NLP Project Report

CS20B051, CS20B071

May 2023

1 Introduction

The main goal of NLP is to enable computers to understand and process human language, and Information Retrieval plays a crucial role in achieving this goal by providing algorithms and techniques for accessing and analyzing large collections of textual data. Having made a simple IR system using the Vector Space Model, we stumbled upon a few shortcomings in the model, like not being able to deal with polysemy, synonymy, spelling errors, etc. Hence, we provide some hypotheses with the aim of improving this model. All the hypothesis testing is done on the Cranfield dataset.

2 Considering context

Currently, all the words are treated independently without looking at the context and the order in which they appear. This was leading to inaccuracies because sometimes the meaning of a word depends on the context and there might be a case where although the query and document have similar words, upon checking the context, we come to know that they are not related at all. For example, if the query is asked about "rectangular wings" and we find a document containing "elliptical wings", they might be matched due to the presence of wings. However, a document about elliptical wings would give us no information about our query.

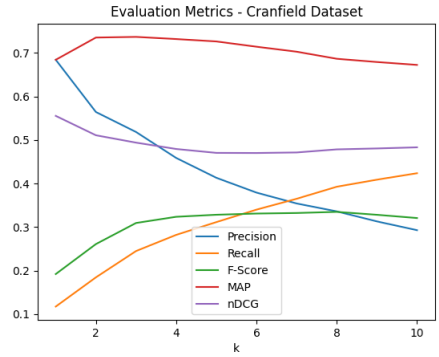
Hypothesis 1: Using bigrams instead of unigrams would give better results as it helps in considering more context.

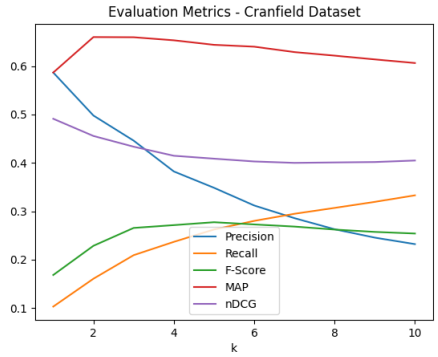
Hypothesis 2: Using a hybrid of bigrams and unigrams instead of just unigrams would give better results as it helps in considering more context.

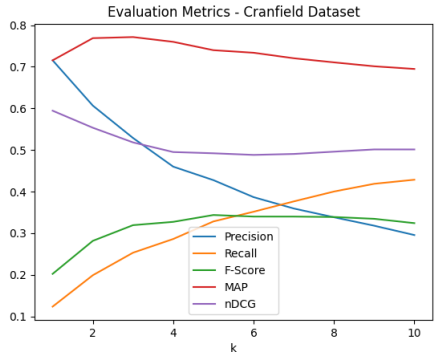
Procedure: We used the CountVectorizer and TfidfTransformer modules scikit-learn package. The n-gram range was varied.

Testing: Using bigrams instead of unigrams would definitely help in considering more context. However, we can see that the performance decreases if we use just bigrams. However, a hybrid model of unigram and bigram(with ngram

range as (1-2)) gives a significant improvement.
Results for unigrams, Results for bigrams, Result for a hybrid model







Conclusion: Hypothesis 1 is incorrect while Hypothesis 2 is correct.

3 Considering Semantics

Semantic relatedness was not considered at all in Vector Space Model because words were directly compared. Hence, two words with the exact same meanings could have been assigned orthogonal vectors. To resolve this, we use LSA(Latent Semantic Analysis) and ESA(Explicit Semantic Analysis).

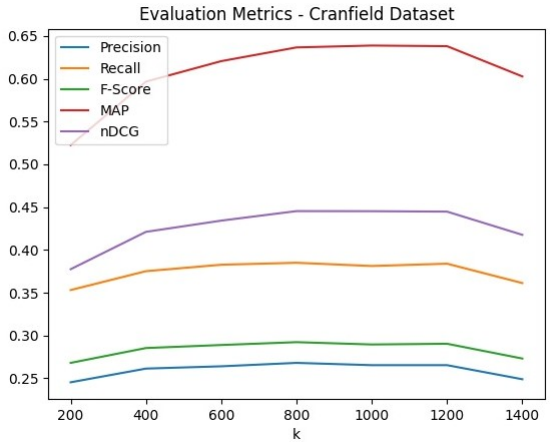
Hypothesis: Incorporating word semantics using LSA/ESA would improve the IR performance.

Procedure:
For implementing LSA, we used the SVD decomposition and find the k-rank approximation of the document-term matrix. The value of k has to be tuned to get the maximum precision. A higher value of k would lead to more precision but less recall. A lower value would lead to more approximation and thus more recall, albeit compromising on the precision.

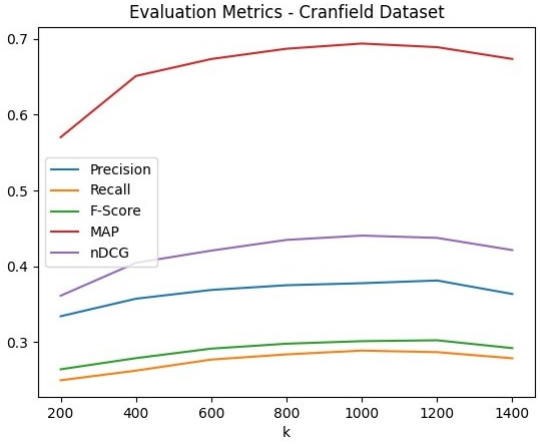
For implementing ESA,We utilized the Spacy library to preprocess text and

generate word vectors. Subsequently, we generate document vectors by averaging the vectors of individual sentences and further averaging those sentence vectors. Our ranking function takes queries as input, generates query vectors using the same process, and ranks documents based on cosine similarity scores between the query and document vectors.

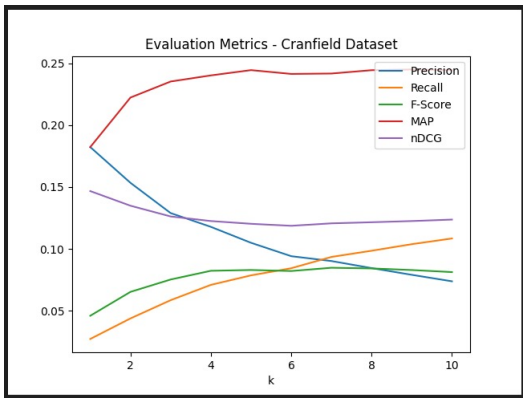
Testing: For LSA, we ran the model for several values of k to see the variations in the evaluation metrics.
Evaluation metrics @10



Evaluation metrics @5



Conclusion: It can be seen here that the values peak at around 1000-1200. Hence, we choose a dimensionality reduction to 1100 for our model. Thus, our hypothesis that LSA will improve the model performance turned out to be correct.
For ESA, the model was extremely slow and ran for almost 15 minutes. The following results were obtained:



Conclusion: ESA performs the worst of all the models we have tried so far. A possible explanation could be attributed to the fact that the Cranfield dataset is just a small subset of all the real-world knowledge and information. However, using ESA, we assign to each word a vector based on real-world information. Thus, the chances of the vectors being similar reduces a lot.

4 Spell-check

Hypothesis: Using a spell-checker shall give better results for queries with spelling errors.
A thing to be noted here is that this won't improve performance on the Cranfield dataset because all the queries are inherently spelled correctly. However, if you're providing custom queries to the model, this would help in resolving the spelling errors.

Procedure: We imported the module enchant from Python, which gives suggestions to correct words.

Testing:
Query 1: "Papers on Arodynamics"

```
Enter query below
Papers on Aerodynamics

Top five document IDs :
925
1379
875
137
1066
```

```
Enter query below
Papers on Arodynamics

Top five document IDs :
875
925
895
1063
983
```

```
Enter query below
Papers on Arodynamics

Top five document IDs :
925
1379
875
137
1066
```

From left to right: Expected output, Output without spell-checker, Output with spell-checker
Thus, it can be seen that the expected output and output with the spell-checker exactly match. On the other hand, the search engine without a spell-checker isn't giving accurate results. In conclusion, even a minor spelling error in a

query like missing the 'e' in aerodynamics won't matter now.

Conclusion: The above hypothesis is correct.

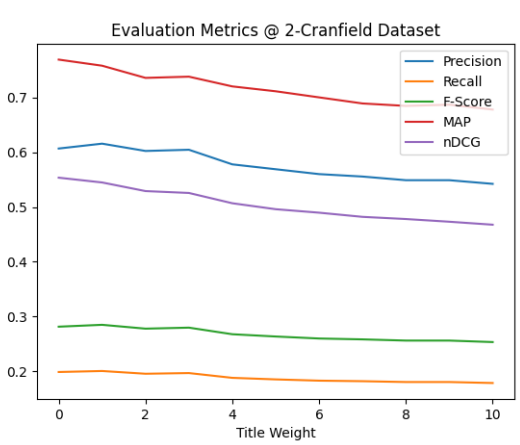
5 Weights to titles

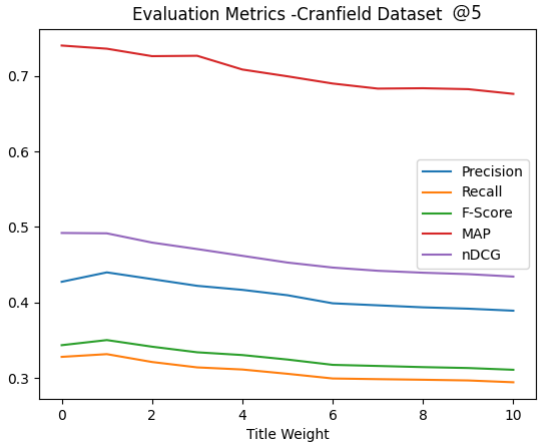
Note that in the Cranfield dataset, each document has a title associated with it. Our previous model didn't take into account their importance. However, they can give more importance to the queries than the text itself because if a query matches the title, it is most probably related to the document. On the other hand, if it is completely unrelated, chances are we won't find a lot of similarity between the query and the document.

Hypothesis: Title of the document isn't taken into account. Considering it would yield better results.

Procedure: The title of the document can be given more weight, say k times that of the text in the document. For this, we can simply add the title sentences to the body k number of times in the main.py file.

Testing: Here are the results we get for different values of k from 1 to 10.





A general trend can be observed. The more we increase the title weight, the worse the performance gets. Although there are places where the title weight = 1 performs better than 0 title with no weight, that is very rare. Thus, we decided to keep the title weight equal to 0 in our final model.

Conclusion: The above hypothesis is incorrect.

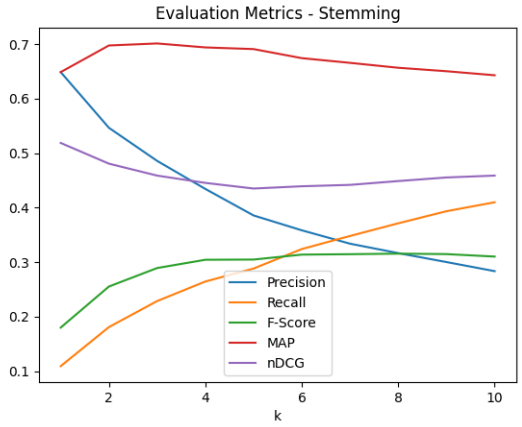
6 Stemming vs lemmatization

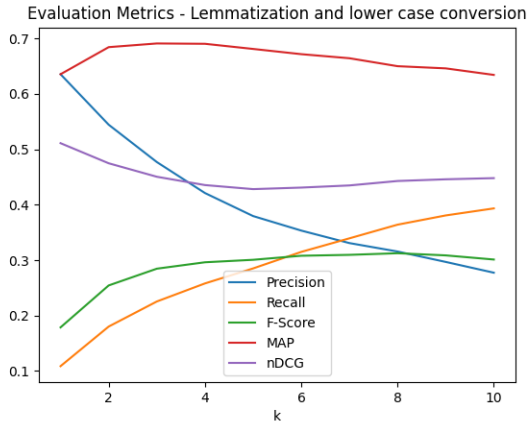
An observation we made in the assignments was that lemmatization wasn't working for a simple query like "Aerodynamics" because it wasn't being re-

duced to "aerodynamic". Hence, we decided to go with stemming for our VSM model. However, lemmatization wasn't working because it wasn't changing the uppercase letters to lowercase. Hence, we try to work on the preprocessing once more.
Hypothesis: While preprocessing, lemmatization with converting to lower-case will give better results than stemming.

Procedure: We find the results for both these preprocessing techniques by using the NLTK package for the WordNetLemmatizer and PorterStemmer.

Testing:



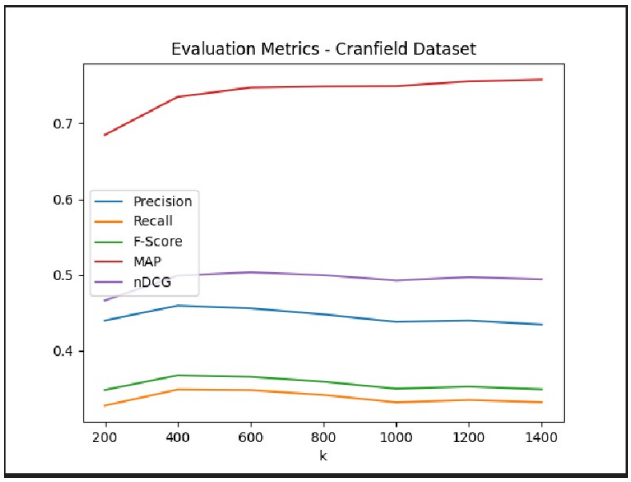


It can be seen that although both of these look very similar, stemming gives slightly better results(Eg. Precision@1 is 0.648 and 0.635, respectively) than lemmatization, even after conversion to lowercase. It is also beneficial for the model because stemming is much faster and more efficient.

Conclusion: The above hypothesis is incorrect.

7 A combination of the above hypotheses

We can combine the hypotheses above in order to optimize our model further. For a model that uses a hybrid of bigram/unigram, LSA and gives a weight of 1 to the title, let's find out what the best value of k can be.

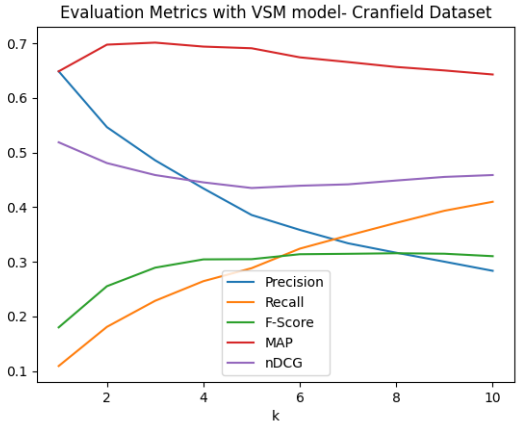


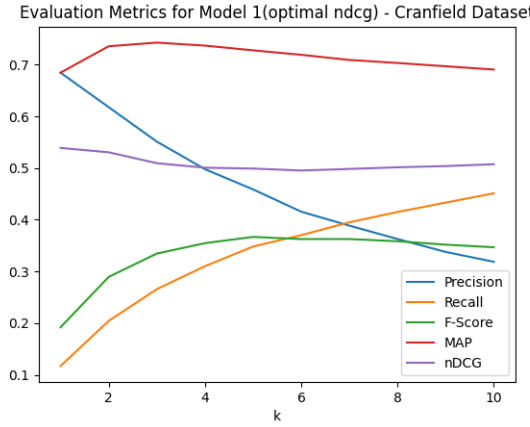
Thus the best results for nDCG are at k = 400, which is the value we use for our final model to optimize nDCG.

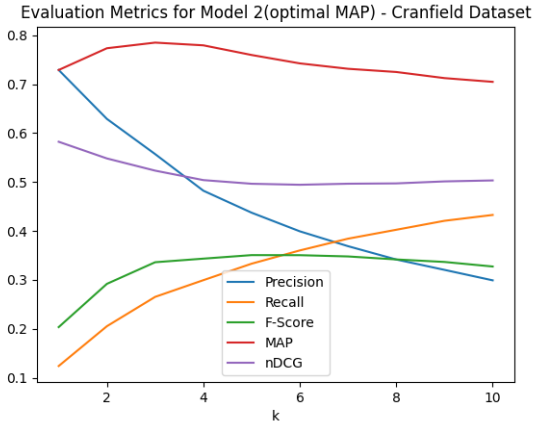
8 Final model: Improvements

Using the above hypotheses and trying out different hybrid models, we decided on the two best models, out of which model 1 optimizes nDCG and model 2 optimizes MAP value.
Model 1: Hybrid unigrams/bigrams + LSA + weight of 1 to title
The initial nDCG@10 is: 0.459
The final nDCG@10 is: 0.510
This shows a percentage improvement of approximately **11%**
The initial MAP@10 is: 0.643
The final MAP@10 is: 0.694
The percentage improvement is approximately **8%**
Model 2: Only Bigrams
The initial nDCG@10 is: 0.459
The final nDCG@10 is: 0.499
This shows a percentage improvement of approximately **8.7%** The initial MAP@10 is: 0.643
The final MAP@10 is: 0.705
The percentage improvement is approximately **9.64%**

A graphical side-by-side comparison of the performance of the three models:







9 References

https://www.geeksforgeeks.org/python-lemmatization-with-nltk/
https://www.geeksforgeeks.org/python-stemming-words-with-nltk/
https://www.geeksforgeeks.org/tf-idf-for-bigrams-trigrams/
https://www.geeksforgeeks.org/python-spelling-checker-using-enchant/
https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
https://spacy.io/models/en