contents

Handling the demerits of Vector Space Model Information Retrieval System

Vector space models have been one of the most efficient and accurate models in the realm of Natural Language processing, especially among applications such as Information Retrieval, etc. However, like any ptherr model in existence, they have their own limitations, such as assumption of word orthogonality. In this project, we have tried to improve the standard Inverse Document Frequency (TF-IDF) model by implementing Latent Semantic Analysis in combination with Word2Vec embedding model and comparing the performances of the respective IR systems. By this method, we take into account the relationship between words while implementing the IR system.

1 Introduction

This project uses different Information Retrieval system based on different models on Cranfield Dataset. This section illuminates some basic concepts pertaining to the dataset and models.

1.1 Cranfield Dataset

The cranfield dataset was created on 1967, by C Cleverdon at the college of Aeronautics, Cranfield. This dataset was initially created to test out document indexing techniques. This dataset consists of 1400 documents containing scientific abstracts extracted from aerospace related papers. It also consists of 225 Queries, the relevant ranked document list, and the importance of each relevant document(qrels). [1]

1.2 Cosine Similarity

In order to calculate the similarity between documents or words in the VSM model, we use the vector representations of the documents or the words and find the cosine similarity between them. Qualitatively, it can be viewed as finding the angle between the words or documents, where the lesser the angle the more related they are. In an IR system, documents are retrieved in ascending order of cosine similarity.

$$sim(u, v) = \frac{u.v}{|u| \times |v|}$$

1.3 Inverse Document Frequency

Inverse Document Frequency or TF-IDF is a first order vector space model that uses term frequency in order to create a term-document matrix. The entries in the matrix are created by multiplying 2 numbers, the term frequency (**tf**) and the inverse document frequency ($log(\frac{N}{n})$ where N is the total number of documents and n is the number of documents in which the particular term has occured. As stated earlier, the drawback of this model is the assumption that all words are orthogonal.

1.4 Latent Semantic Analysis(LSA)

Latent Semantic Analysis is a technique that is implemented on the TF-IDF matrix to handle the word Orthogonality problem. In this method, the term-document matrix created by tf-idf undergoes svd decomposition in the following manner:

$$A = U\Sigma V^T$$

where A is the $m \times n$ term-document matrix, Σ is a diagonal matrix with k singular values of matrix A. the shape of Σ is $k \times k$. Using this formula, we get the k^{th} approximation of the matrix A which takes into account the relationship between words. In this the term and document vectors are as follows:

$$t = U\Sigma$$

$$d = V\Sigma$$

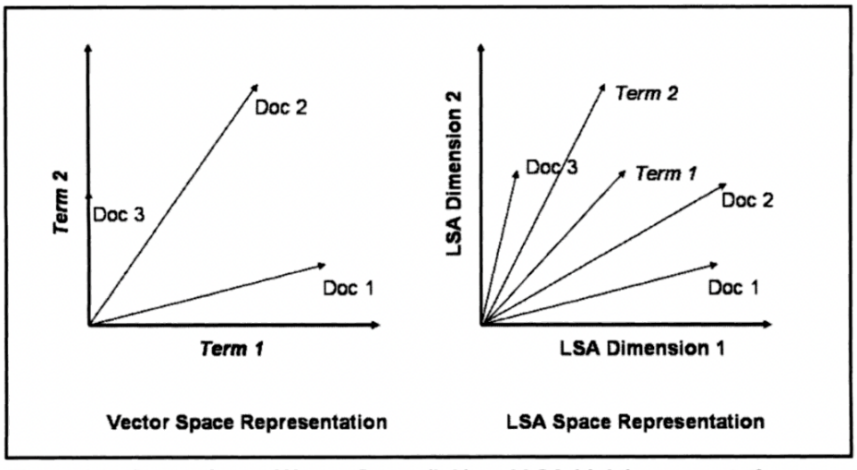

Figure 4.1 Comparison of Vector Space (left) and LSA (right) representations

Figure 1: Red Latent Semantic Analysis, Microsoft Research, Redmond Washington

1.5 Word2vec Embedding Model

Word2vec is a shallow neural network model that uses 1 hidden layer. It takes in words as inputs and returns a vector obtained from prediction based on a global word corpus. This way, we are also taking into account information outside the given dataset. The closeness of the vectors generated dictates the similarity between words. These vectors are in turn used to generate document and query matrices. [2]

2 Data Preprocessing

2.1 Sentence Segmentation

Here we divide the sentences into meaningful units which could be used further for other pre-processing tasks

2.2 Tokenization

We further split the sentences into words as tokens for processing them

2.3 Stop word Removal

We use the punkt tokenizer to get rid of commonly occuring stop words. Different articles, prepositions, conjunctions and other similar words are removed. Here we remove all the punctuation marks as well. (', ", ;, :, /, ?, [,], —, etc).

2.4 Inflection Reduction

Both stemming and lemmatization are useful in this setting. Stemming is faster and easier to perform compared to lemmatization, hence initially stemming was used by search engines. However it can return mistaken words as caring gets cut to "car", thus'over-stemming' can result in mistaken results.Stemming has a high recall and so does lemmatization. Lemmatization however uses morphological analysis and also achieves a higher precision. It is compu- tationally intensive,but, with the increased computational power, it is not an issue to modern search engines.Hence lemmatization is 'better' than stemming for search engine applications for better results. Overall the findings suggest that language modeling techniques improve document retrieval, with lemmatization techniques producing the best result. Thus we use lemmatization in our data preprocessing.

3 Models

In this project, we incorporated 4 different models (refer figure 2)

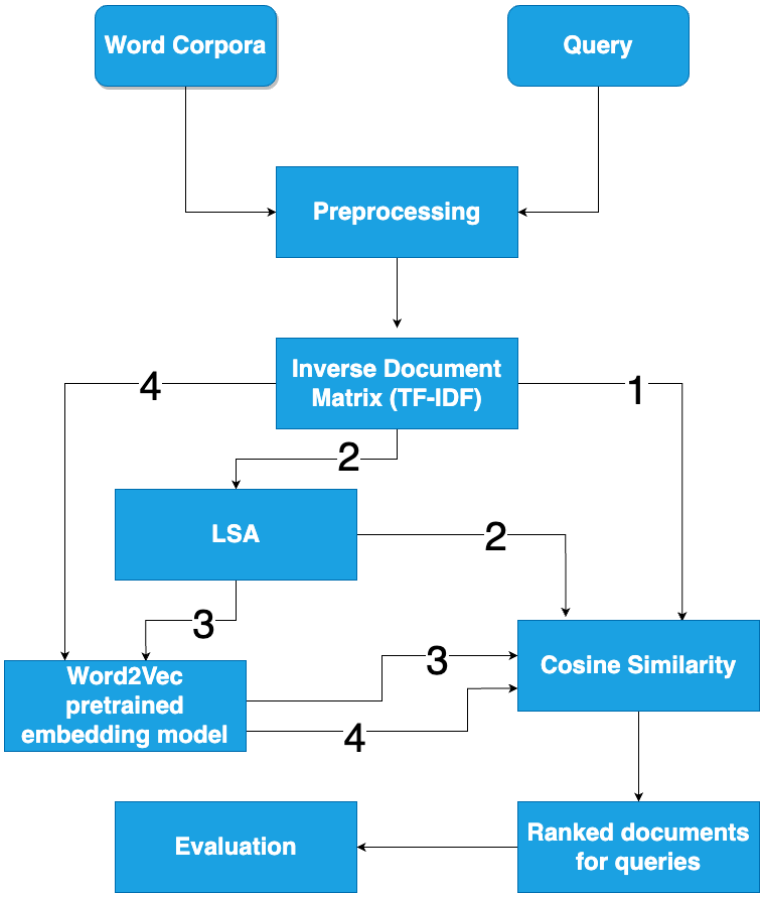

Figure 2: Models implemented

- **Model 1:** This is the base model where the corpus after preprocessing is subject to TF-IDF method. The document vectors generated from this method are then retrieved for the query based on cosine similarity and are evaluated using methods explained in the next section.
- **Model 2:** This model implements LSA on TF-IDF matrix and then uses cosine similarity on the modified document and query vectors for retrieval of documents.
- **Model 3:** In this model, word2vec embedding is implemented on top of LSA. this is done by first retrieving the word vectors correspondingly from the pretrained model (trained on wiki-gigaword corpus). The document vectors extracted from this method are then processed further for retrieval.
- **Model 4:** In this model, we directly implement word2vec embeddings on the tfidf

matrix by using weighted average method.

4 Evaluation

In this project, we use extrinsic evaluation techniques

- **Precision @ K:** Precision is the percentage of documents returned that are relevant to the search. Only the top results generated by the algorithm are taken into consideration when evaluating Precision @ K at a specific cut-off rank.
- **Recall @ K:** The percentage of relevant documents that are successfully recovered is known as recall. Recall @ k is assessed at a predetermined cut-off rank, much like accuracy.
- **Mean Average Precision (MAP):** Across all recall levels, Mean Average Precision (MAP), which gives a single-figure indicator of quality. It has been shown that MAP has among assessment metrics particularly strong discrimination and stability. Average Precision for a particular information need is the average of the precision value acquired for the set of top k documents that remain after each relevant document is retrieved, and this value is then averaged over information requirements.
- **nDCG:** normalized Discounted Cumulative Gain is a measure of ranking quality that takes into account the position of relevant items in a ranked list. The nDCG score ranges from 0 to 1, with 1 indicating perfect ranking and 0 indicating the worst possible ranking. To compute nDCG, we first compute the DCG score for the ranked list of items, and then normalize it by the ideal DCG score, which is the DCG score for a perfectly ranked list of items. The DCG score is computed as follows:

  $$\mathrm{DCG}_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

  The ideal DCG score is computed by sorting the items by their relevance scores in descending order and then computing the DCG score for this sorted list.Once we have computed the DCG score and the ideal DCG score, we can compute the nDCG score as follows:

  $$\mathrm{nDCG}_p = \frac{\mathrm{DCG}_p}{\mathrm{IDCG}_p}$$

- F-Score: It combines precision and recall into a single value, which solves the problems associated with utilising either one alone. It is the harmonic mean of them.

5 Results

The evaluation metrics of different models were plotted and shown in the figures below

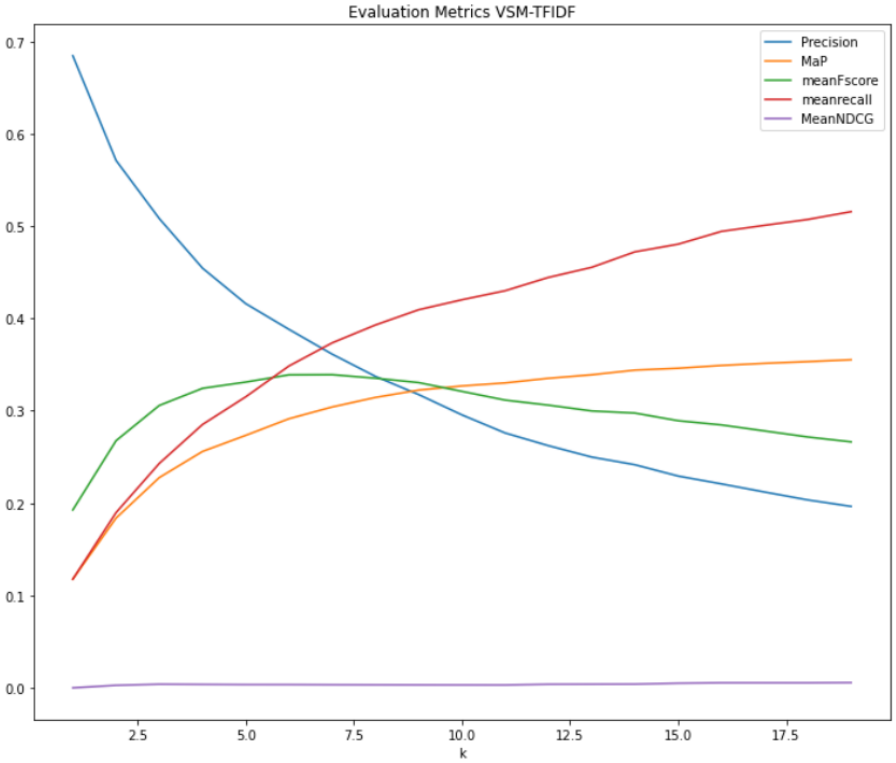

Figure 3: TFIDF Model

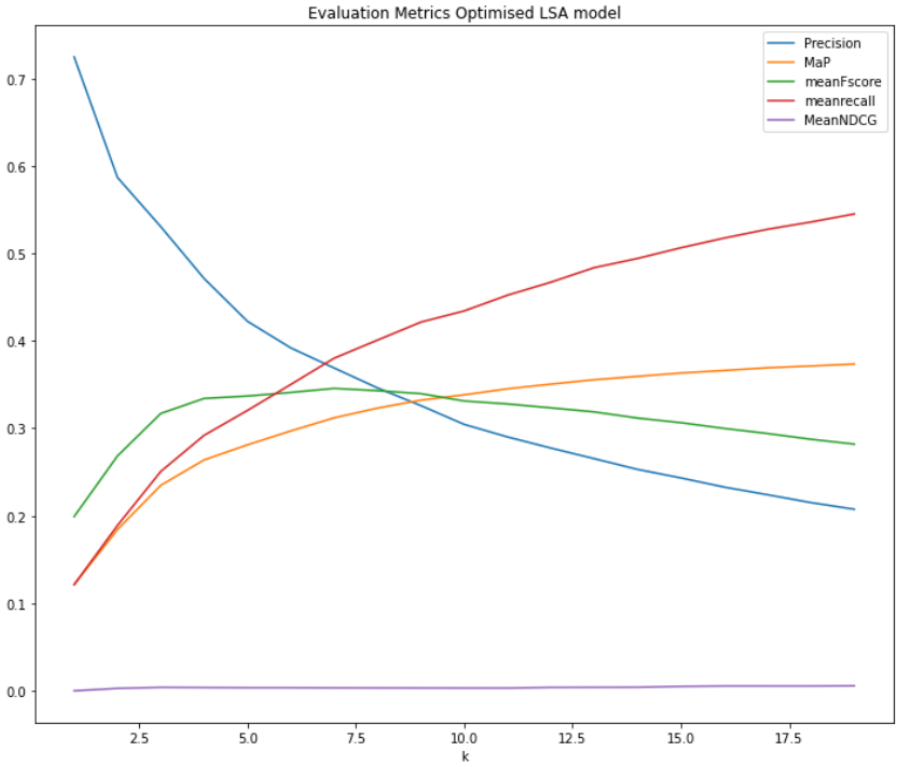

Figure 4: LSA Model

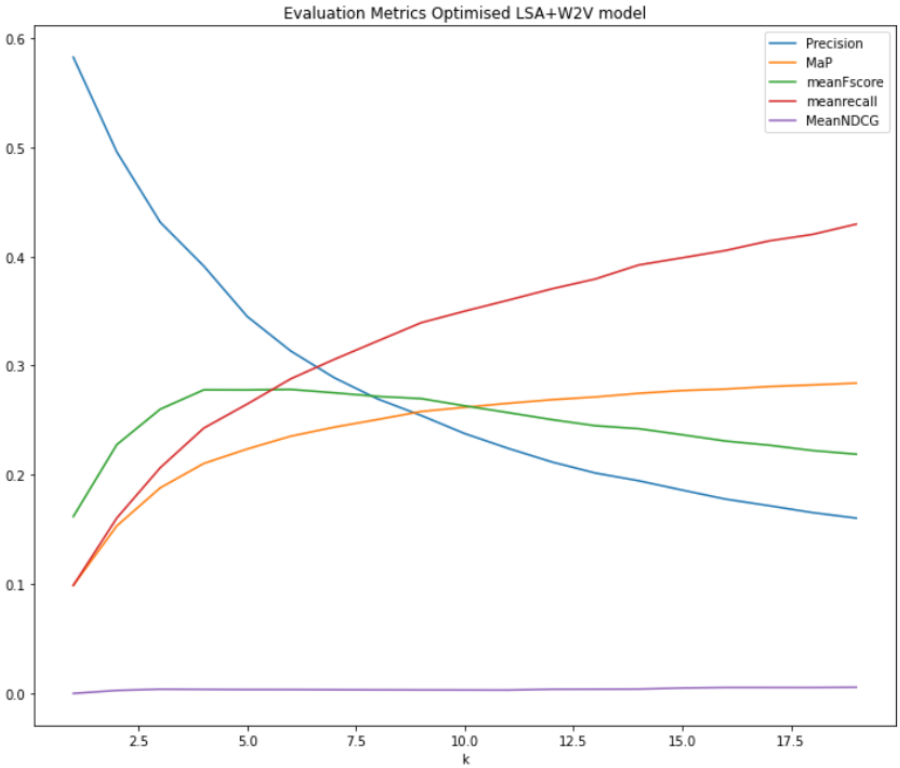

Figure 5: LSA+Word2Vec Model

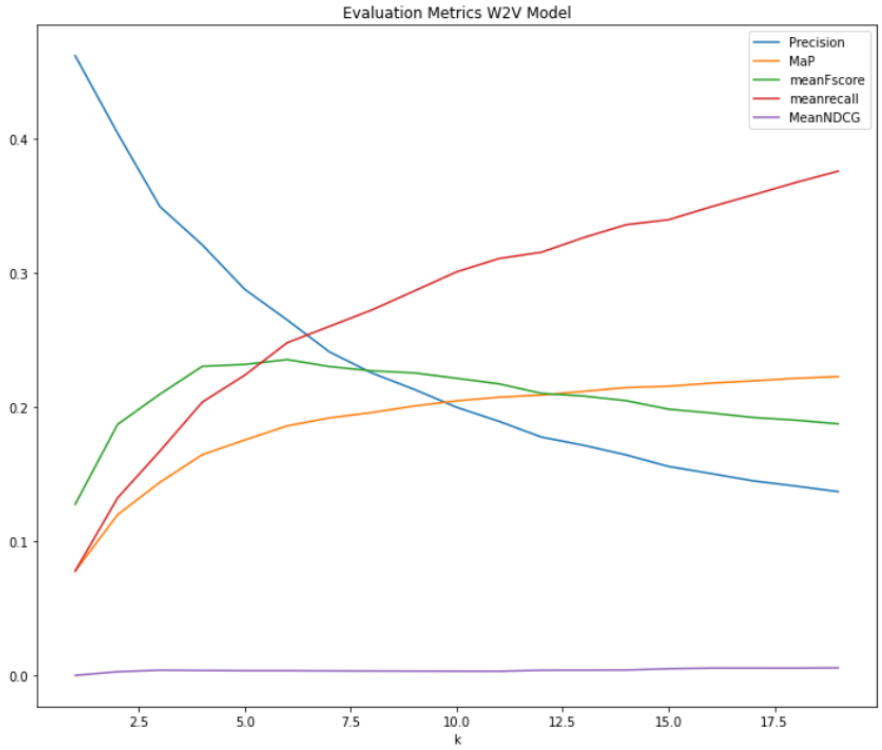

Figure 6: W2V Model

The table below shows different metric scores at k =10 for each of our models We

Model	Precision	MaP	meanFscore	meanrecall	MeanNDCG
TFIDF	0.295556	0.326918	0.321014	0.420230	0.003142
LSA	0.304444	0.338259	0.331191	0.434138	0.003142
LSA+W2V	0.237778	0.261831	0.263169	0.349854	0.003142
W2V	0.200000	0.204791	0.221611	0.301034	0.003142

Table 1: Evaluation Measures at k=10 for all models

can observe from the above table that our LSA model performs well.

6 Conclusion

We conclude that while LSA handled word relation problem present in the TFIDF method, the embedding model failed to do so. Thus, our hypothesis was proved to be false. This maybe the case due to 2 reasons:

1. The word2vec model reduced the dimensionality of the document vector significantly.
2. The pretrained model was learning to capture ord relations in a corpus that contains language dissimilar to the Cranfield dataset.

References

1. CLEVERDON, C. (1967). The CRANFIELD TESTS ON INDEX LANGUAGE DEVICES. Aslib Proceedings, 19(6), 173–194. doi:10.1108/eb050097
2. Google Code Archive - Long-term storage for Google Code Project Hosting. (n.d.). https://code.google.com/archive/p/word2vec/