Information Retrieval System for the Cranfield Dataset

Emil Biju and D. Tony Fredrick

Indian Institute of Technology Madras
EE17B071 & EE17B154

Abstract. The project aims to develop and improve an information retrieval system built on the Cranfield dataset. We perform experiments on new vector representation methods, faster query processing, a query auto-completion system and an automated spelling correction system. Through these improvements, we build a more accurate, reliable and user-friendly search engine that works with an interactive application.

Keywords: Information Retrieval · Natural Language Processing · Cranfield · Scientific text NLP · Domain-specific NLP

1 Introduction

In this project, we explore methods to develop and improve an Information Retrieval system on the Cranfield dataset. We perform experiments and devise approaches aimed at improving document/query representation, retrieval efficiency and product usability. We start by exploring various sentence and word-level vector representation methods like LSA, Glove Word Embeddings, Glove-weighted TF-IDF and the Universal Sentence Encoder. Next, we explore ways of combining word or sentence-level representations into query/document vectors using simple averaging, differential weighting, sentence-specific comparisons, etc.

Next, we explore ways to improve the speed of retrieval using parallelization techniques and pre-computing of intermediate results to demonstrate a 66% reduction in runtime from that reported in Assignment 2. Further, we develop a query auto-complete system that leverages query similarity, recentness and popularity and devise a novel methodology using parse trees to evaluate our model.

Then, we devise a spelling correction approach inspired by ideas shared in the class and report 83.82% accuracy on a wide range of possible spelling errors. Finally, we develop an interactive application that acts as an interface to our Python scripts for ease of use and experimentation.

2 Problem Statement

The purpose of this project is to build a Information Retrieval system/Search Engine that operates on a dataset of documents related to aerodynamics called

the Cranfield dataset. The developed system must accept both user-entered queries and queries available from the file `cran_queries.json` containing 225 queries and retrieve relevant documents from the set of documents available in the `cran_docs.json` file containing 1400 documents. The system is evaluated on standard evaluation metrics like precision, recall, F1-score, nDCG and mean average precision using the ground truth relevance rankings provided in the `cran_qrels.json` file.

Besides, we extend the problem statement to address further issues in the IR system. The system is expected to have fast retrieval time and low computational cost at runtime. Besides, the system must have a query auto-completion system and an automated spelling correction system to reduce the load on the user and to improve the likelihood of the system returning useful results.

3 Motivation

On analysing the performance of our Information Retrieval system developed as part of Assignments 1 and 2 of the course, we observed the following shortcomings. The motivations for each section of our project arise from the need to address these to develop a more user-friendly, accurate and efficient system.

1. **High vector dimensionality:** When the corpus has long documents and a large vocabulary, the TF-IDF vectors have high dimensions and small dot-products, making it difficult to distinguish the relative relevance of documents for a query.
2. **Low recall due to synonymy:** Some relevant documents were not retrieved because it used synonyms to words in the query.
3. **Low precision due to polysemy:** Some irrelevant documents are retrieved due to false positive matches of words in documents that carried a different sense from the query.
4. **Slow processing:** Retrieving relevant documents for all Cranfield queries together took about 1 min, 20 sec which could be improved.
5. **Computationally intensive:** High vector dimensionality also leads to higher computational costs.
6. **Bag of words assumption:** The TF-IDF vector representation does not consider the ordering of words in the document/query.
7. **Cognitive load on the user:** The entire query has to be entered accurately by the user to obtain appropriate results. An auto-completion/spelling correction system can reduce the load on the user.

4 Background and Related work

4.1 GloVe Word Embeddings

Glove embeddings [2] are vector representations of words that are said to capture relationships between words that encode their meanings. To understand how

GloVe embeddings capture semantic information, we learn how these embeddings are created. Given the term-term co-occurrence matrix X, we know that X_{ij} represents the co-occurrence between the ith and jth words. Hence,

$$P(j|i) = \frac{X_{ij}}{\sum X_{ij}} = \frac{X_{ij}}{X_i}$$

The objective of GloVe training is to learn vectors for each word such that the dot-product of the vectors is high if the log probability of their co-occurrence is high. Hence,

$$v_i^T v_j = \log P(j|i) = \log X_{ij} - \log X_i$$
$$v_j^T v_i = \log P(i|j) = \log X_{ij} - \log X_j$$

Adding the 2 equations, we get

$$v_i^T v_j = \log X_{ij} - 0.5 \log X_i - 0.5 \log X_j$$

Noting that $\log X_i$ and $\log X_j$ act as biases, we get

$$v_i^T v_j = \log X_{ij} - b_i - b_j$$

The training objective can then be formulated as

$$\min_{b_i, b_j, v_i, v_j} \sum_{i,j} (v_i^T v_j + b_i + b_j - \log X_{ij})^2$$

The objective may further be modified to give higher weighting to words that co-occur neither too rarely nor too frequently.

4.2 Latent Semantic Analysis

This is a method for obtaining compressed representations of documents by generating a low-rank approximation of the term-document matrix. It helps to capture inherent relationships between meanings of terms by the assumption that words that are close in meaning occur in similar pieces of text. First, the SVD (Singular value decomposition) of the term-document matrix T is computed.

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

The 1st k components (corresponding to the k largest eigenvalues) from each row of V^T are used to represent each document. Effectively, LSA reduces the vector dimensionality by using a small set of k concepts to represent documents. Then, each query is mapped to the same space to get k-dimensional representations of queries.

$$q_k = \Sigma_k^{-1} U_k^T q$$

where q is the original TF-IDF vector of the query and q_k is the new representation.

The similarity between a document and a query is then computed by the cosine similarity between their corresponding k-dimensional vectors.

5 Proposed Methodology

5.1 Improving Vector Representations

In Section 3, we discussed issues with our Information Retrieval system. We observed that issues $1, 2, 3, 5, 6$ can be addressed by using better vector representations than TF-IDF. Vector representations that capture semantic relations between words, are low dimensional and encode the ordering of words are the ideal requirements.

GloVe Word Embeddings

We first explore the use of GloVe word embeddings as a way of representing each word in a document/query. The word representations can then be combined into document/query representations. Here, we make use of the 300-dimensional pre-trained GloVe embeddings for each word as obtained from [1]. However, there are two issues:

- These embeddings are trained on the Common Crawl dataset [2] obtained from common webpages. However, the Cranfield dataset is scientific and domain-specific. Hence, some words in the Cranfield dataset could have meanings that are different from the commonplace meanings of the words.
- Due to the domain-specificity of the Cranfield dataset, not all words are covered in the pre-trained GloVe embedding set.

Cranfield-trained GloVe

To address the two issues mentioned above, we train the GloVe vectors on the Cranfield dataset by treating its documents as the corpus. This generates vectors for words absent from the standard GloVe vocabulary and also obtains more domain-specific vectors (since they are trained based on the co-occurrence of words within the Cranfield dataset). However, we observed that since the Cranfield dataset is relatively small, this corpus alone is insufficient to fully capture the relation between words. Hence, for words that are present in the standard GloVe vocabulary, we take the sum of 0.9 times the original GloVe vector and 0.1 times the newly generated one. Giving higher weightage to the newly generated one was observed to reduce performance since the corpus it is trained on is small. This would enable a mixing of the general vector and the domain-specific one. For the remaining words, the newly generated vector is solely used.

GloVe-weighted document/query vectors

As will be discussed in the results, the aggregated GloVe/USE vectors do not perform as well as the TF-IDF because a large number of relevant (query, document) pairs closely share specific words and some of this information is lost

[1] Source of pre-trained GloVe embeddings: https://nlp.stanford.edu/projects/glove/

[2] Common Crawl dataset: https://commoncrawl.org/the-data/

in aggregation. This method is inspired by the query expansion approach presented in class.However, we expand both the query and the document using the following method as it was observed to give better results. First, we note that the dot product between GloVe vectors is informative of the closeness between two words. We normalise all the GloVe vectors so that they have unit norm. Then, for every vocabulary word, we pre-compute the dot product between its GloVe vector and the GloVe vector of every other vocabulary word. If $|V|$ is the vocabulary size, this gives a $|V| \times |V|$ matrix with each row corresponding to one vocabulary word in the alphabetical order. If $glove\_sim_t$ is the row vector for term t, doc_TI is the TF-IDF vector for the document, TI_t is the TF-IDF for term t in the document, the vector for each document is given by:

$$doc\_vec = doc\_TI + \sum_{t \in doc} \alpha \times TI_t \times glove\_sim_t$$

The above sum is over all terms t in the document. The above equation effectively adds the TF-IDF vector to a set of vectors, one corresponding to each term in the document which captures similarity not just with the same term, but with the remaining terms in the vocabulary as well. The query vector is also computed in a similar way. $\alpha = 0.01$ was observed to give the best results.

The reasoning for this approach is as follows:

- For indices corresponding to terms in the document, the TF-IDF vector will have large non-zero values. Since this is added to *doc_vec*, large values for these indices are maintained.
- There may be words that are close in meaning to words in the document but which do not occur in the document. The *glove_sim* vector weighs such words depending on its closeness to the actual word. By adding this vector, we are giving some weight to such words in the vocabulary depending on its relevance to the document terms.
- We give lower weightage to terms that are not present in the document by using the factor $\alpha < 1$.

WordNet weighted document/query vectors

This method is similar to the one presented above, except that we use the Wordnet similarity between terms as a measure of their similarity instead of the dot product of normalized GloVe vectors. Given two synsets s_1 and s_2, their similarity is measured using the WuPalmer – WordNet Similarity as follows

$$wup(s_1, s_2) = 2\frac{depth(LCS(s_1, s_2))}{depth(s_1) + depth(s_2)}$$

This similarity measure gives a measure of the "distance" between the synsets in the WordNet graph. For ease of implementation, the first sense of each synset was used. For terms that do not exist in WordNet, a similarity of 0 was assigned. $\alpha = 0.03$ was observed to perform best in this case. As will be seen later, these representations perform better than the TF-IDF vectors.

Latent Semantic Analysis

In this method, as described in Section 4, LSA is used to obtain k dimensional vector representations of documents and queries. The cosine similarity between these vectors is used as a measure of the similarity between a document and query. A k value of 200 is used in our work which is decided following experiments in Section 7.1. We attempt 3 variants of LSA as follows:

- LSA with the document/query vectors as their TF-IDF vectors
- LSA with the document/query vectors as the Glove-weighted vectors (explained above)
- LSA with the document/query vectors as the WordNet-weighted vectors (explained above)

Universal Sentence Encoder (USE)

The Universal Sentence Encoder (USE) [4] makes use of a pre-trained Deep Learning-based Transformer architecture to encode sentences. The generated representations are context-aware and take into account the ordering and identity of all words in the sentence due to the self-attention mechanism used in the Transformer. A 512-dimensional vector is generated by the Transformer encoder for each word which are then summed together to get a representation for the sentence. To ensure generalizability, USE is pre-trained on several standard tasks including conversational tasks, text classification, unsupervised learning, etc. This makes the sentence vectors useful for new tasks like Information Retrieval (in our case).

Common advantages: (i) Faster processing, (ii) Lower dimensionality (GloVe: 300, USE: 512 v/s TF-IDF: 8282)

Explicit Semantic Analysis (ESA)

In contrast to LSA, Explicit Semantic Analysis is a method that tries to keep the dimensions of the document vectors explicit or meaningful. In this method, we work with a subset of domain-specific Wikipedia articles. Each Wikipedia article is treated as one dimension. The number of Wikipedia articles becomes the length of the vector representation of each word. We can assign the strength in each "dimension" to be equal to the number of times the word occurs in the Wikipedia article corresponding to that dimension. The sum of the vector representations of a document's constituents words gives us the vector representation of the document. The same method can be used to find the vector representation for the query too. Some additional information specific to our implementation:

- A list of 224 articles in the domain of aerodynamics was obtained using the PetScan online tool.
- With the help of the wikipedia Python library, we extracted the content of the pages using the Page IDs returned by PetScan. The content was preprocessed to remove equations, symbols, reference links etc. and the words were converted to their root forms using lemmatization.

- The vector representation of each word in the Cranfield corpus vocabulary was calculated and then stored in a JSON file.
- At runtime, we can generate the vector representation of the queries and documents using the pre-computed word vectors.
- The vectors are then ranked using cosine-similarity.

5.2 Combining Vector Representations

Simple Averaging

The naive method for combining vector representations is to simply average the word vectors (in case of GloVe) or sentence vectors (in case of USE) to obtain an embedding for the document/query. The advantages are: (i) fast processing and (ii) simple implementation. The disadvantages are: (i) loss of information when several vectors are aggregated into a single vector, (ii) No preferential weighting to important words/sentences

Differential title weighting

We note that the title of a document contains topical and highly relevant information about what the document provides. It is likely that a query also asks for such topical information which may be present in the title. Hence, we choose to give thrice the weight to the title compared to the remaining sentences in the document. The weighted average of the word (or sentence) vectors is then taken. This addresses disadvantage (i) stated above.

Sentence-wise comparison

We further wish to minimise the loss of information due to averaging of a large number of vectors. Hence, we propose to compare each sentence (including the title) from the document with the query vector separately. Hence, if there are s_d sentences in document d, there are s_d sentence vectors which may be obtained directly from USE or by averaging GloVe vectors of words within the sentence. The cosine similarity between the query vector and the s_d sentence vectors are obtained separately and the maximum among them is regarded as the similarity score between the document and query. By doing this, we are finding the similarity between the 'most relevant' sentence from the document to the query.

5.3 Improving Retrieval Speed/Efficiency

This section explains methods that we used to improve the speed of retrieval at runtime. From our experiments (as will be shown in Table 1), lemmatization takes a major chunk of the processing time. Hence, we used parallelization to reduce the time involved in this process. For this purpose, we used the `joblib` library to distribute the task over available CPU cores, so that each core completes the task for a subset of sentences.

To reduce the delay of our system further, we observed that a large proportion of the computation that occurs at runtime is pre-determined and can hence be pre-computed and stored. When a new query is issued, the system can then retrieve the pre-computed files and only perform the necessary computation that is specific to the entered query.

Hence, we pre-compute the following files in order to save runtime.

- Pre-processed document and query text from the Cranfield dataset (after segmentation, tokenization, lemmatization and stopword removal)
- Inverted Index (document IDs with token frequencies)
- TF-IDF vectors of all Cranfield queries (useful for comparison in auto-complete system)
- Query Log (for storing popularity and recentness in auto-complete system)
- Cranfield-trained GloVe embeddings
- SVD of the TF-IDF matrix for LSA
- WordNet similarity matrix for all terms in the vocabulary

We also allow these files to re-computed at runtime, if required in special cases (for example, if the dataset is altered). To enable this, the `-preprocess` argument can be passed at runtime in the terminal or the `Preprocess @ Runtime` button can be clicked in the interactive application (Section 6).

5.4 Query Auto-Completion System

System Design

For this purpose, we define 3 metrics that influence the prediction of our auto-complete system:

1. **Similarity**: This is the most crucial metric for comparison. It measures the similarity between the user-entered incomplete query and the candidate queries in the dataset.
2. **Recentness**: This is a measure of how recently each candidate query has been issued to the system by a user. It is based on the proposition that queries that have invoked interest in the recent past are more likely to be of interest to a new user.
3. **Popularity**: This is a measure of the number of times a candidate query was issued to the system. It is based on the proposition that more popular queries are more likely to be issued again by a user.

Implementation

For our auto-completion system, we declare the set of queries in the Cranfield dataset as our *candidate queries*. We first compute the TF-IDF vectors of all candidate queries based on the inverted index developed for the terms in the documents. This is stored as a separate file which can be retrieved at runtime to save computational time.

Computing similarity: When a user enters an incomplete query, the TF-IDF vector of this query is computed and dot products with vectors of all candidate queries are found. The top 5 candidate queries with the highest dot product values are shortlisted and each dot product value is divided by the highest dot product value among the 5 queries. This provides a normalized similarity score that lies between 0 and 1.

Query Log: We maintain query log consisting of all the candidate queries from the Cranfield dataset which is updated each time a new query from the dataset is entered to the system. At each such instance, the latest time instant at which the query is issued is updated and the total number of times the query has been issued is incremented by 1 in the query log.

Computing recentness and popularity: To measure recentness of each of the 5 shortlisted queries, the time span between the current time instant and the instant when the query was last issued (as obtained from the query log) is computed and then normalized by dividing by the longest time span among the 5 queries. This provides the recentness score. To measure popularity, the total number of times the query was issued is normalized by the largest such value among the 5 queries to obtain a popularity score.

The 3 normalized scores are then combined to get an auto-complete score (AC) as

$$AC(q) = 0.5 \times similarity(q) + 0.3 \times popularity(q) + 0.2 \times recentness(q) \quad (1)$$

The weightage given to each metric is decided by the following arguments: (i) Since similarity between queries is the most determining factor for relevance, atleast 50% weightage must be given to this metric, (ii) Relative weightage for popularity and recentness are derived from the work in [3], (iii) An additional reason for lower weightage for recentness is that updates to the latest time of issue of a query are quite discontinuous, due to few users using the system at discontinuous time intervals. This makes the recentness metric slightly less reliable.

Each shortlisted query is ranked based on the AC score obtained above with the 1st rank assigned to the query with the highest score.

5.5 Spelling Correction System

We devise two methods for enabling automated spelling correction in our system. For each term in the user-entered query, we first check if the term is present in our vocabulary. We refer to those terms which are not found in the vocabulary as *invalid terms*.

Method 1

In this method, we develop a unigram language model that gives the likelihood of each term in the vocabulary. We compute the likelihood as

$$LS(w) = \frac{count(w_q)}{count(w_c)}$$

where, w_q is the number of times the term w occurs in the query, w_c is the number of times it occurs in the corpus and V is the vocabulary size. We note that we do not need to use Laplace smoothing since all the words for which the likelihood is computed are part of the vocabulary and hence, have frequency ≥ 1. Now, for each invalid term in the user-entered query, we generate all possible words at edit distances of 1 and 2. Then, we obtain the likelihood of each generated word using the unigram model and regard the word with the highest likelihood as the potential replacement for the invalid term.

Method 2

In this method, we extract bigrams from all terms in our vocabulary and create an inverted bigram index where each bigram is linked with all words that contain that bigram. For each invalid term, we extract all bigrams and use the inverted bigram index to find all terms in the vocabulary that share bigrams with the invalid term. After this, we shortlist terms from the vocabulary that have $> 50\%$ overlap in bigrams with the query term as candidate replacements for the query term. Then, we compute the edit distance between the invalid term and each candidate query and regard the term with the lowest edit distance as a potential replacement for the invalid term.

Once potential replacements for all invalid terms are found, the system generates a prompt for the user to verify whether the replacement is acceptable or if a fresh query can be issued by the user. Please note that edit distance above refers to the Levenshtein distance.

6 Interactive Application

We developed an interactive application using the `ipywidgets` library that acts as an interface to the Python script. The application allows the user to choose hyperparameters like the segmenter and tokenizer used, k value for LSA, vector representation method, etc using dropdown lists and text boxes. Custom queries can also be entered and the 5 top retrieved docs are displayed in order within the application which can be run on a Jupyter notebook. A screenshot of the same is seen in Figure 1. This prevents the need for the cumbersome use of the terminal to alter and test various hyperparameters.

7 Experiments

7.1 How do we choose k for LSA?

Here, we explore various ways of choosing the value of k for the LSA. 3 methods have been proposed in the literature and we work with all 3 of them to arrive at the optimal value. The first method is to plot the singular values obtained after performing SVD against the component number. This plot is referred to as a scree plot and is shown in Figure 2. The component number where the knee

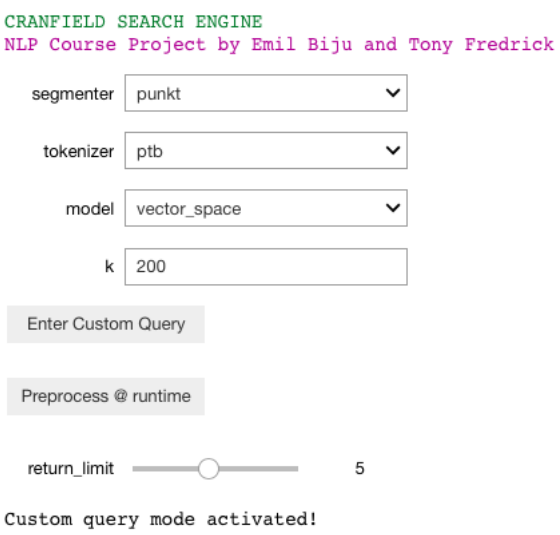

Fig. 1: Screenshot of the interactive application for the search engine

of the plot occurs, i.e, where there is a sudden change in the gradient can be chosen as the value of k. We observe that the knee occurs in the 100-200 range.

The second method is to plot the cumulative proportion of variance explained by the first k components. If s_k is the kth singular value, the proportion of variance explained due to the is computed as

$$pov(k) = \frac{s_k^2}{\sum_{m=1}^{M} s_m^2}$$

The cumulative variance @ k, say c_k is computed as

$$c_k = \sum_{m=1}^{k} pov(k)$$

A rule of thumb is to ensure that atleast 70% of the variance is considered. We observe that this is satisfied at $k = 300$.

The third method is to plot the metric used for evaluating the downstream result of the system and to find the value of k at which the system gives the best result. This is shown in Figure 3. We plot the Precision and Recall values @ 1 and @ 10 and observe that they peak around $k = 200$. We note that the trends for other metrics (like nDCG, F1-score, etc.) are also similar. While one might expect the precision to increase at high k for LSA, it reduces here for very high k. To explain this, we note that unlike the usual setting, the vectors for different sentences in a query/document are aggregated (by averaging or other methods as discussed). Each dimension in a very high dimensional vector individually stores little information about the sentence which on averaging is easily lost. At very

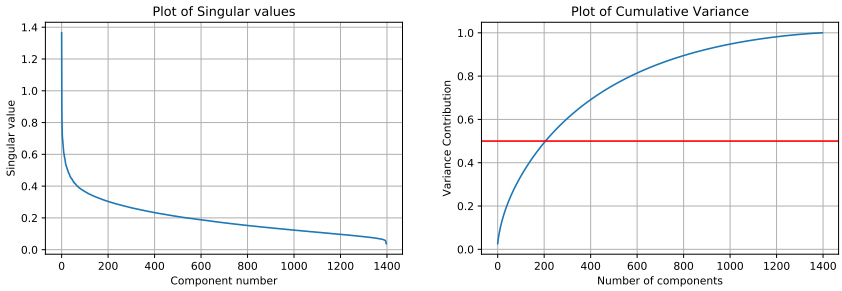

Fig. 2: (a) Plot of the singular values of each component, (b) Plot of the cumulative variance over components

low k, precision is low because low dimensional vectors have a *blurring* effect, i.e, identifying the most relevant document from similar documents is challenging.

From the above 3 results, a value of k in the range of 100-300 can be considered ideal. Since the downstream results are best for $k = 200$, we report further results on the same.

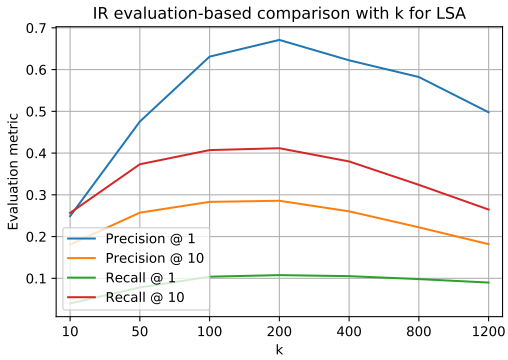

Fig. 3: Plot of the Precision and Recall metrics across different choices of k for LSA

7.2 How good are the runtime gains due to pre-computing?

Here, we discuss the runtime gains achieved by pre-computing basic information as mentioned in Section 5.3. The break-up of the runtime incurred by each operation is shown in Table 1. By storing the pre-processed files (after segmentation, tokenization, lemmatization and stopword removal), we save nearly 24s in runtime since these computations need not be performed. Besides, the inverted

index and TF-IDF matrix only need to be loaded from memory instead of being built at runtime. The total runtime has reduced from 52.511s to 24.577s which implies a reduction in runtime by 53.2%.

In the case of LSA, computing the SVD matrix takes 19.235s which by storing the SVD matrix. The overall time for ranking documents (which includes the SVD computation time) reduces from 60.4s to 17.93s. Thus, the overall time for retrieving queries for all documents using the LSA method is reduced from 92.618s to 25.133s, which implies that the runtime is reduced to less than 1/3rd. The gains due to parallelization of the lemmatization step were observed to be minimal (nearly 1s), but by pre-computing the lemmatized documents, this step does not ultimately affect our runtime.

Operation performed	**Computed at runtime (s)**	**Pre-computed (s)**
Segmentation	0.740	0
Tokenization	1.682	0
Lemmatization	21.200	0
Stopword Removal	1.154	0
Total preprocessing	24.795	0
Build Inverted Index	0.849	0
Load Inverted Index	0	1.577
Build TF-IDF	0.233	0.179
Compute SVD	19.235	0
Load SVD matrix	0	5.73
Ranking documents		
— TF-IDF	20.320	17.374
— LSA	60.427	17.930
Running Evaluation	7.396	7.203
Total runtime		
— TF-IDF	52.511	24.577
— LSA	92.618	25.133

Table 1: Break-up of the runtime incurred by each operation involved in retrieving documents for all Cranfield queries. By pre-computing basic information, considerable gains are made. We compare the runtimes for the vector space model (TF-IDF) and for LSA

7.3 Why is cosine similarity not used in auto-complete model?

An important aspect to note in our design choice is that we use the dot product between the TF-IDF vectors of the candidate and incomplete queries instead of using the cosine similarity. This is because it was observed that since the product of vector norms that cosine similarity uses in the denominator scales with the query length, the cosine similarity score sometimes gave better scores to shorter

queries that may not be very relevant. In effect, the length of the query had a greater influence than the similarity. Such an issue does not arise when the cosine similarity between TF-IDF vectors of documents and queries are considered because the documents are several sentences long and the denominator is hence quite large that minor variations in document/sentence lengths have a minimal impact on the overall score.

For example, consider the incomplete query input: **flow solutions**. When normalization is not done (as in our case), the top 3 auto-complete predictions all contain both the words 'flow' and 'solution' or 'solutions'.

1. what approximate **solutions** are known to the direct problem of transonic **flow** in the throat of a nozzle, i.e. finding the **flow** in a given nozzle .
2. can the procedure of matching inner and outer **solutions** for a viscous **flow** problem be applied when the main stream is a shear **flow** .
3. what approximate **solutions** are known to the indirect problem of transonic **flow** in the throat of a nozzle, i.e. finding a nozzle which has a given axial velocity distribution .

However, when cosine similarity is used, the system tends to predict shorter queries from the dataset which may have only one of the words occurring in it. Even among the returned queries, shorter queries are ranked higher even if they are less relevant (with recentness and popularity parameters being the same).

1. previous **solutions** to the boundary layer similarity equations .
2. what approximate solutions are known to the direct problem of transonic **flow** in the throat of a nozzle, i.e. finding the **flow** in a given nozzle .
3. can the procedure of matching inner and outer **solutions** for a viscous **flow** problem be applied when the main stream is a shear **flow** .

Clearly, the results when using the dot product, i.e, without normalization are more relevant than when cosine similarity is used.

7.4 Where does the spelling correction system fail?

We analysed the performance of the spelling correction system on the evaluation dataset that was generated by us. We noticed that while the system achieves an accuracy of 83.82%, the failure cases correspond to the following:

Misspelt word	**Model prediction**	**Expected word**
scientits	scientists	scientist
figurs	figure	figures
dceay	cease	decay
exposrue	expose	exposure

Table 2: Examples of cases where the spelling correction system does not generate the expected result on the evaluation dataset.

- When another valid word from the vocabulary is closer (in terms of edit distance) to the misspelt word than the expected word. These cannot be regarded as failures of the model since the returned words are valid predictions and might indeed be the intended word of the user or an appropriate substitute (like the singular form of a plural word or vice-versa).
  Examples: 1st 2 rows of Table 2.
- When the misspelt word contains transposed characters. Since we use the Levenshtein distance, transposition cannot be done at unit cost and hence transposition of characters is a costly operation (requires 2 insertions and 2 deletions). A solution is to use the Damerau–Levenshtein distance, but incurs longer runtime. **Examples**: last 2 rows of Table 2.

8 Results

8.1 Improved Vector Representations

The results on various IR metrics for different vector representation approaches are shown in Table 3. It must be noted that the '+' symbol means that each new strategy is compounded over the ones adopted in the previous rows. For example, the Cranfield-trained GloVe approach also uses averaging with 3 times weight to the title. It is clear that each strategy improves the results on the IR metrics.

Vector Representation	**Precision**	**Recall**	**F1-score**	**MAP**	**nDCG**
Glove Variant					
+Simple Averaging	(0.42, 0.18)	(0.07, 0.26)	(0.11, 0.19)	(0.42, 0.47)	(0.33, 0.29)
+Thrice weighted title	(0.47, 0.19)	(0.07, 0.27)	(0.12, 0.21)	(0.47, 0.49)	(0.37, 0.32)
+Cranfield-trained GloVe	(0.47, 0.19)	(0.08, 0.28)	(0.12, 0.21)	(0.47, 0.50)	(0.37, 0.32)
+Sentence-wise	(0.54, 0.20)	(0.09, 0.30)	(0.15, 0.22)	(0.54, 0.52)	(0.42, 0.34)
Universal Sentence Encoder					
+Simple Averaging	(0.40, 0.18)	(0.07, 0.27)	(0.12, 0.20)	(0.40, 0.46)	(0.32, 0.30)
+Thrice weighted title	(0.52, 0.18)	(0.09, 0.27)	(0.14, 0.20)	(0.51, 0.53)	(0.42, 0.33)
+Sentence-wise	(0.52, 0.18)	(0.09, 0.27)	(0.14, 0.20)	(0.52, 0.54)	(0.43, 0.34)
Vector Space (TF-IDF)	(0.66, 0.28)	(0.11, 0.40)	(0.19, 0.30)	(0.66, 0.65)	(0.54, 0.46)
+Glove similarity	(0.64, 0.27)	(0.11, 0.38)	(0.18, 0.29)	(0.64, 0.64)	(0.51, 0.44)
Latent Semantic Analysis	(0.67, 0.29)	(0.11, 0.41)	(0.18, 0.31)	(0.67, 0.64)	(0.52, 0.46)
+GloVe similarity	(0.68, 0.29)	(0.11, 0.42)	(0.18, 0.32)	(0.68, 0.65)	(0.54, 0.47)
+WordNet similarity	(0.68, 0.28)	(0.11, 0.40)	(0.18, 0.31)	(0.68, 0.66)	(0.54, 0.46)
Explicit Semantic Analysis	(0.18, 0.09)	(0.03,0.14)	(0.05,0.1)	(0.18, 0.26)	(0.14, 0.14)

Table 3: Comparison of the performance of our Information Retrieval system on the Cranfield query and document set. Each tuple contains the @1 and @10 results. Eg: (Precision @ 1, Precision @ 10)

Considerable improvement on all metrics is observed both in the Glove Variant and the Universal Sentence encoder when sentence-wise comparison is used. This is because quite often just a single sentence from the document may substantially overlap with the query and the remaining sentences may only be adding noise in comparison with the query. Assigning 3 times weight to the title also shows improvement. **Hence, an approach that uses thrice-weighted title and sentence-wise comparison performs better than the an approach that simply averages the vectors on all 5 evaluation measures on the Cranfield dataset.** The performance of the Universal Sentence Encoder based approach is close to that obtained from the GloVe variants, but not better. A possible reason is that we use the pre-trained weights without specifically training on the Cranfield dataset as is done for GloVe. Hence, domain-specific training of embedding vectors is beneficial in improving performance.

The vector space model/TF-IDF approach performs better than the above approaches. Reasons are the following: (i) No aggregation of sentence/word vectors is involved here. (ii) Relevant queries usually have exact word overlaps with the documents. This is usually the case in scientific/domain-specific texts since there are specific terminologies that are used to ask or explain about specific concepts.

LSA gives the best results on our experiments across metrics. This shows that in the Cranfield dataset, co-occurring words tend to have similar meanings and compressed vector representations are able to represent the documents and queries well. Besides, we note that LSA incurs less computational cost due to much lower dimensional vectors. **Hence, LSA performs better than TF-IDF vector space model on all 5 evaluation measures (except MAP @ 10) on the Cranfield dataset.**

GloVe similarity outperforms TF-IDF in the vector space method. Besides, GloVe similarity and WordNet similarity-based vector representations with LSA outperform the LSA method with TF-IDF. **Hence, GloVe and WordNet based query/document expansion techniques do better than naive TF-IDF on all 5 evaluation measures on the Cranfield dataset.**

We make a few key observations regarding the advantages of different representation methods. Using WordNet/GloVe based similarity metrics helps overcome the synonymy issue: when the query *'thermodynamic'* is entered, we observed that the results were poor as most of the relevant documents contained the word *'thermodynamics'* and not *'thermodynamic'*. But given that the GloVe similarity and WordNet similarity between these two words is high, using the GloVe/WordNet similarity vectors give a high weight to the word *'thermodynamics'* and hence the relevant documents are retrieved.

Secondly, suppose our query is a very specific phrase that occurs only in a single sentence from a document in the dataset, say *'axisymmetric hypersonic blunt bodies'*. Sentence-wise comparison methods return the correct document (Doc 85) containing this sentence as the first result because the similarity computed between the query and this sentence gives a very high score. However, when sentence-wise comparison is not used and the query is compared to documents

as a whole, documents in which the individual words occur across its various constituent sentences get higher score (the terms may not occur together as a phrase and may differ in meaning from the intent of the query). Hence, this method also helps to give importance to the ordering of words in the query as a sentence in which all the words co-occur is assigned a higher score.

8.2 Query Auto-completion

For obtaining a quantitative score for our auto-completion system, we first need to generate a large number of possible incomplete queries that a user may enter the system. It is also important that these incomplete queries are correctly labelled with the most relevant candidate query. The work by [1] states that users are likely to enter noun phrases into the text box of a search engine when trying to express a query. We also note that noun phrases within queries in the Cranfield dataset express topical information that helps understand the purpose of the query and the kind of information that the query seeks.

Hence, we create parse trees from all Cranfield queries using the `RegexpParser` from the `nltk` library. Then we extract all noun phrases from the parse tree to create a set of 945 incomplete queries. Each incomplete query is labelled with the complete query from which it was derived as the ground truth for auto-completion. We run these queries through our auto-completion system which generates 5 ranked results. Then, we use the Mean Reciprocal Rank (MRR) to measure the performance, both in terms of retrieving the relevant query and assigning it a high rank.

$$MRR = \frac{1}{|IQ|} \sum_{i=1}^{|IQ|} \frac{1}{rank_i}$$

where $rank_i$ is the rank assigned to the relevant query for the incomplete query i and $|IQ|$ is the number of incomplete queries used. We report an MRR of **0.683** for our auto-complete system. We also report the accuracy of this method as the percentage of incomplete query inputs for which the correct query is returned in any one of the top 5 places. The resultant accuracy that we achieve is **87.8%**.

8.3 Spelling Correction

Speed considerations: On experimenting with the two proposed methods in Section 5.5 for spelling correction, we observed that Method 1 is too slow for practical use. To generate all words at edit distance of 1 from a word with n characters, the following possibilities exist: n deletions, $26n$ substitutions and $26(n+1)$ insertions, which results in a total of $53n+26$ words. Therefore, generating all possible words at edit distances upto 2 from a given word implies that $\approx (53n+26)^2$ words have to be generated. Hence, it takes nearly 20 seconds to generate the potential replacement for each word. However, Method 2 is much faster and runs in 3-4 seconds which is more suitable for practical use.

Accuracy: To evaluate the accuracy of our spelling correction system, we first need to generate a labelled dataset containing misspelt words and the corresponding spelling corrections. For this, we take each word from the vocabulary, modify these words randomly by inserting a random character, deleting a character, transposing characters or replacing a character with another one. The modified words are then passed to the spelling correction system and we obtain accuracy with which the correct source word is predicted. Method 2 was observed to achieve an accuracy of **83.82%**.

9 Conclusion

Hence, through this project, we have developed an information retrieval system on the Cranfield dataset that achieves high performance on standard IR metrics like precision, recall, F1-score, nDCG and MAP. The system also achieves fast processing of queries at runtime. It is also built to be user-friendly by enabling query auto-completion, spelling correction and an easy-to-use user interface application. We would like to thank Prof. Sutanu Chakraborti for his invaluable guidance and support through this well organized course. We also thank the TA's for their valuable time and effort as a part of this course.

References

1. Bougouin Adrien, Boudin Florian Daille, Béatrice TopicRank: Graph-Based Topic Ranking for Keyphrase Extraction, Asian Federation of Natural Language Processing. https://www.aclweb.org/anthology/I13-1062
2. Pennington Jeffrey, Socher Richard, Manning Christopher GloVe: Global Vectors for Word Representation, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). https://www.aclweb.org/anthology/D14-1162
3. Giovanni Di Santo, Richard McCreadie, Craig Macdonald, Iadh Ounis, Comparing Approaches for Query Autocompletion. Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval. https://dl.acm.org/doi/10.1145/2766462.2767829
4. Daniel Cer, Yinfei Yang, Sheng-yi Kong, et. al., Universal Sentence Encoder for English. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. https://www.aclweb.org/anthology/D18-2029/