Information retrieval system on Cranfield dataset

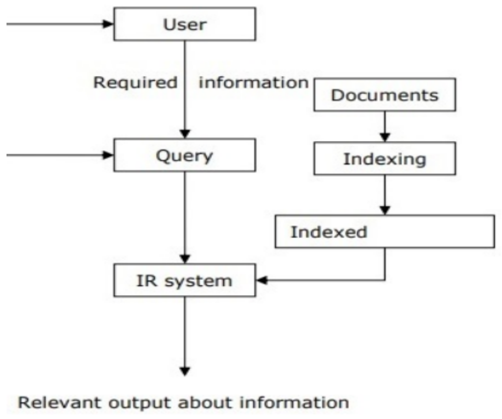
K Santosh [CH17B053], Jain Devansh Rakesh [CH17B050]

Abstract. Information retrieval is one of the most common applications of natural language processing. We present a methodology to implement an information retrieval system which, given a query, returns a set of document ids in decreasing order of relevance. For this purpose, we use various methods to model and gauge similarity between a query and a document. In this work we choose a vector space model (VSM) for representing the query and document and use cosine similarity to find degree of relevance. We perform various preprocessing techniques such as sentence segmentation, tokenization, stop-word removal, spell-check etc before applying the VSM. We run the system on a cranfield data set and get reasonably well results.

Keywords: Information Retrieval, Vector Space Model, Latent Semantic Indexing, Word2Vec

1. Introduction

Information retrieval (IR) may be defined as a software program that deals with the organization, storage, retrieval and evaluation of information from document repositories, particularly textual information. The system assists users in finding the information they require but it does not explicitly return the answers of the questions. It informs the existence and location of documents that might consist of the required information. The documents that satisfy the user's requirement are called relevant documents. A perfect IR system will retrieve only relevant documents.The main objective of Information retrieval is to supply right information, to the hand of the right user at the right time. Various materials and methods are used for retrieving our desired information. the user must enter a query in natural language that describes the required information. Then the IR system will return the required documents related to the desired information.



Mathematically, models are used in many scientific areas having the objective to understand some phenomenon in the real world. A model of information retrieval predicts and explains what a user will find in relevance to the given query. IR model is basically a pattern that defines the above-mentioned aspects of retrieval procedure and consists of the following −

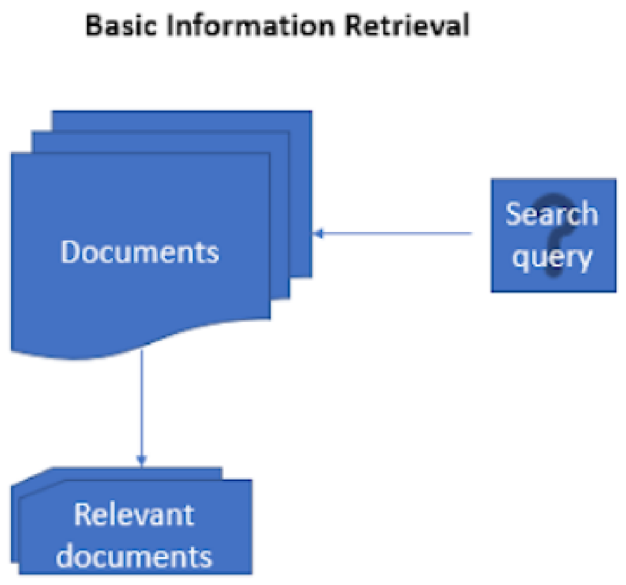
- A model for documents.
- A model for queries.
- A matching function that compares queries to documents.

Mathematically, a retrieval model consists of −

1. D − Representation for documents.
2. Q − Representation for queries.
3. F − The modeling framework for D, Q along with the relationship between them.
4. R (q,di) − A similarity function which orders the documents with respect to the query. It is also called ranking.

2. Problem definition

An information retrieval system needs to be built which can retrieve documents from a set of documents given a query for the document. We have a naive vector space model, which uses the classical approach to implement the IR system. However, it fails to retrieve relevant documents for some queries(**write drawbacks**). We need to find and implement methods which can address these issues and make the model more efficient and accurate.



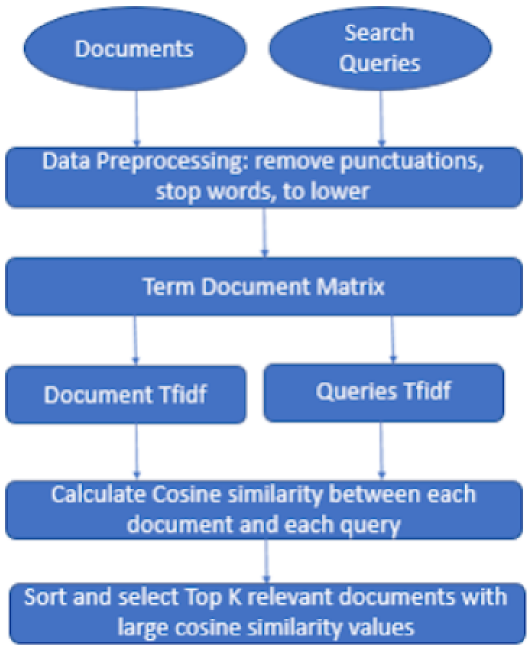
Document retrieval system

3. Background and related work

An information model (IR) model can be classified into the following three models −
Classical IR Model:
It is the simplest and easiest to implement IR model.Boolean, Vector and Probabilistic are the three classical IR models.
Non-Classical IR Model:
It is completely opposite to the classical IR model. Such kinds of IR models are based on principles other than similarity, probability, Boolean operations. Information logic model, situation theory model and interaction models are the examples of non-classical IR models.
Alternative IR Model:
It is the enhancement of classical IR models making use of some specific techniques from some other fields. Cluster model, fuzzy model and latent semantic indexing (LSI) models are examples of alternative IR models.
In classical IR models, the models are based on mathematical knowledge that was easily recognized and understood as well. Boolean, Vector space and Probabilistic are the three classical IR models. The primary data structure of most of the IR systems is in the form of inverted index. We can define an inverted index as a data structure that lists, for every word, all documents that contain it and frequency of the occurrences in the document. It makes it easy to search for 'hits' of a query word. A simple Boolean model is easiest to implement but it does not give a ranked set of retrieved documents and the similarity function is boolean which is unreliable. Gerard Salton and his colleagues suggested a model, which is based on Luhn's similarity criterion. The similarity criterion formulated by Luhn states, "the more two representations agreed in given elements and their distribution, the higher would be the probability of their representing similar information."
The Vector-Space Model (VSM) for Information Retrieval represents documents and queries as vectors of weights. Each weight is a measure of the importance of an index term in a document or a query, respectively. The index term weights are computed on the basis of the frequency of the index terms in the document, the query or the collection. At retrieval time, the documents are ranked by the cosine of the angle between the document vectors and the query vector. For each document and query, the cosine of the angle is calculated as the ratio between the inner product between the document vector and the query vector, and the product of the norm of the document vector by the norm of the query vector. The documents are then returned by the system by decreasing cosine.

4. Information retrieval models

1. Vector Space Model


Documents
Search Queries
Data Preprocessing: remove punctuations, stop words, to lower
Term Document Matrix
Document Tfidf
Queries Tfidf
Calculate Cosine similarity between each document and each query
Sort and select Top K relevant documents with large cosine similarity values

The vector space model is a model which represents the documents in a vector space of terms as dimensions. It is used in information retrieval, filtering and relevance ranking. The values of the document vector are determined by a weighting scheme. One of the best known schemes is the tf-idf weighting. tf refers to the term frequency in a document and idf is given by log(D/df), where D = number of documents and df = number of documents in which the term is present. Similar vectors are formed for the queries too. These vectors are used to find similarities between the query and documents using similarity measures like cosine similarity, euclidean distance, etc.

2. Latent Semantic Indexing

Latent Semantic Indexing (LSI) is a technique used for analyzing the relationships between a set of documents and the terms contained in them by producing a set of concepts related to the documents and terms. A term document matrix is created using the tf-idf weighting of vectors and then using a matrix decomposition method known as Singular value decomposition (SVD), a k-rank approximation of the original matrix is calculated. Here, k can be considered as the number of concepts to generate. LSI uncovers the underlying latent semantic structure in the usage of words in a body of text and how it can be used to extract the meaning of the text in response to user queries, commonly referred to as concept searches. Queries, or concept searches, against a set of documents that have undergone LSI will return results that are conceptually similar in meaning to the search criteria even if the results don't share a specific word or words with the search criteria.

3. Query Expansion using wordnet

Query Expansion is the process of reformulating a given query in order to get an improved document retrieval. A search query is expanded in order to match additional terms in documents. Query is expanded by adding semantically related words (synonyms, antonyms, hyponyms, hypernyms) of the terms already present in the original query. A re-weighting of the terms in the query is then done.

4. Word2Vec

Word2vec is a technique which uses a neural network to learn word associations from a corpus of text. Each word is represented mathematically as a vector of a certain dimension. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located close to one another in the space. Word2Vec is a method to construct such an embedding. It can be obtained using two methods:

Continuous Bag Of Words: This method takes the context of each word as the input and tries to predict the word corresponding to the context. One hot encoding of the input word is used to measure the output error compared to one hot encoding of the target word. In the process of predicting the target word, the vector representation of the target word is learnt.

Skip Gram: In this method the target word is used to predict the context words. In other words, target word is provided as input into the network and the model outputs C probability distributions. For each context position, C probability distributions of V probabilities, one for each word, are obtained.

5. Proposed methodology

We make changes in the naive implementation of VSM to improve the overall effectiveness of the model. We use evaluation metrics such as precision, recall, nDCG etc., to get an idea of how well the IR system is performing with these changes. Finally we use hypothesis testing to arrive at a conclusion on which model (including method of preprocessing) is performing the best.

1. Data Preprocessing

The documents and queries are preprocessed using the following steps:

- **Segmentation**

The sentences are segmented using the Punkt Sentence Tokenizer from the natural language toolkit. This tokenizer divides a text into a list of sentences by using an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences.

- **Tokenization**

The segmented sentences are then tokenized into words using a suitable tokenizer. In the VSM trained for cranfield dataset in the assignment we used the Penn TreeBank tokenizer. We observed that the treebank tokenizer faces some issues where some special characters (e.g. "/", ".") are retained in the tokenized words leading to poor matchings. Hence, now we use the Regexp tokenizer from nltk which splits a string into substrings using a regular expression and does not retain the punctuations or special characters.

- **Spelling Correction**

The tokens are then checked for spelling errors and corrected if found any. For this purpose we use TextBlob. Each token is converted into a TextBlob object and then the *correct* method is applied to it which returns the correctly spelled token.

- **Lemmatization**

The spell corrected tokens are then reduced with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma. In order to achieve this, the tokens are tagged with the parts of speech they belong to using nltk's *pos_tag* method and then WordNet Lemmatizer (from nltk) is used to lemmatize the tagged tokens.

- **Stop Word Removal**

Finally, the tokens which occur very frequently in the text which may not be significant to the domain under consideration are removed using a list of english stopwords from nltk.

2. Models

After the data preprocessing step, the documents are represented as numerical vectors/features of the words present in the documents. Queries are also passed through same preprocessing to get a numerical query vector indicating the terms in the query. Next we use the following model to perform document retrieval for the given query:

- **Vector space model (VSM):**

We first carry out the term document indexing to get the indices of documents present in each term. Then we calculate Inverse document frequency (IDF) values of a given term. The document vectors are constructed by multiplying the term frequency and inverse document frequency (IDF) values of a term in a given document. Similarly a query vector is constructed by multiplying the term frequency of each term in the query with the inverse document frequency (IDF) of that term. In case the query contains words which are not seen before a smoothing factor of 1 is included in the IDF formula to give the IDF value of the unseen word as log(N) (N is the number of documents).
IDF_smooth = log(D/(df+1)) + 1

- **Latent semantic indexing (LSI):**

Here, we perform singular value decomposition on the tf-idf weighted term document matrix, A. The decomposition is as follows: $A = U\Sigma V^T$. After performing the decomposition, we clip the U and V matrices keeping only the first k columns. The Σ matrix is made diagonal having first k singular values in descending order as its entries. The value of k is determined by calculating the frobenius norm of the difference matrix, $A - A_k$ based on a certain threshold. The query vector is transformed using the following expression: $q' = \Sigma^{-1}U^Tq$.Then, the cosine similarity between the transformed document and query vectors are calculated for relevance ranking.

- **Query expansion (using wordnet):**

We observe that for some queries we miss retrieving the right documents because the query has different words with similar semantic meaning to that of the required document. Convection VSM does not consider any semantic similarity. It is predominantly based on term-term similarity of document and query. To ensure that semantic similarity is also taken into account along with the term-term similarities between the document and query, we use the query expansion method. We use word-net to get synset for all the words in a given query and add a certain number of synonyms to the query. Since the model is bag-of words, adding more terms does not disrupt the query representation in any undesirable way. We control the weights of the synonyms (always less than the original word) to ensure original words are given appropriately more importance. We also control the number of synonyms from the syset added to the query vector for each term. Adding too many words will improve recall but harm accuracy. Since VSM similarity is based on term-term similarity of document and query, this method is expected to improve the recall of the system as now query has more terms. <can add about tuning 2 parameters now>

- **Word2Vec:**

  The publicly available Google word2vec embeddings is used to provide the word embeddings for each word. The documents and queries are then modelled using these word embeddings corresponding to the terms present in them.

  After getting the document and query methods from the above methods, we calculate the similarity metric as cosine of the angle between the two vectors. In vector form, It can be expressed as:

$$Score(\vec{d}\vec{q}) = \frac{\sum_{k=1}^{m} d_k \cdot q_k}{\sqrt{\sum_{k=1}^{m} (d_k)^2} \cdot \sqrt{\sum_{k=1}^{m} m(q_k)^2}}$$

$$Score(\vec{d}\vec{q}) = 1 \, when \, d = q$$

$$Score(\vec{d}\vec{q}) = 0 \, when \, d \, and \, q \, share \, no \, items$$

  Where, dk and qk is the kth element of the final document vector d and query vector q. The retrieved documents are arranged in the order of cosine value of its angle with the query.

  Note: As planned in the proposal, we also tried bigram indexing and wordnet term-term similarity indexing which had the potential of improving the IR system, but due to computational expense we couldn't implement it to get the output.

3. Evaluation

For quantitative evaluation, we have different types of metrics for information retrieval systems:

- **Precision:**

  It is defined as the ratio of number of correct documents retrieved divided by the total number of documents retrieved. In information retrieval we use a metric called precision at k which is precision for the top k retrieved documents. *Average Precision (MAP)* is calculated by taking an average of all the precision@K values where the Kth position has a correct document.

- **Recall:**

  Recall is the number of correct documents retrieved divided by the total number of correct documents

- **F-Score:**

  It is defined as the harmonic mean of precision and recall. Usually precision is high when recall is low and vice versa so, finding just a mean of the two values would not do any good. Whereas F-score penalises the lower values accordingly and thus helping to build a better model.

- **Normalized Discounted Cumulative Gain(nDCG):**

  Discounted Cumulative Gain (DCG) is the metric of measuring ranking quality. It is mostly used in information retrieval problems such as measuring the effectiveness of the search engine algorithm by ranking the articles it displays according to their relevance in terms of the search keyword.
  The traditional formula of DCG (Discounted Cumulative Gain) for a given query accumulated at a particular rank position p (or k) is defined as:

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)} = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2(i+1)}$$

  Where rel_i is the graded relevance of the result at position i. It is obtained from subject matter experts.
  For a query, the normalized discounted cumulative gain, or nDCG, is computed as:

$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCG_p},$$

  where IDCG is ideal discounted cumulative gain,

$$\mathrm{IDCG_p} = \sum_{i=1}^{|REL_p|} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

  and REL_p represents the list of relevant documents (ordered by their relevance) in the corpus up to position p.

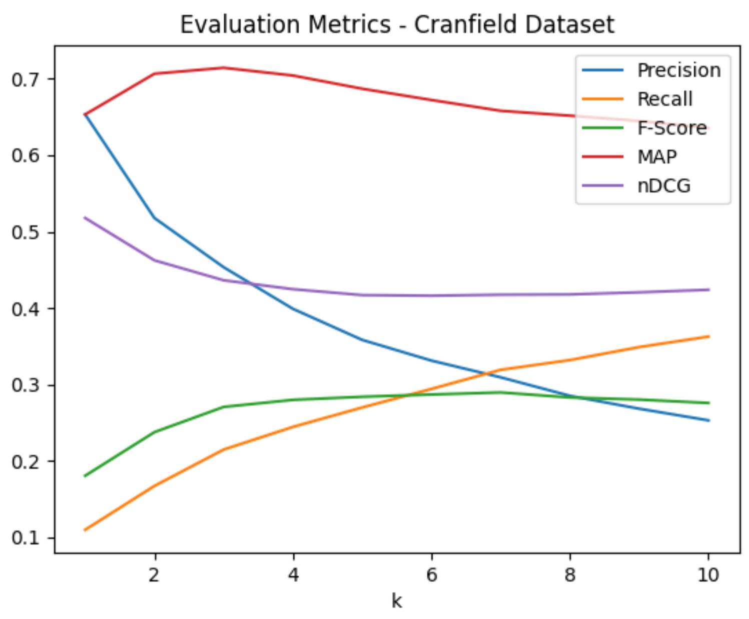
  We primarily use this metric for evaluation and hypothesis testing for model comparison.

6. Experiments and Results

- **Vector Space Model (baseline):**

  Precision, Recall and F-score @ 1 : 0.65333, 0.10962, 0.18039
  MAP, nDCG @ 1 : 0.65333, 0.51778
  Precision, Recall and F-score @ 2 : 0.51778, 0.16703, 0.23744

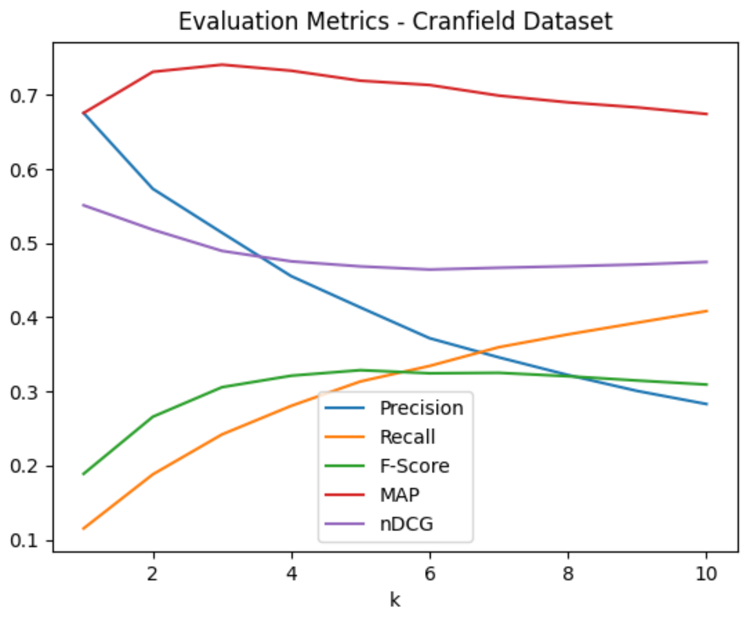
MAP, nDCG @ 2 : 0.70667, 0.46238
Precision, Recall and F-score @ 3 : 0.45333, 0.21464, 0.27050
MAP, nDCG @ 3 : 0.71444, 0.43625
Precision, Recall and F-score @ 4 : 0.39889, 0.24431, 0.27974
MAP, nDCG @ 4 : 0.70432, 0.42460
Precision, Recall and F-score @ 5 : 0.35822, 0.26973, 0.28378
MAP, nDCG @ 5 : 0.68702, 0.41687
Precision, Recall and F-score @ 6 : 0.33111, 0.29402, 0.28673
MAP, nDCG @ 6 : 0.67228, 0.41607
Precision, Recall and F-score @ 7 : 0.30921, 0.31903, 0.28945
MAP, nDCG @ 7 : 0.65812, 0.41740
Precision, Recall and F-score @ 8 : 0.28500, 0.33181, 0.28284
MAP, nDCG @ 8 : 0.65175, 0.41776
Precision, Recall and F-score @ 9 : 0.26815, 0.34882, 0.28009
MAP, nDCG @ 9 : 0.64481, 0.42051
Precision, Recall and F-score @ 10 : 0.25289, 0.36254, 0.27563
MAP, nDCG @ 10 : 0.63559, 0.42384



- **VSM with improved tokenizations and spell corrections:**

  Precision, Recall and F-score @ 1 : 0.67556, 0.11535, 0.18900
  MAP, nDCG @ 1 : 0.67556, 0.55111
  Precision, Recall and F-score @ 2 : 0.57333, 0.18827, 0.26597
  MAP, nDCG @ 2 : 0.73111, 0.51819
  Precision, Recall and F-score @ 3 : 0.51407, 0.24213, 0.30583
  MAP, nDCG @ 3 : 0.74074, 0.48960
  Precision, Recall and F-score @ 4 : 0.45556, 0.28068, 0.32131

MAP, nDCG @ 4 : 0.73259, 0.47553
Precision, Recall and F-score @ 5 : 0.41333, 0.31355, 0.32878
MAP, nDCG @ 5 : 0.71912, 0.46870
Precision, Recall and F-score @ 6 : 0.37185, 0.33451, 0.32458
MAP, nDCG @ 6 : 0.71330, 0.46447
Precision, Recall and F-score @ 7 : 0.34603, 0.35977, 0.32523
MAP, nDCG @ 7 : 0.69904, 0.46692
Precision, Recall and F-score @ 8 : 0.32222, 0.37703, 0.32057
MAP, nDCG @ 8 : 0.68993, 0.46891
Precision, Recall and F-score @ 9 : 0.30074, 0.39279, 0.31481
MAP, nDCG @ 9 : 0.68315, 0.47129
Precision, Recall and F-score @ 10 : 0.28311, 0.40843, 0.30938
MAP, nDCG @ 10 : 0.67424, 0.47464



- **Latent Semantic Indexing:**

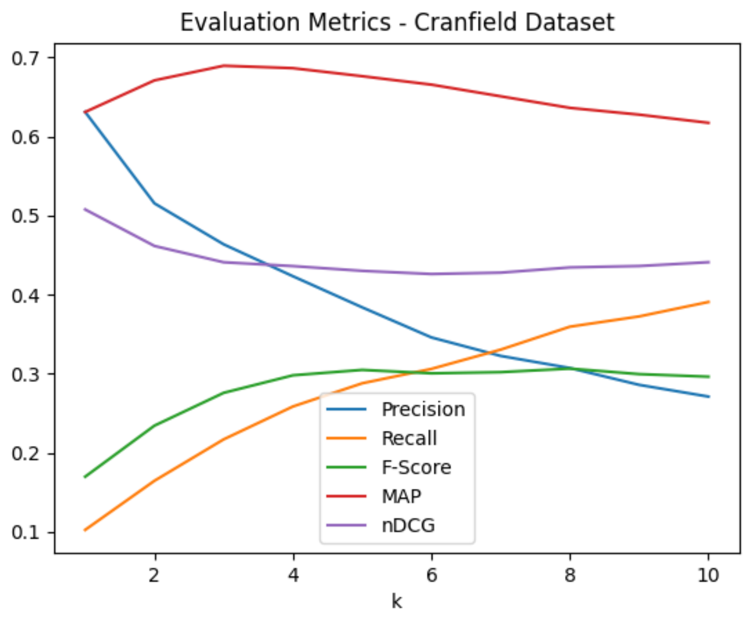
  Here we need to find the value of k, to get the k rank approximation of the term document matrix, A. We use a threshold of 0.4 and try values of k from 1 till a value of k such that:

$$|A - A_k| < 0.4$$

  Finally, we use the value of k = 200

  Precision, Recall and F-score @ 1 : 0.63111, 0.10251, 0.16977
  MAP, nDCG @ 1 : 0.63111, 0.50778
  Precision, Recall and F-score @ 2 : 0.51556, 0.16459, 0.23433
  MAP, nDCG @ 2 : 0.67111, 0.46166
  Precision, Recall and F-score @ 3 : 0.46370, 0.21701, 0.27585

MAP, nDCG @ 3 : 0.68963, 0.44102
Precision, Recall and F-score @ 4 : 0.42333, 0.25857, 0.29802
MAP, nDCG @ 4 : 0.68654, 0.43617
Precision, Recall and F-score @ 5 : 0.38400, 0.28791, 0.30486
MAP, nDCG @ 5 : 0.67640, 0.43016
Precision, Recall and F-score @ 6 : 0.34593, 0.30613, 0.30052
MAP, nDCG @ 6 : 0.66571, 0.42616
Precision, Recall and F-score @ 7 : 0.32254, 0.33050, 0.30200
MAP, nDCG @ 7 : 0.65096, 0.42789
Precision, Recall and F-score @ 8 : 0.30722, 0.35952, 0.30652
MAP, nDCG @ 8 : 0.63634, 0.43450
Precision, Recall and F-score @ 9 : 0.28593, 0.37250, 0.29952
MAP, nDCG @ 9 : 0.62776, 0.43627
Precision, Recall and F-score @ 10 : 0.27111, 0.39078, 0.29627
MAP, nDCG @ 10 : 0.61742, 0.44108

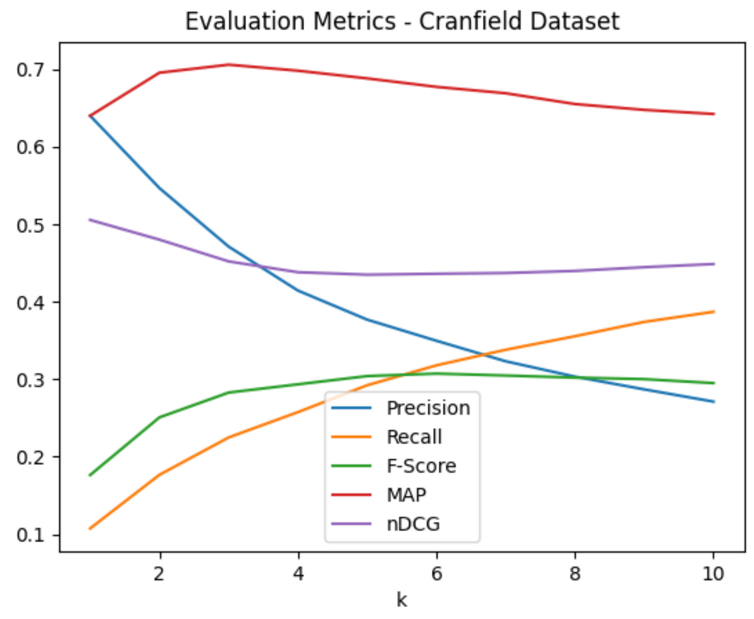


- ## Query Expansion

There are two parameters to be tuned here, the number of synonyms to be added to the original query, n and the term frequency discounting of the new terms added, α. Hence term frequency of a synonym = $\alpha \times tf(original\, term)$.After tuning we use $n = 4,\ \alpha = 0.85$

Precision, Recall and F-score @ 1 : 0.64000, 0.10744, 0.17636
MAP, nDCG @ 1 : 0.64000, 0.50556
Precision, Recall and F-score @ 2 : 0.54667, 0.17662, 0.25075
MAP, nDCG @ 2 : 0.69556, 0.48012

Precision, Recall and F-score @ 3 : 0.47111, 0.22486, 0.28278
MAP, nDCG @ 3 : 0.70593, 0.45206
Precision, Recall and F-score @ 4 : 0.41444, 0.25754, 0.29343
MAP, nDCG @ 4 : 0.69815, 0.43809
Precision, Recall and F-score @ 5 : 0.37689, 0.29222, 0.30406
MAP, nDCG @ 5 : 0.68821, 0.43491
Precision, Recall and F-score @ 6 : 0.34963, 0.31788, 0.30725
MAP, nDCG @ 6 : 0.67731, 0.43608
Precision, Recall and F-score @ 7 : 0.32317, 0.33801, 0.30478
MAP, nDCG @ 7 : 0.66906, 0.43707
Precision, Recall and F-score @ 8 : 0.30333, 0.35548, 0.30209
MAP, nDCG @ 8 : 0.65515, 0.43975
Precision, Recall and F-score @ 9 : 0.28691, 0.37396, 0.30012
MAP, nDCG @ 9 : 0.64759, 0.44459
Precision, Recall and F-score @ 10 : 0.27111, 0.38702, 0.29501
MAP, nDCG @ 10 : 0.64232, 0.44863

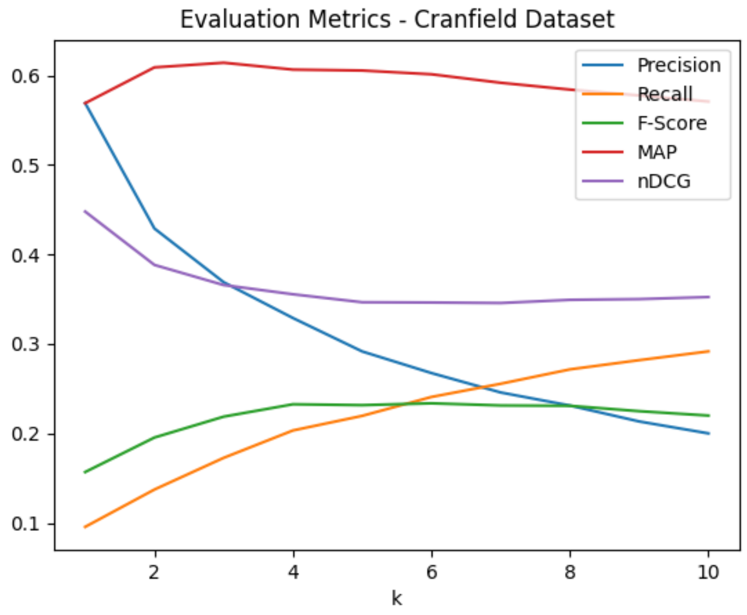


- **Word2Vec**

Here, we used google's pre-trained word2vec model to get the word embeddings.

Precision, Recall and F-score @ 1 : 0.56889, 0.09558, 0.15683
MAP, nDCG @ 1 : 0.56889, 0.44778
Precision, Recall and F-score @ 2 : 0.42889, 0.13728, 0.19537
MAP, nDCG @ 2 : 0.60889, 0.38820
Precision, Recall and F-score @ 3 : 0.36889, 0.17268, 0.21870
MAP, nDCG @ 3 : 0.61407, 0.36572

```
Precision, Recall and F-score @ 4 : 0.32889, 0.20328, 0.23253
MAP, nDCG @ 4 : 0.60642, 0.35543
Precision, Recall and F-score @ 5 : 0.29156, 0.21958, 0.23153
MAP, nDCG @ 5 : 0.60538, 0.34651
Precision, Recall and F-score @ 6 : 0.26741, 0.24069, 0.23361
MAP, nDCG @ 6 : 0.60119, 0.34621
Precision, Recall and F-score @ 7 : 0.24571, 0.25544, 0.23120
MAP, nDCG @ 7 : 0.59170, 0.34565
Precision, Recall and F-score @ 8 : 0.23111, 0.27150, 0.23083
MAP, nDCG @ 8 : 0.58406, 0.34918
Precision, Recall and F-score @ 9 : 0.21333, 0.28181, 0.22480
MAP, nDCG @ 9 : 0.57754, 0.35001
Precision, Recall and F-score @ 10 : 0.20000, 0.29162, 0.21991
MAP, nDCG @ 10 : 0.57075, 0.35233
```



7. Model comparisons and hypotheses testing

Since there is no single quantitative metric describing the exact performance of an IR system, we obtain evaluation metrics for various queries and perform hypothesis testing to check if the average score of the model is higher than the baseline VSM model. From all the metrics previously discussed, we choose nDCG @ 4 as the evaluation metric for a given query in a given IR system. We choose 4 as the ground truth for relevant documents is available for k = 4. For hypothesis testing, we consider nDCG values for all queries obtained from model1 of the IR system as population1 and similarly values obtained from model2 as population2. The hypothesis is to check if the average nDCG of model 2 is greater than model 1. For this we perform a two tailed paired t-test hypothesis test with a significance level $\alpha = 0.15$. We calculate the 85% confidence interval of the difference in means.

- If the interval contains 0, m1 = m2
- If the interval is negative, m1 < m2
- If the interval is positive, m1 > m2

Mean NDCG of population1 (model1) = m1
Mean NDCG of population2 (model2) = m2

Null hypothesis: m1 = m2
Alternate hypothesis: m1 < m2
For all tests we consider baseline VSM as model1.

Model being compared with baseline	**t-statistic**	**p-value**	**Accept/Reject H_0**	**Confidence interval of difference in means**	**Conclusion**
VSM with corrections	-1.2538	0.1064	Reject	$[-0.0390, -0.0036]$	Better than baseline
LSI	-0.5164	0.3034	Failed to Reject	$[-0.0287, 0.0097]$	Not Better than baseline
VSM with query Expansion	-0.3983	0.3456	Failed to Reject	$[-0.0247, 0.0111]$	Not Better than baseline
Word2Vec	3.3470	0.9994	Failed to Reject	$[0.0618, 0.1177]$	Worse than baseline

8. Conclusion

After applying different models we observe the following:

<u>VSM with corrections:</u>

We observe that when we apply improved tokenization along spell check on both queries and documents, the average nDCG score has increased compared to the baseline VSM model. There were considerable spelling errors in both queries and documents, due to which the term-term overlap between a query and a potentially relevant document is decreased resulting in lower similarity score. Applying spell check has resolved this issue and improved the model. Also, we observed that the treebank tokenizer faces some issues where some special characters (e.g. “/”, “.”) are retained in the tokenized words leading to poor matchings. Hence, using the Regexp tokenizer solved the issues of punctuations present in the tokens thereby providing better matchings.

<u>Latent semantic indexing (LSI) :</u>

In LSI we used SVD on the term-term matrix of documents and queries to get a lower dimensional representation of the documents and queries in concept space. But in our case the model does not seem to be significantly improving with this addition. It may be attributed to lack of latent structure in our data set. Using the value of k as large as 200 doesn’t give us better results than VSM as there may be no well defined concepts in the given dataset which is key for LSI’s performance.

VSM with query Expansion:

We used query expansion using wordnet to capture semantic similarity along with term similarity between documents and queries. The expected results were not obtained possibly due to the fact that the synonyms which we added might not be present in the documents. Since we added synonyms from wordnet, it was not a context aware method. For a particular context we have a smaller set of synonyms and that would help us achieve the improvement which we aimed for. But in this case we used a general synset for query expansion.

Word2Vec:

Here also it can be observed that the results are comparatively worse compared
to vector space models because of the small dataset. Fine-tuning the word2vec embeddings on small dataset can cause a lot of issues like : –

- There might be no, or very few, examples where the desired-to-be-alike words are in similar nearby-word contexts. With no examples where there are shared nearby-words, there's little basis for nudging the pair to the same place
- The internal Word2Vec task, of predicting nearby words, doesn't need them to be near each other, if they're each predicting completely different words.
- The pre-trained word2vec model may not contain some key terms present in the cranfield dataset resulting in poor embeddings and vector conversions.
- A much better chance to move from an initially-arbitrary location to a useful location.

Since only the co-occurrence is concerned, the word vector contains limited semantic information of the word vector which is not very much efficient in retrieving the documents containing synonyms.

9. References

1. "Information Retrieval" Wikipedia, en.wikipedia.org/wiki/Information_retrieval.
2. CS6370 course lectures and material.
3. "Information Retrieval System Explained" Upgrad, www.upgrad.com/blog/information-retrieval-system-explained.
4. "RegExp Tokenizer." NLTK Documentation, www.nltk.org/_modules/nltk/tokenize/regexp.html.
5. "Latent Semantic Analysis" Wikipedia, en.wikipedia.org/wiki/Latent_semantic_analysis.
6. Pawde, K., Purbey, N., Gangan, S., & Kurup, L. "Latent Semantic Analysis for Information Retrieval" www.erpublication.org/published_paper/IJETR022617.pdf.
7. "Document Search Using Vector Space Model" Data Science Central, www.datasciencecentral.com/profiles/blogs/information-retrieval-document-search-using-vector-space-model-in.
8. "Textblob Spelling Correction" Towards Data Science, towardsdatascience.com/textblob-spelling-correction-46321fc7f8b8.
9. Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." *arXiv preprint arXiv:1310.4546* (2013).