Improving Basic Vector Space Model

Nithin Uppalapati$^{1[EE18B035]}$ and D Krithin$^{2[EE18B047]}$

Indian Institute of Technology Madras

1 Introduction

A search engine is basically a piece of software that searches a database systematically for particular information specified in a query. Well known examples include Google, Bing, Yandex etc. In this project we attempt to build a smaller scale toy search engine which works on the Cranfield dataset and retrieves documents present in the dataset relevant to the given query. We write the code in python and take advantage of it's vast set of inbuilt libraries like numpy, scipy and nltk to accomplish the above mentioned task.

2 Problem Definition :

We have already implemented a basic search engine using a simple vector model as a part of the assignment. We have evaluated the model's performance using evaluation measures such as, MAP (Mean Avg. Precision), F-Score, nDCG. We understood the drawbacks of our model in the part 2 of our assignment. The main drawback being unable to identify synonymy between words. We will try to overcome these drawbacks in this project. Our aim is to improve the perfomance of the model using techniques taught in the course such as LSA, Query expansion etc.

3 Motivation :

The main motivation behind this project and this course is to understand how a search engine works, what are the ways to improve it and how to analyse the retrieved results asa close to human judgement. We also get to work with various techniques and understand their drawbacks. We will try combining two different methods and see whether it increases/decreases the performance. We also try to get an intuitive understanding of the different techniques used in the project and how they effect the search engine's performance.

4 Background and related work :

4.1 Pre-Processing Work (Assignment-1, P1):

• **Concatenation**: Title, Description and Keywords (when available) of contents are concatenated in a single text line.
• **Tokenization**: Text data is separated into tokens using the word tokenizer from nltk (Python library).
• **Lower case**: Every token is converted to lower case, in order to recognize similar tokens like "Smith" and "smith" as only one.
• **Punctuation marks removal**: punctuation marks, such as ".", ",", ":", "!", etc., are removed from the text.
• **Stop words removal**: Words that are excessively frequent are removed from text, because it is known that they do not have significant information.
• **Lemmatization**: Tokens are converted to its lemma using the wordnet lemmatizer from nltk (Python library).

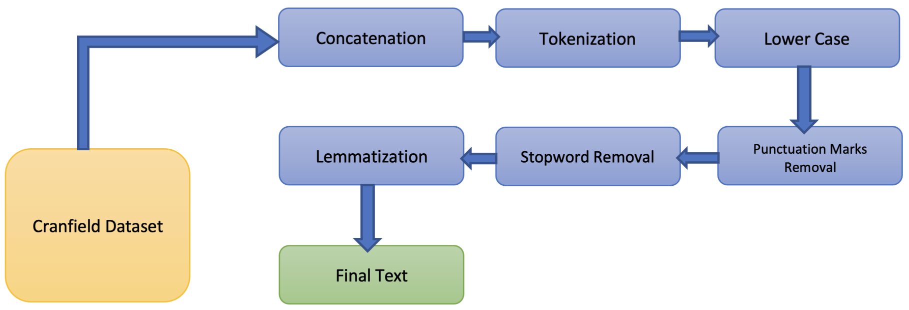

Fig. 1. A flow chart showing the steps involved in the process of pre-processing the documents.

4.2 Creating a Vector space Model (Assignment-1, P2):

In this part we implemented a simple vector space model for information retrieval. We started of preprocessing all the documents in the corpus and built an index from it. This was done by creating a TF-IDF matrix. Then we took the preprocessed queries and then ordered the documents based on their relevance to the given queries. We also evaluated this model by comparing the order of the documents to the ideal order of documents. This is how we implemented the vector space model(We referred to this model as the baseline model throughout the report).

5 Proposed Methodology :

5.1 LSA :

Latent Semantic Analysis (LSA) is a distributional semantic technique, which is an extension of TF-IDF vector space mode. It can analyse linguistic properties such as synonymy and polysemy of words.

First we have an mxn matrix (words vs docs) containing the TF-IDF values. We prefer using TF-IDF values instead of count occurrence because high frequency does not account for better classification. The ideology behind TF-IDF is that high frequency words do not provide much information in differentiating documents whereas rare words contribute much more weight.

Then we perform truncated singular value decomposition (SVD), which is applied to the TF-IDF matrix. LSA returns a term-document matrix where similar documents and similar words are placed closer. The specific number of columns in the output matrix is equivalent to the document topics. Then we truncate the first k concepts, the reason behind doing this is that the matrix is very sparse and noisy or includes a lot of low frequency words.

Similar to factor analysis, principal components analysis, the main purpose of LSA is dimension reduction. And also it is a type of spectral learning (unsupervised), and thus it brings all the documents in ”k” concepts (clusters).

5.2 Query Expansion :

Query expansion is a method in which the query is reformed to improve retrieval performance. The query can be reformulated in many ways, for example we can construct a query expansion matrix using the similarities of words and multiply it with query vector to obtain the reformulated query. Another way is to take the queries before stemming, after stopword removal and then add synonyms of words present in the query to the list. We tried both methods and found the second one to be faster.

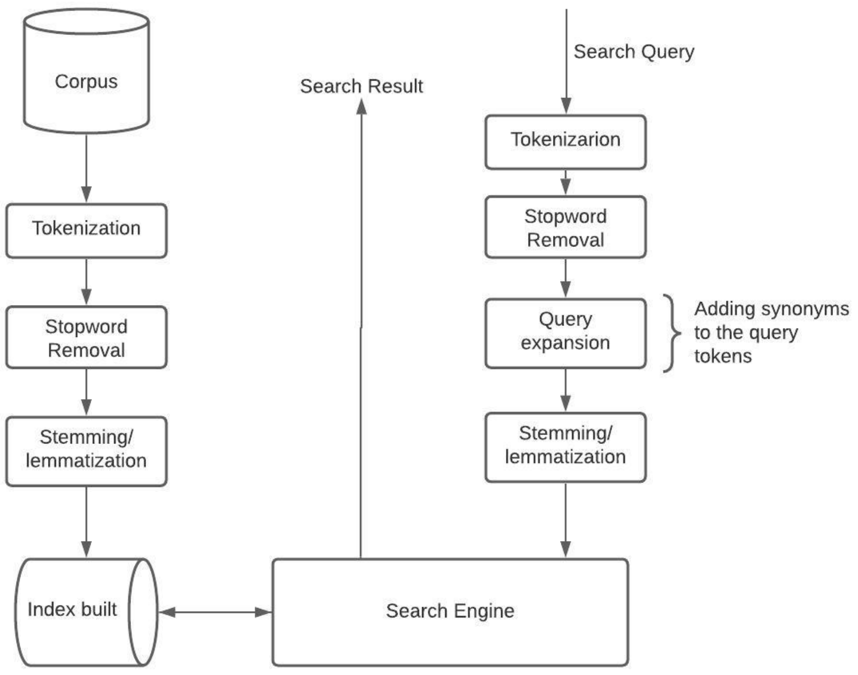

Fig. 2. Flowchart of query expansion *.isalpha == True*

6 Experiment :

6.1 Experiment - A: Finding optimal k (no. of concepts) for LSA:

Initially we wanted to find the optimal number of concepts (k) by using the coherence measure which assesses the quality of the concepts. We calculate the concept coherence value for each topic, find the mean for K topics and find k for which the mean coherence score is highest, but we weren't able to implement this. Hence we decided to run our code with different values of k and compare the eval_metrics and PvsR graphs and obtain the best value of k. This was possible because an instance of our code took just under 2 mins to run. See [Fig.3] for the evaluation of the metrics mentioned above. And also see [Fig.4] for Precision vs Recall plot for the baseline model and LSA ($k = 301$) model.

After doing a discrete search, we ended up with optimal value of k as 301. And we checked that nDCG is peaking at this value of k.

6.2 Experiment - B: Removing numerical values from collection of words

In this setup, we tried to only retain the "alpha" characters. In other words, we retained only those words which satisfy the below rule:

```
    <word>.isalpha() == True  ## Pyhton's inbuilt function for
                                 strings
```

and observe the effect on precision vs recall plot with out baseline model. The reason for this doing and testing this experiment is that, as there were various symbols embedded in the alpha-numeric characters, so the chances of that term being in a query are lesser, as lot of people represent mathematical equations in lots of ways. And another source of numeric characters in the documents is due to Bibliography, hence that was the motivation behind performing this experiment.

But as we anticipated, there was a trade-off, Recall got increased and Precision got decreased but small amounts. The reason being, due to removal of those alpha-numeric / numeric characters, there was a decrease in the granularity in the vector space model. See [Fig.5] for for the results when compared to baseline (Precision vs Recall Curve).

6.3 Experiment - C: Testing Okapi BM25 model

Okapi BM25 (BM is an abbreviation of best matching) is a ranking function used by search engines to estimate the relevance of documents to a given search query based on the probabilistic retrieval framework.

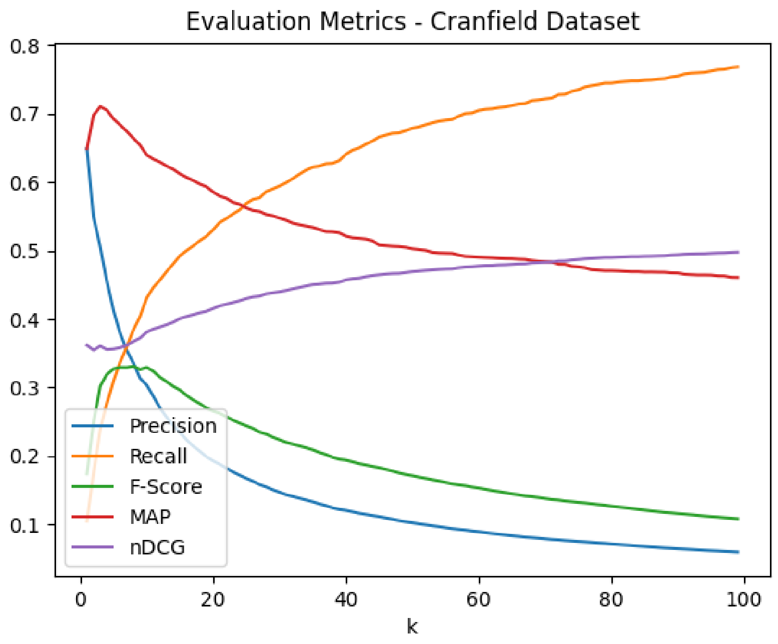

Fig. 3. Evaluation metrics evaluated on LSA (k=301) model, plotted for first 100 retrieved documents

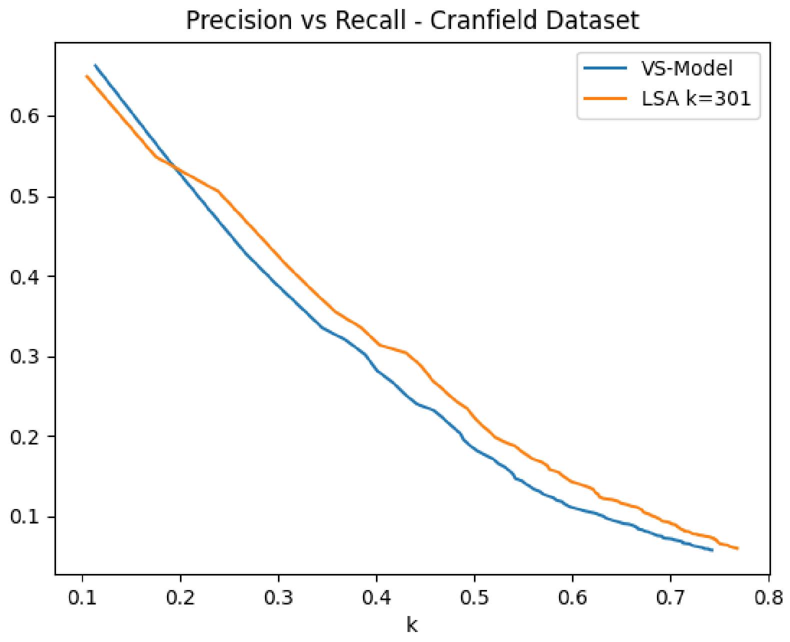

Fig. 4. Precision vs Recall of LSA (k=301) when compared with Baseline model, plotted for first 100 retrieved documents

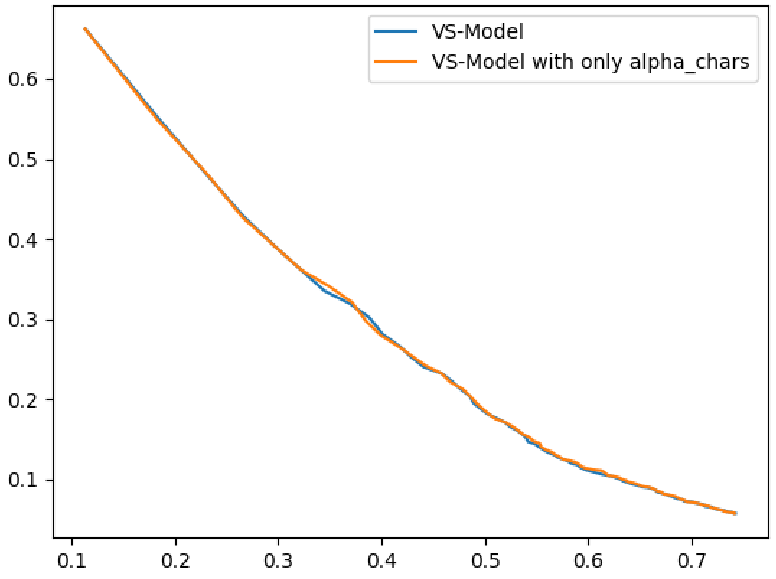

Fig. 5. Precision vs Recall when compared with Baseline model, plotted for first 100 retrieved documents, while only considering *.isalpha == True*

BM25 is a bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document, regardless of their proximity within the document. It is a family of scoring functions with slightly different components and parameters. One of the most prominent instantiations of the function is as follows.
Given a query Q, containing keywords $q_1, ..., q_n$ the BM25 score of a document D is:

$$score(D,Q) = \sum_{i=1}^{n} IDF\left(q_i\right) \cdot \frac{f\left(q_i, D\right) \cdot \left(k_1 + 1\right)}{f\left(q_i, D\right) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{avgdl}\right)} \tag{1}$$

Where $f(q_i, D)$ is q_i's term frequency in the document D, $|D|$ is the length of the document D in words, and avgdl is the average document length in the text collection from which documents are drawn. we implemented a naive BM25 model to compare with our approach. Precision vs Recall graph was compared to the baseline model in [fig.6].
This is how we trained the BM25 model, with cranfield dataset:

```
docs_df = pd.read_json('cranfield/cran_docs.json')
docs_df = docs_df.drop(["bibliography", "author"], axis=1)
nlp = spacy.load("en_core_web_sm")
bm25 = BM25Okapi(tok_text)
results = bm25.get_top_n(tokenized_query, df.text.values, n=k)
## where k is the no. of documents to be retrieved.
```

1. First we loaded the documents in pandas dataframe.
2. Then we removed unwanted data such as Bibliography, and author names.
3. Then we are loading "en_core_web_sm" which is a pipeline optimised for CPU on performing operations for English language tasks.
4. Then we train the BM25 model and then pass a query (string format) to get the list of retrieved documents, in the "results" variable.

6.4 Experiment - D: Query expansion

We implemented query expansion by reformulating the query after stopword removal and before stemming/lemmatization. We appended synonyms of each word of the query to the existing query list, so we ended up with a list of lists with each sublist containing tokens of a query along with their synonyms. We experimented by first adding just one synonym of each word and then we added two synonyms for each word(this can be done in the code by modifying the inequality of count). Then we performed stemming/lemmatization as usual. Then we plotted the P vs R plots of both the above cases with and without combining with LSA. The plots are shown in [Fig.7], [Fig.8], [Fig.9].

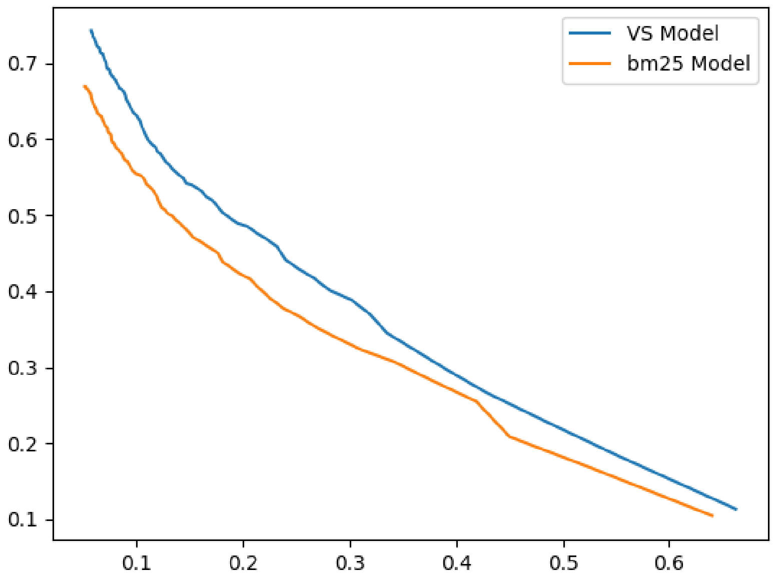

Fig. 6. Precision vs Recall of BM25 model when compared with Baseline model

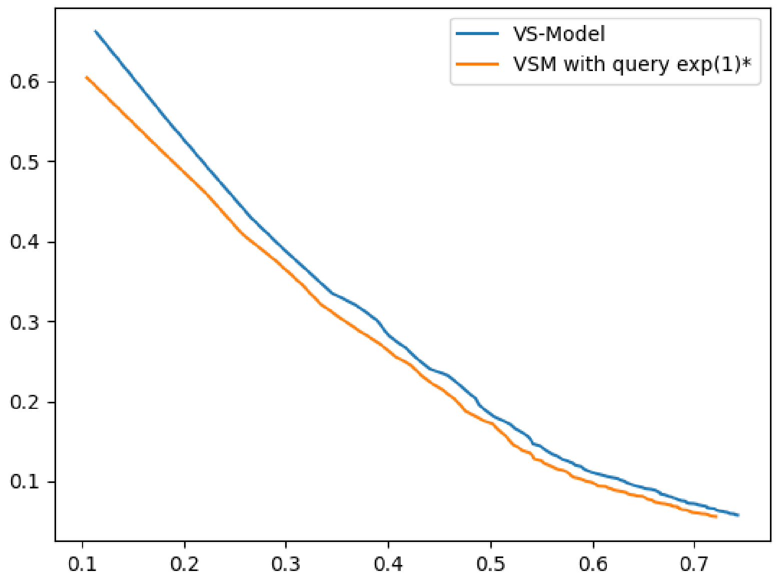

Fig. 7. VS-Model when operated on expanded query space (1 synonym) when compared to Baseline Model

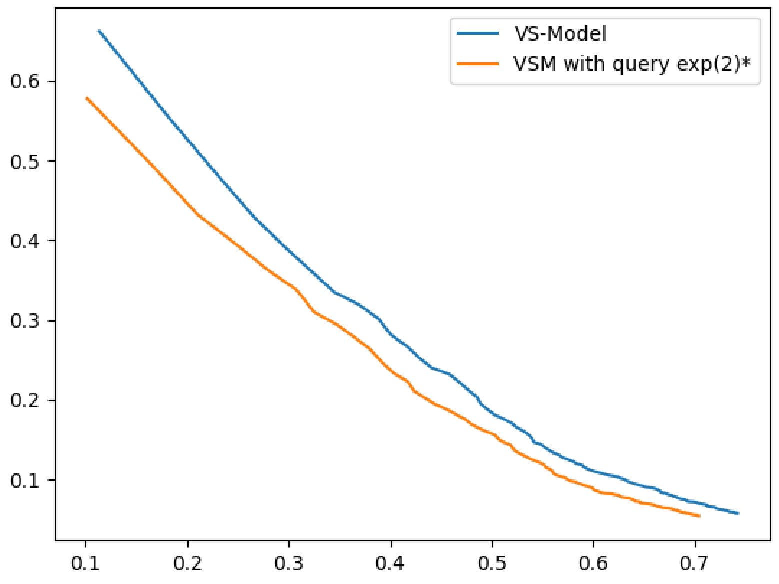

Fig. 8. VS-Model when operated on expanded query space (2 synonyms) when compared to Baseline Model

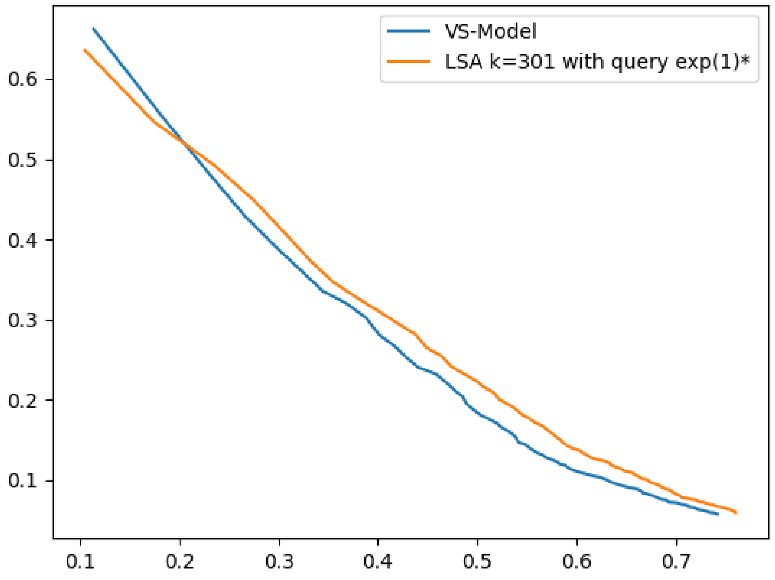

Fig. 9. LSA (k-301) model when operated on expanded query space when compared to Baseline Model

7 Results :

- When we applied query expansion on our baseline model, (with 1 extra synonym and with 2 extra synonyms) the area under the P vs R curve got decreased. Refer [Fig.7] and [Fig.8]
- When we applied query expansion on LSA model (k=301), (with 1 extra synonym and with 2 extra synonyms) the area under the P vs R curve got increased when compared to just applying query expansion on the baseline model, but it is still lesser than that of just applying LSA on the baseline model. Refer [Fig.9]
- When we applied only LSA (k=301) on the baseline model, this is best model we found so far. You can refer the results at [Fig.3] and [Fig.4]
  Map Peak occurred at 3rd retrieved document.
  F-Score peak occurred at 5th retrieved document.

8 Conclusion :

In our overall implementation, LSA (k=301) is the best performing model

We experienced a decreased in results when we implemented query expansion using wordnet synonyms. Part of the reason is that wordnet's 1st sense of synset is more general in nature, while we are operating on cranfield dataset, which is more technical in nature.

We also observed that polysemy tends to decrease precision and synonymy tends to decrease recall.

By the end of this project we have tried out various techniques, studied their advantages and disadvantages and got a better understanding of where certain methods can be used to improve the search engine's performance.

References

1. GangKou,YiPeng:AnApplicationofLatentSemanticAnalysisforTextCategorization(2015) .InternationalJournalofComputers,Communications&Control.DOI:10.15837/ ijccc.2015.3.1923
2. https://www.researchgate.net/publication/276127714_An_Application_of_ Latent_Semantic_Analysis_for_Text_Categorization
3. https://queryunderstanding.com/query-expansion-2d68d47cf9c8
4. https://en.wikipedia.org/wiki/Okapi_BM25