Now Lucene Me Now You Don't

Isla Hoe

hoei@tcd.ie

# ***Abstract- This paper presents a search engine built using the Lucene 7.2.1 API. It was evaluated using a collection of four documents from the 8th TREC Conference [1]****.* ***The performance of this search engine was measured using the TREC-Eval tool. This implementation resulted in a Mean Average Precision (MAP) of 0.2999.***

**1**. **INTRODUCTION**

The purpose of any information retrieval (IR) system is to perform a search over a set of documents in response to a query given by a user and return a set of the most relevant documents.

Major search engines are increasingly striving for faster and more precise results. Google, for example, has implemented “must include” features, where users can now specify that parts of their query must appear in the results of a search.

A query is a set of terms submitted to a search engine, designed to describe the information the user is looking for. Query formation is a key factor in creating an effective search engine. Because of the difficulty associated with extracting semantic meaning and context from a query, most IR models depend on well formulated queries [2].

The IR model presented in this paper is built using *Apache Lucene* [3]*.* Lucene is actually a search engine moreover it's a “ AP and code library which can easily be used to add search capabilities to IR applications”.

The collection of documents used in this project are from four sources, Financial Times Limited (1991, 1992, 1993, 1994), the Federal Register (1994), the Foreign Broadcast Information Service (1996) and the Los Angeles Times (1989, 1990). These documents are queried using topics 401-450 from the TREC-8 collection and evaluated using the metrics; Mean Average Precision (MAP) and individual precision and recall scores from the TREC-Eval metric generation tool.

**2**. **IMPLEMENTATION**

Implementation of this model consists of two main phases; Indexing and Querying. Both phases require some level of text processing which was carried out using a custom built Analyzer.

**2.1 Custom Analyzer**The custom analyzer, *StandardStemAnalyzer()* was constructed to tailor operations performed on all text streams, by positively manipulating the text in order to optimise the overall performance of the search engine.

*2.1.1 Stemming*

Stemming was employed to reduce each word into smaller tokens, for example, common suffixes such as ‘s’, ‘ed’ and ‘es’ are removed. The Porter Stemming algorithm (*PorterStemFilter*) which reduces words with the same meaning to a common root was used. However, the KStemFilter was also tested but it was found that it resulted in lower MAP scores.

*2.1.2 Stop Word Removal*

Stopwords are words which help form a sentence structure but contribute very little on their own to the description of the topic. These words include words like ‘the’, ‘as’ and ‘is. The stopwords used for this implementation were taken from the Onix Text Retrieval Toolkit [4].

*2.1.3 Normalisation*

The *EnglishPossessiveFilter* was used to remove all the possessives (trailing 's) from words. The *LowerCaseFilter* was used to convert each token to a full lowercase token which makes it for easier comparison and which is also needed for filters that only work with lowercase tokens.

## **2.2 Indexing Phase**

Each collection of documents (and queries) were ‘indexed’ to convert the text into a usable format. Document and query objects were created which contained a set of fields unique to each collection Both documents and queries were parsed using the open source java library Jsoup. The fields for both documents and queries were chosen by manually inspecting the tags for each collection and choosing those which were deemed most useful.

*2.2.1 Documents*

Many fields were considered initially that seemed like they could be a possible identifier for a document however given the fact that only the text field was consistent across all 4 data sources, that was the only one used in the final program. Fields such as title and date appeared regularly in some but irregularly in others and when added to the index lowered the MAP score and so were left out.

To combat this lack of consistency across data sources; if the information in a document seemed like it could be relevant such as the “graphic” field in the LATimes or the “footnote” field in the FR, those fields were appended to the text field so that that information wasn’t lost.

*2.2.2 Queries*

The list of topics was indexed using the same method as the document indexing. Each query was created as a *HashMap.* These queries were the benchmark for each final query. During the querying phase, these queries are refined and expanded. Before extracting the information the query was parsed and then separated into a set of elements, where each element contains the original query text. The document number, title, description and narration were added as fields for this query. In order to provide the query with some level of context a set of relevant and non-relevant terms were extracted based on text from the ‘narration’.

This implementation was very naive and simply identified keywords in each sentence by searching for the terms ‘relevant’, ‘not relevant’ or ‘irrelevant’ to identify within that sentence which terms existing in the sentence should be deemed non-relevant. In order to accomplish this, 26 unit tests were created where the relevant and non-relevant terms are manually identified in each narrative. Both the algorithm and a custom stop-word list were adjusted in order to maximise the number of tests that pass.

The full set of queries is returned as a list of hashmaps.

**2.3 Querying Phase**

As mentioned the construction of the query is highly important, research has shown that the length and occurrence of query terms has a significant effect on the returned results [5]. The querying phase was confined to a single function (*queryHandler)* and can be seen in figure 2. This phase was broken down into four functions; *query*, *createQuery*, *constructPhaseQuery*, and *runQuery*.

*2.3.1 Query*

This function calls the other three functions, stores the results from each query search and writes those results to file which can be run using Trec\_Eval 9.

*2.3.2 CreateQuery*

The query then needed to be constructed from the parsed information extracted from the topics file. This was done using a boolean query. The boolean query was constructed from the following:

1. The title of the topic.
2. The description of the topic.
3. The relevant terms from the narrative.
4. The phrases in the title.

All of these parts of the query were fed through the analyser used to index the documents before the query was performed.

*Phrases*

It is possible to implement phrase matching by creating a set of phrases during the indexing phase [6], however it was found that it was far more efficient and resulted in better map scores if phrases were constructed using the description text from the topics files. These phrases were constructed without removing stopword and sets of two and three words.

*Boosting*

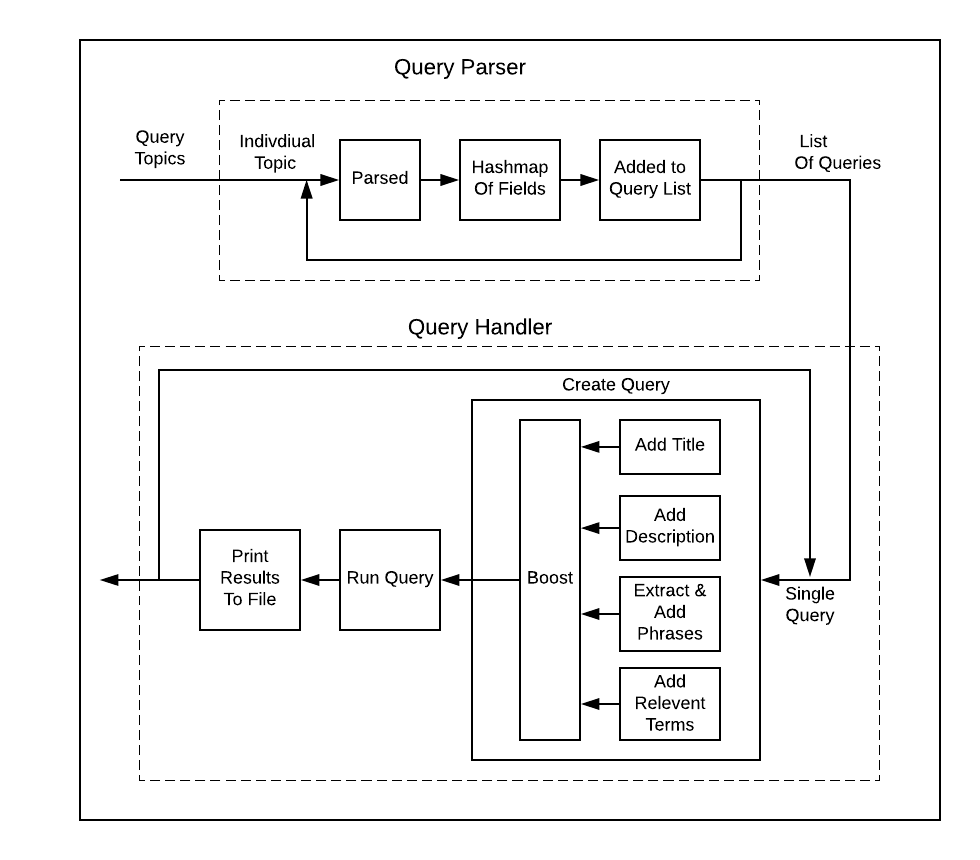
Initially, all query terms and fields hold the same level of importance. By applying the *BoostQuery* function, the importance of each can be manipulated in order to place greater emphasis on more relevant terms/fields. The following is a breakdown of our relevant boost values:

* Phrases were boosted by 2.5
* The title was boosted by 1.5
* Both the description and relevant terms were boosted by 0.5

The final boost values were decided based on the effect they had on the MAP score, and were tested by changing the percentage of the boost applied not merely the value of the boost.

*2.3.3 RunQuery*

Instead of constructing all the queries, storing them in a query index and then executing them one by one each query object was created and then immediately executed, such the result of each query was stored ut the query object was not.



**Figure 2:** Block diagram of the querying phase

**3**. **TESTING AND EVALUATION**

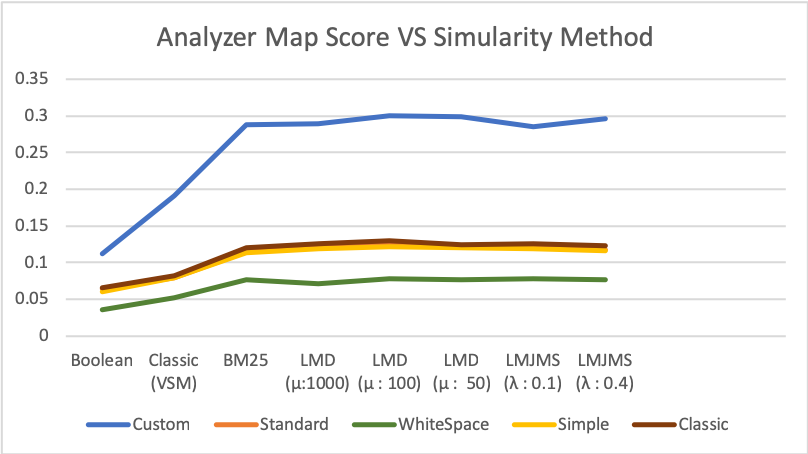
**3.1 Retrieval Models**

In the area of Information Retrieval, the Postulates of Impotence outline how difficult it can be to define relevance for a single user, mainly due to its subjective nature. Various retrieval models have been developed which attempt to mimic relevance decisions by trying to formalise the underlying decision-making process of the user [7].

*Similarity Methods*

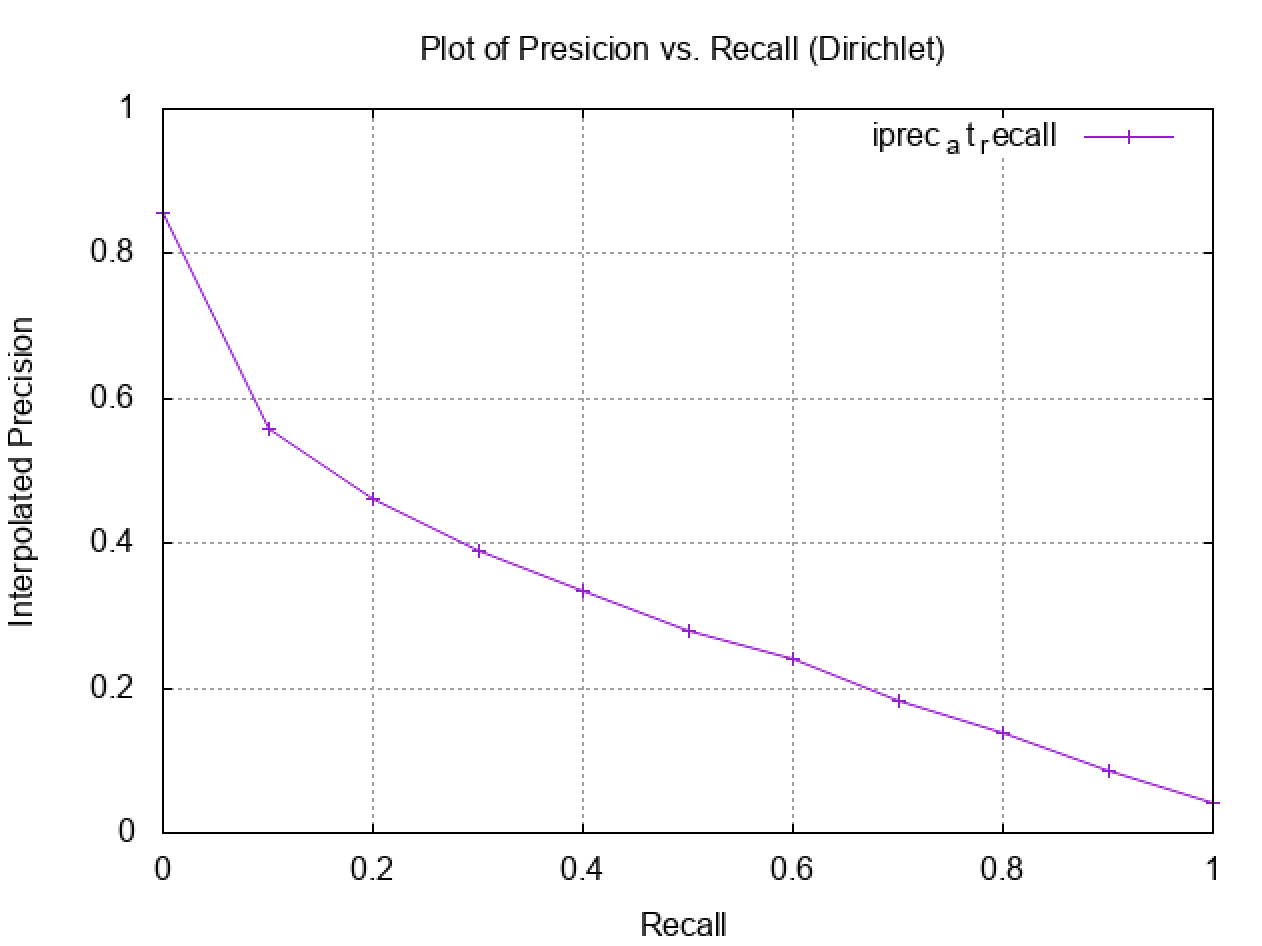
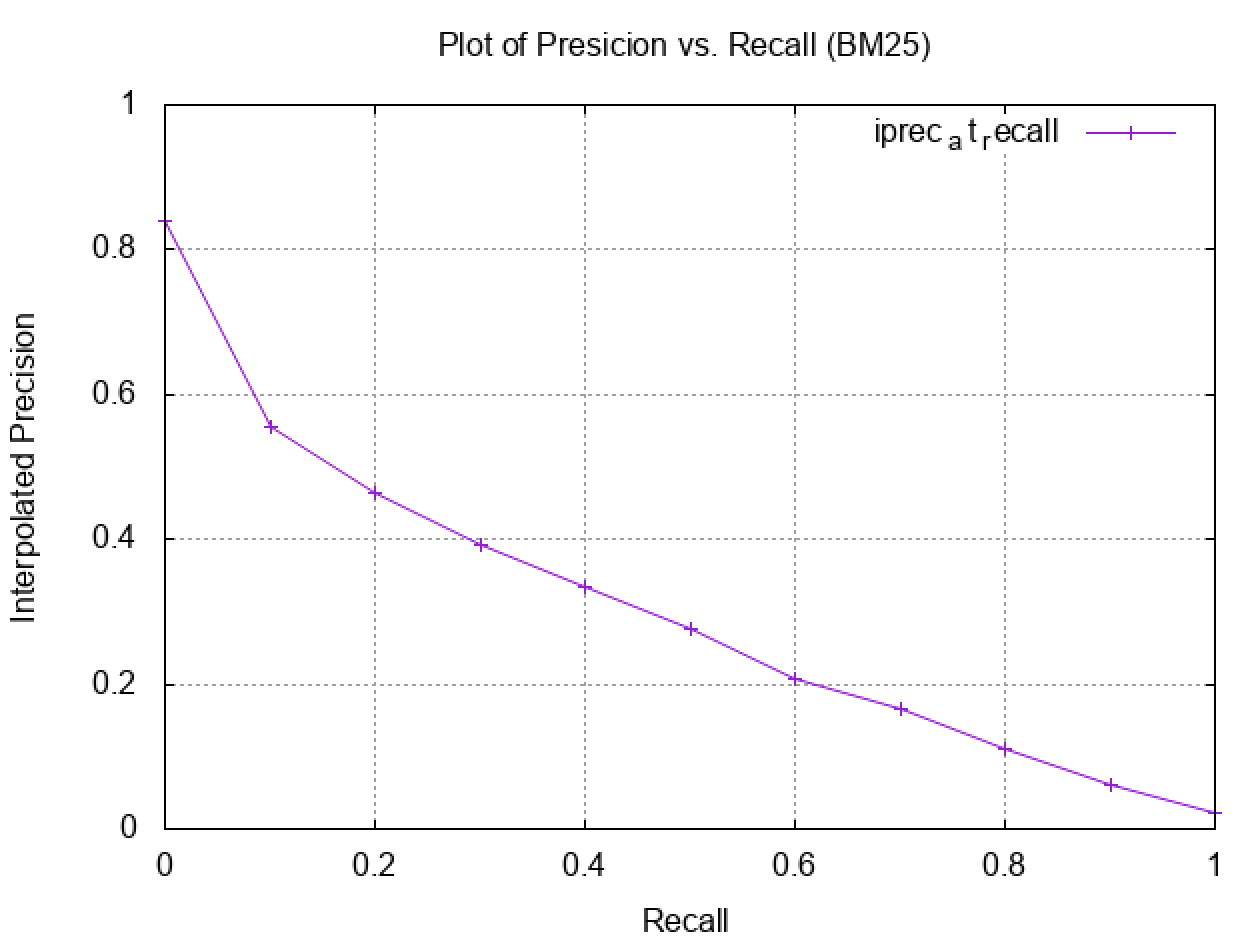
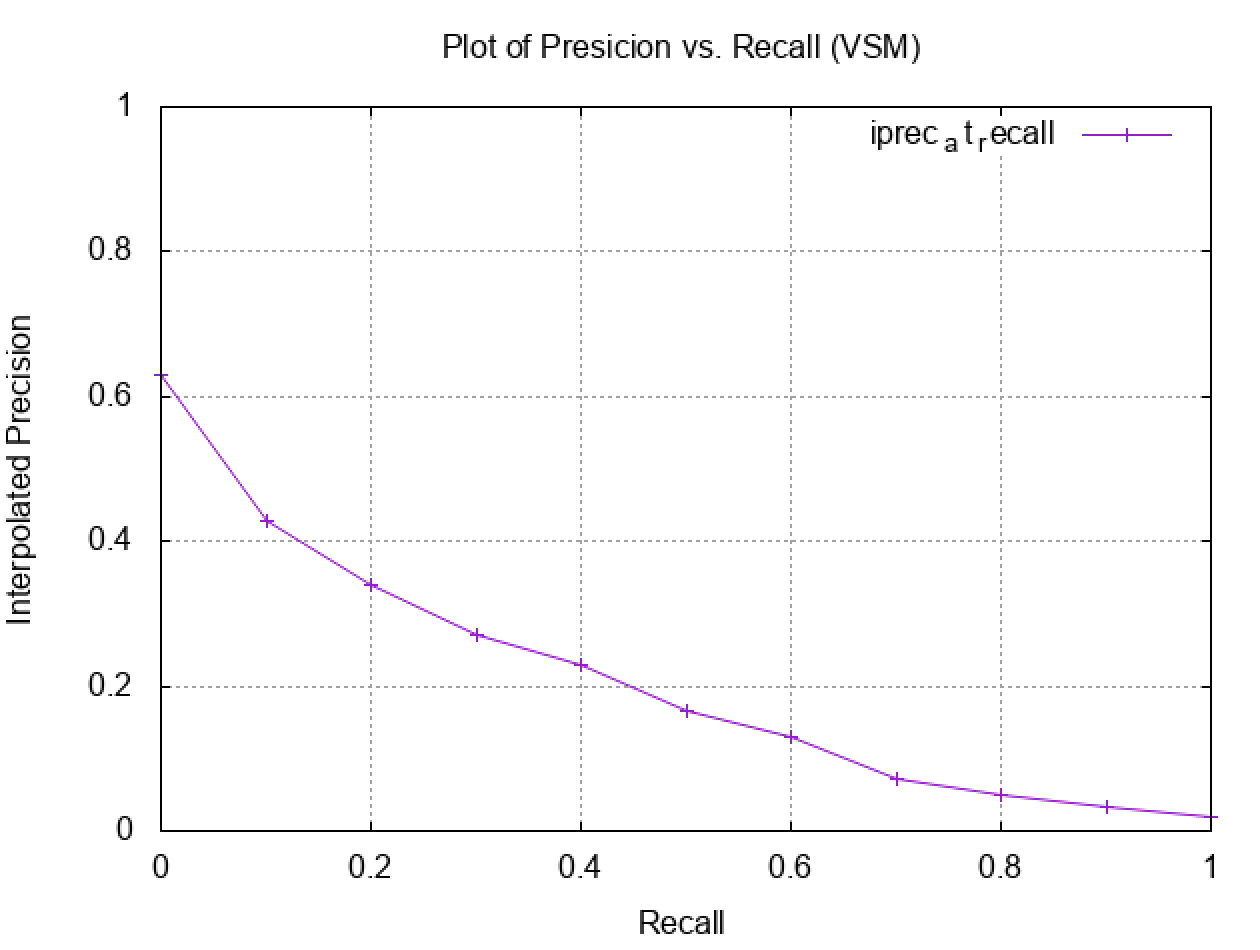
Five different similarity methods were implemented and tested. These models included the Boolean retrieval model, the Vector Space Model, BM25 and two language models; Dirichlet and Jelinek-Mercer [8]

The Dirichlet Language model resulted in the highest MAP scores. This model generally provides the best results for short queries and is a natural way to specify prior knowledge of estimating the probabilities in a multinomial distribution.



**Fig. 3.a:** MAP scores for each analyzer and similarity model

Estimation is based on the document length, ɑ, and µ, where µ relates to the relative weighting of terms or the number of matching terms. Large values for µ favour the number of matching terms whereas small values for µ place more importance on the relative weighting of words [2]. As can be seen, in figure 3.a various values for µ were tested and the optimum value for this system was µ = 100, which resulted in a map score of 0.2999. This is likely due to the fact that the queries which were constructed were heavily dependent on the importance of specific terms.

****

**Fig. 3.b:** Recall v Precision (Dirichlet) **Fig. 3.c:** Recall v Precision (BM25) **Fig. 3.d:** Recall v Precision (VSM)

The Dirichlet model proved slightly more successful than the Jelinek-Mercer model which calculates a query likelihood ranking using interpolation smoothing. However it's clear that the use of language models for evaluation are more effective than the other models.

Comparing the graphs created using the gnuplot function in terminal in conjunction with trec eval the figures 3.b, 3.c and 3.d it's clear that although the BM25 similarity model and the Dirichlet model produce similar MAP scores, 0.2873 and 0.2999 respectively, the precision using the BM25 model drops at lower recall value than the Dirichlet model.

**3.2 Analyzers**

The custom analyzer proved very successful in optimising the map scores, excluding the results of the boolean similarity method it preformed drastically better than all of the other standard analyzers. This makes it clear that it's necessary to customize filters with regards to different collections.

**4**. **DISCUSSION & FURTHER WORK**

One of the most unintuitive outcomes of this project was the inability to contextualize the query in order to increase the relevant results. An attempt was made by incorporating the use of the non-relevant terms from the narrative text by calculating the TF-IDF score and only using the most unique terms in the non-relevant phrase and adding them to the Boolean query as “Must not” terms.

However by implementing a “must not” query at such an early stage it removed a large number of documents which may have been deemed relevant because some of the non relevant words such as “disaster” would have appeared in relevant papers in a different context and the “must not” term removed them completely.

Although that effort was abandoned a second algorithm was developed but not yet implemented which is expected to provide more satisfactory results. By using the concept of pseudo relevance feedback [9] to query a set of documents and then requery the results of those documents the following steps would be achievable in performing a better search.

1. Compile the extended query to include terms classified as relevant.
2. Using this query compile, in order of relevance, a list of results.
3. Create a second two part query which:
   1. Looks for the non-relevance condition
   2. Tries to satisfy that condition.
4. Run the second query over the initial set of resulting documents to get a second set of results.
5. Remove these results from the initial set.
6. Return this as the final set of results.

In addition to adding a phrase query, a name field was also created under the assumption that boosting that field would increase the amount of retrieved relevant documents and thus improve the precision score. However after implementing this it was found to have no improvement on the MAP score and when the boolean “must” was added it greatly decreased the MAP score and only marginally increased the precision score. In order for this query to have a decent effect, it is recommended that a synonym filter be added in order to highlight topical documents using different names or nicknames.

**5. CONCLUSION**

The final MAP score achieved in this project was 29.99%. It has been clear throughout the process that the implementation of key features such as efficient analyzers and choosing the right similarity model are the key to developing a successful search engine.

**REFERENCES.**

[[1] “Text REtrieval Conference (TREC) Publications.” [Online]. Available: https://trec.nist.gov/pubs.html. [Accessed: 30-Nov-2018].](https://www.zotero.org/google-docs/?Wx9KZk)

[[2] “Search Engines Information Retrieval in Practice.” [Online]. Available: https://ciir.cs.umass.edu/irbook/. [Accessed: 28-Nov-2018].](https://www.zotero.org/google-docs/?Wx9KZk)

[[3] *Apache Lucene*.](https://www.zotero.org/google-docs/?Wx9KZk)(Available at: <http://lucene.apache.org/>) [Accessed on 30-Nov-2018] [.](https://www.zotero.org/google-docs/?Wx9KZk)

[[4] “Onix API Manual.” [Online]. Available: http://www.lextek.com/manuals/onix/. [Accessed: 30-Nov-2018].](https://www.zotero.org/google-docs/?Wx9KZk)

[[5] B. He and I. Ounis, “Studying Query Expansion Effectiveness,” in *Advances in Information Retrieval*, vol. 5478, M. Boughanem, C. Berrut, J. Mothe, and C. Soule-Dupuy, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 611–619.](https://www.zotero.org/google-docs/?Wx9KZk)

[[6] O. Gospodnetić and E. Hatcher, *Lucene in action*. Greenwich, CT: Manning, 2005.](https://www.zotero.org/google-docs/?Wx9KZk)

[[7] D. Swanson R., “Historical note: Information retrieval and the future of an illusion.,” Journal of the American Society for Information Science, 1988, pp. 92–98.](https://www.zotero.org/google-docs/?Wx9KZk)

[[8] D. . MacKay and L. C. . Peto, “A hierarchical Dirichlet language model. Natural language engineering,” 1995, pp. 289–308.](https://www.zotero.org/google-docs/?Wx9KZk)

[[9] Z. Zhong and H. T. Ng, “Word Sense Disambiguation Improves Information Retrieval,” in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Jeju Island, Korea, 2012, pp. 273–282](https://www.zotero.org/google-docs/?Wx9KZk)