IMPLEMENTING AN INFORMATION RETRIEVAL SYSTEM

TUESDAY 29TH OCTOBER 2019

PATRICK KNOTT

Executive Summary

Self-learning is a crucial component of tertiary education, it prepares students for the workforce or for academia. Both professional paths require an ability for independent education. As part of this process, Kingsland University of Technology offer their students a knowledge platform, in the form of an online information retrieval system. Essentially, a pre-selected source of information on a variety of subjects.

This project sought to improve the whole system. The generation of the database, the methods of searching, and increase the options for students to use the results.

The newer system had small increases to the time to generate the system, time to search and the size of the system on disk. When measured via trec, the newer system scored slightly lower than the old one.

However, this does not take into account the new ability to iteratively improve the search. The small decreases in other metrics is a worthwhile price for the new systems capabilities to improve, expand, and sharpen search queries.

TABLE OF CONTENTS

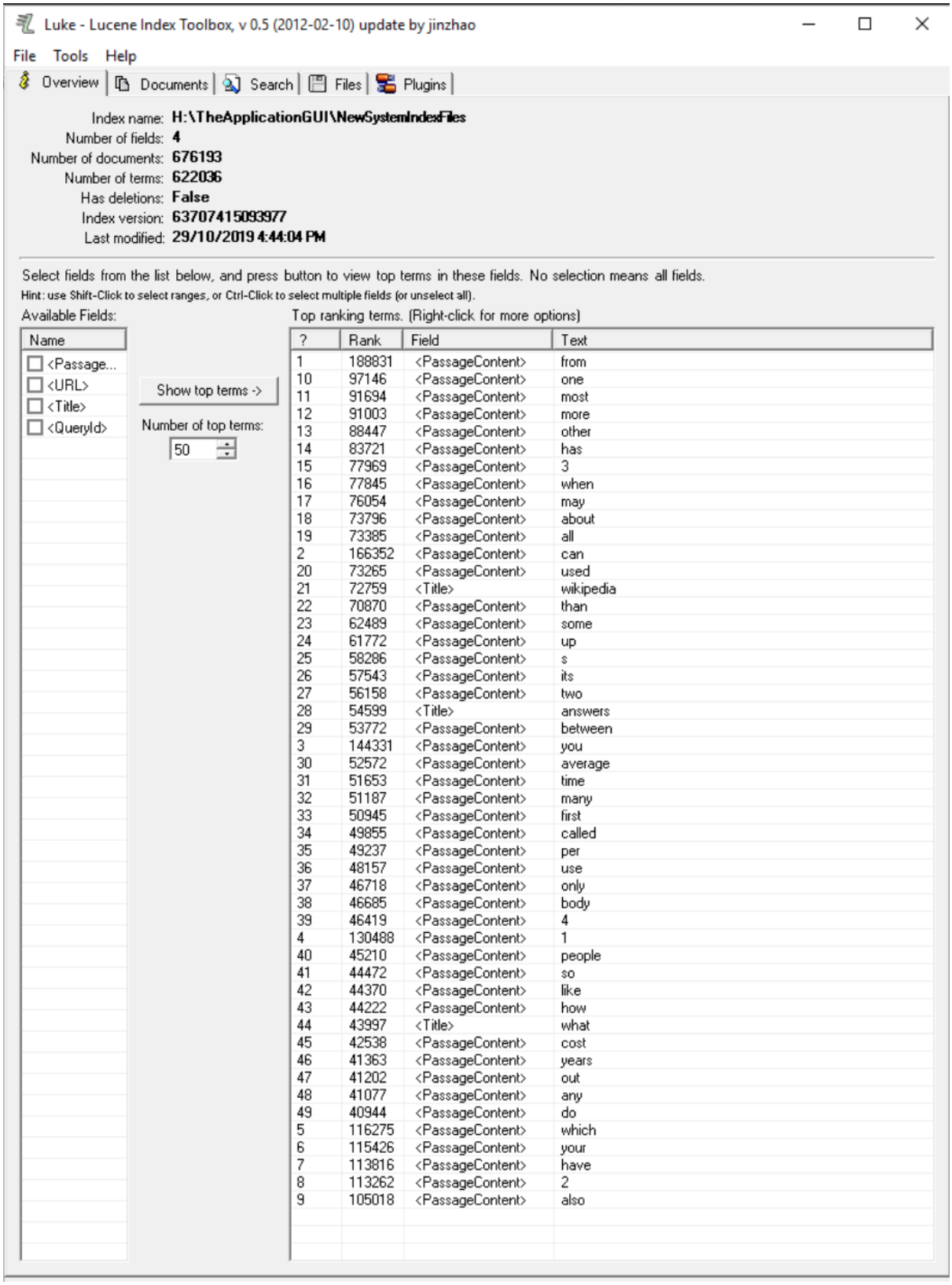


Figure 1. Index overview.

1.DESIGN

1.1.1 Overall Design: Classes

1. **Passage**: for json deserialization, holds the data types for the passage text, passage ID, URL and is selected Boolean
2. **RootObject**: for json deserialization, holds multiple Passage class objects, as well as the data types for the query, query_type and answers
3. **NewSystem**: parent class of whole system
4. **NewSimilarity**: contains the changes to the scoring function

1.1.2 Overall Design: Major Methods

AskForBoost():

- get boost from user for fields or terms before indexing or searching respectively

IndexText():

- indexes one passage, URL, title, query_ID set

IndexCycle():

- calls AskForBoost for field boosts, repeatedly calls IndexText for each passage, URL, title, query_ID set

SearchIndex():

- calls ExpandQuery() to get search parameters, then searches, returns the results to another function

DisplayResults():

- displays ordered list of results containing the ranking, title, document ID, score and the URL

Figure 2. Example of search result.

LoadJson():

- deserializes json source file

GenerateTitle():

- attempts to parse URL into a title for each passage

SaveTrecResults():

- saves a set of results in format suitable for trec_eval, to a new file or appended to an existing file

GenerateTrecResult():

- searches for one of the supplied queries, converts the results into the trec_eval format

SelectIndividualTrecResults():

- asks user to pick individual supplied queries, repeats in loop till user chooses to stop, then calls SaveTrecResults() with the set of results

SelectRangeTrecResults():

- asks user to pick start and finish indices for supplied questions, then calls SearchIndex() for each of them, then calls SaveTrecResults() with the set of results

GenerateQrel():

- creates a file containing the correct answers to the supplied queries in the format required for trec_eval

ManualNaturalLanguageQuery():

- asks user to input text question, searches, displays results

AskIfQueryExpanding():

- asks user whether they want to use the query expansion capabilities with this search

ExpandQuery():

- if a user wants to expand query, asks user to use synonyms for various POS, boost terms, if any terms must be included or excluded from results

Demonstration():

- uses an example question to show the DisplayResults() output
- then asks user for parameters to query expand the same question to demonstrate the different in results

SetupQuestions():

- asks user for hyper-parameters

UseQASystem():

- essentially the home page once the system has been built

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
KUT EduQuiz is ready for you to use
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        >>Enter 1 to see a demonstration of the answering system using supplied queries.
        >>Enter 2 to manually enter queries.
        >>Enter 3 to select a range of supplied queries to generate the input to Information Retrieval evaluation program trec_eval.
        >>Enter 4 to select individual supplied queries as input to Information Retrieval evaluation program trec_eval.
        Or, enter exit to finish.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Figure 3. Home screen.

LoadWordNet():

- creates the WordNet data file to lookup synonyms when expanding queries

```
Loading WordNet database...
Time to produce WordNet database: 03.656 seconds
Load completed.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Figure 4. Loading WordNet

Run():

- main thread of program

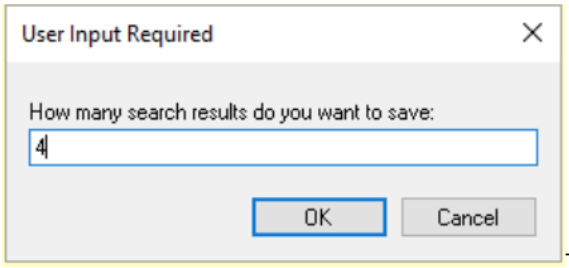


Figure 5. Implementation of InputBox(). The text and input-type parameters of InputBox() can be altered.

InputBox():

- method to get inputs from user

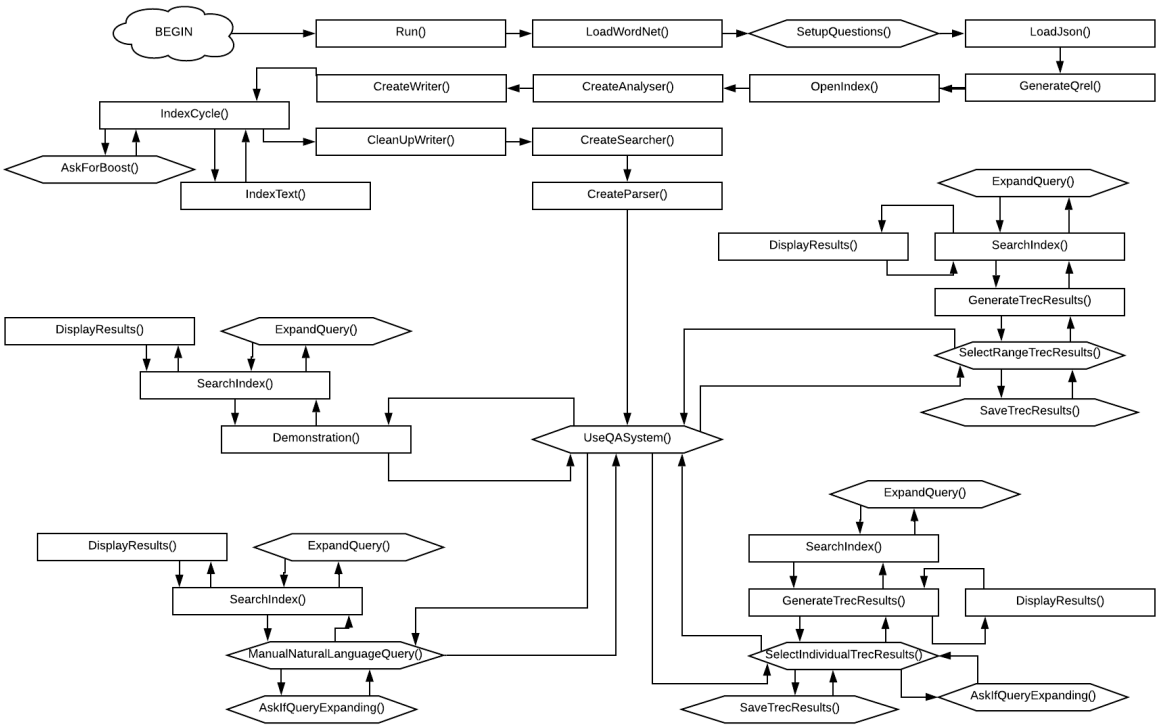


Figure 6. Control-flow overview. Hexagonal denotes that user input is required to complete this method, rectangular denotes back-end methods.

The three horizontal rows at the top of Figure 6 contain the initialization and preparation steps. They build the qrel files, the WordNet data base, and process the json source file into an index. The user needs to input the data file location and name, any field boosts, names for the trec results outputs, and how many results to show on the screen.

After the preparations have been completed, the UseQASystem method is called. It acts as the home screen for the system. It offers the user four options.

Option one is a demonstration which uses an automated, natural language query "are polar bears black". It prints the results to the screen as an example of the system output. Then, user is asked to input any of the query expansion options:

- boost a query term
- only return results with a particular query term
- exclude results with a particular term
- use WordNet to add synonyms (nouns, verbs, adjectives, or adverbs)

The second option from the home screen is to enter a natural language query. The user types in a query, then they're asked the query expansion questions as outlined above. After the search is completed and the results are printed to the screen, the user is then asked if they would like to see the full passage from one of the results (see Figure 7). Finally, the user is asked if they want to save the results to file, and if so, how many results to save.

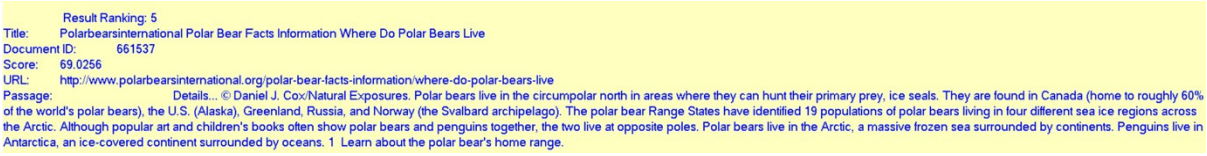

Result Ranking: 5
Title: Polarbearsinternational Polar Bear Facts Information Where Do Polar Bears Live
Document ID: 661537
Score: 69.0256
URL: http://www.polarbearsinternational.org/polar-bear-facts-information/where-do-polar-bears-live
Passage: Details... © Daniel J. Cox/Natural Exposures. Polar bears live in the circumpolar north in areas where they can hunt their primary prey, ice seals. They are found in Canada (home to roughly 60% of the world's polar bears), the U.S. (Alaska), Greenland, Russia, and Norway (the Svalbard archipelago). The polar bear Range States have identified 19 populations of polar bears living in four different sea ice regions across the Arctic. Although popular art and children's books often show polar bears and penguins together, the two live at opposite poles. Polar bears live in the Arctic, a massive frozen sea surrounded by continents. Penguins live in Antarctica, an ice-covered continent surrounded by oceans. 1 Learn about the polar bear's home range.

Figure 7. The full passage from the fifth ranked result for search "are polar bears black".

The last two options are ways to generate search results suitable for trec analysis. With option three from the home screen, the user can select a range of supplied queries. The user is asked for starting and finishing indices. The search is then completed, printing out results to the screen for each query. The user is then asked to accept the supplied filename enter a new one for the results to be saved to on the computer. If the file already exists, the results will be appended to it.

The final option from the 'homepage' is to manually select supplied queries to generate the trec results. Instead of supplying the first and last index for a range of queries, the user is asked to input a single index. It is searched for, the results are appended to an internal variable, and the user is asked to input another index. When the user is finished, they're asked to accept the supplied filename or supply a new one. As before, if the file already exists, the results will be appended to it.

After using any of these four methods to interact with the Lucene system, the user returns the UseQASystem screen to ask what to do next, or to enter "exit" to shut the program down.

1.2 Indexing Strategy

The indexing begins with the OpenIndex() method, it initializes the Lucene.Net.Store.Directory object. Then, CreateAnalyser() and CreateWriter()

initialize the Lucene.Net.Analysis.StandardAnalyzer and Lucene.Net.Index.IndexWriter objects.

Next, IndexCycle() performs the management of the indexation. It asks the user if they want to boost any of the fields. It then calls the IndexText() method and sends it one combination of passage, title, passage ID, and URL. IndexText() turns the strings into Field objects. It stores each of them, but only indexes the passage and the title. It does not store term vectors for the title as it is such a small field anyway. It does store term vectors for the passage field as some of these are quite long.

1.3 Source Document Structure

The URL, passage text, algorithmically generated title and passage ID are stored in the index. The passage and title are indexed. The URL was not indexed as the information was contained in the title, so indexing the URL too would simply slow the system down. Term vectors were generated for the passages, but not for the titles as they're so short anyway, it seemed an unnecessary overhead.

1.4 Document Errors

```
In[20]: print("Query: ", data[7]['query'], "\nQuery ID: ", data[7]['query_id'])
   ...: for key in data[7]['passages']:
   ...:     print(key)
   ...:
Query:  what does a metabolic acidosis need to reverse the condition
Query ID:  19706
{'is_selected': 0, 'url': 'http://emedicine.medscape.com/article/242975-overview', 'passage_text': 'Background. Metabolic acidosis is a clinical disturba
{'is_selected': 0, 'url': 'http://www.healthline.com/health/acidosis', 'passage_text': '1 Both diarrhea and vomiting can cause this type of acidosis. 2
{'is_selected': 0, 'url': 'http://medical-dictionary.thefreedictionary.com/metabolic+acidosis', 'passage_text': "Metabolic acidosis, as a disruption of t
{'is_selected': 0, 'url': 'http://www.healthgrades.com/conditions/acidosis', 'passage_text': 'Acidosis is a serious metabolic imbalance in which there is
{'is_selected': 0, 'url': 'http://emedicine.medscape.com/article/242975-overview', 'passage_text': 'A normal serum HCO 3 - level does not rule out the pr
{'is_selected': 0, 'url': 'http://medical-dictionary.thefreedictionary.com/metabolic+acidosis', 'passage_text': 'acidosis. 1. the accumulation of acid an
{'is_selected': 0, 'url': 'https://en.wikipedia.org/wiki/Acidosis', 'passage_text': 'For acidosis referring to acidity of the urine, see renal tubular ac
{'is_selected': 0, 'url': 'http://www.vibranthealthandwealth.com/vibrance/articles/acid.html', 'passage_text': 'It is defined as excessive blood acidity
{'is_selected': 0, 'url': 'http://www.healthline.com/health/acidosis', 'passage_text': 'Acidosis occurs when your kidneys and lungs can't keep your body'
```

Figure 8. Query with no passage selected as correct.

Some queries did not have a passage selected as the answer (see Figure 8). These were kept in the system because the assignment instructions explicitly state that the passage which are not selected as query answers were to be retained (page 10, "There are two sets...).

No other document errors were found during the course of this software development, nor in the related literature review (Baja, 2016; Wadhwa, 2018).

1.5 Search Strategy

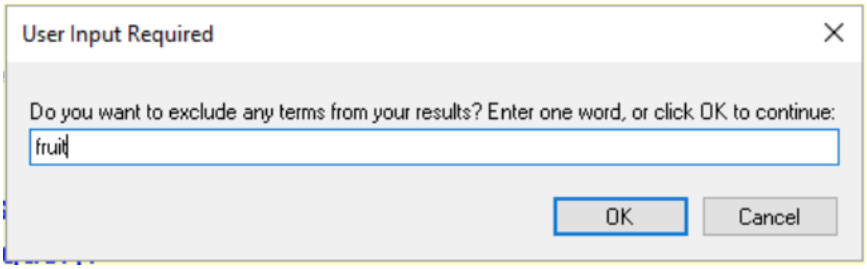


Figure 9. To help to distinguish between sporting bats and animal bats, all results containing the word fruit are excluded.

```
        Original query: saw baseball bats
No boost for word saw
No boost for word baseball
No boost for word bats

        Parsed query: (PassageContent:saw Title:saw) (PassageContent:"power saw" Title:"power saw") (PassageContent:saw Title:saw)
(PassageContent:"sawing machine" Title:"sawing machine") (PassageContent:proverb Title:proverb) (PassageContent:adage Title:adage)
(PassageContent:byword Title:byword) (PassageContent:baseball Title:baseball) (PassageContent:bats Title:bats)
```

Figure 10. The search query "saw baseball bats" is expanded with nouns for saw.

A query is generated, either by instance of InputBox() is for the user to type in their natural language query, or a supplied query is generated by its index. This query is passed to SearchIndex(). It calls ExpandQuery() for the query expansion. The user is asked if they want to boost the first word in the query, and then whether they want to use WordNet synonym expansion. If so, they're separately asked if they want to include nouns, verbs, adverbs and adjectives. The synonyms are copied from the WordNet data base and added to the query string. They are then asked if they want to only receive results containing this term. This process is repeated for each term in the original query. At the end, they're asked if they want to exclude results containing a particular term (see Figure 9). The expanded query is returned to the SearchIndex() method.

```
            Result Ranking: 1
Title:      Bhg Recipes Healthy Dinner Healthy Meat Substitutes
Document ID:        484389
Score:      30.51033
URL:        http://www.bhg.com/recipes/healthy/dinner/healthy-meat-substitutes/
There are 3 fragments of text which match the query:
            Matching fragment 1
<B>Healthy</B> <B>Meat</B> Substitutes. If you're not ready to go vegetarian but you're trying to cut calories
            Matching fragment 2
 from your <B>meat</B> intake, try these <B>healthy</B> <B>meat</B> swaps. With an eye on calories, fat, and protein, we
            Matching fragment 3
 substitutions chart! <B>Healthy</B> Substitutes for <B>Meat</B> Ground Beef: For 1 pound ground beef, substitute 1 pound
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Figure 11. The first three matching fragments from the top ranked result of query "is meat healthy" on the new system. (NB they are supposed to be bolded, but I couldn't get the HTML to work in C#)

```
Title of result at rank 1: Bear Website Bear Pages Black Bear Basic Bear Facts 101 What Is A Spirit Bear
Score25.41589 = (MATCH) sum of:
  0.7439107 = (MATCH) weight(Text:are in 643277), product of:
    0.1660421 = queryWeight(Text:are), product of:
      2.003631 = idf(docFreq=247855, maxDocs=676193)
      0.0828706 = queryNorm
    4.480254 = (MATCH) fieldWeight(Text:are in 643277), product of:
      2.236068 = tf(termFreq(Text:are)=5)
      2.003631 = idf(docFreq=247855, maxDocs=676193)
      1 = fieldNorm(field=Text, doc=643277)
  4.42174 = (MATCH) weight(Text:polar in 643277), product of:
    0.6053365 = queryWeight(Text:polar), product of:
      7.304598 = idf(docFreq=1235, maxDocs=676193)
      0.0828706 = queryNorm
    7.304598 = (MATCH) fieldWeight(Text:polar in 643277), product of:
      1 = tf(termFreq(Text:polar)=1)
      7.304598 = idf(docFreq=1235, maxDocs=676193)
      1 = fieldNorm(field=Text, doc=643277)
  15.30451 = (MATCH) weight(Text:bears in 643277), product of:
    0.6333006 = queryWeight(Text:bears), product of:
      7.642042 = idf(docFreq=881, maxDocs=676193)
      0.0828706 = queryNorm
    24.16626 = (MATCH) fieldWeight(Text:bears in 643277), product of:
      3.162278 = tf(termFreq(Text:bears)=10)
      7.642042 = idf(docFreq=881, maxDocs=676193)
      1 = fieldNorm(field=Text, doc=643277)
  4.945738 = (MATCH) weight(Text:black in 643277), product of:
    0.4526899 = queryWeight(Text:black), product of:
      5.462611 = idf(docFreq=7797, maxDocs=676193)
      0.0828706 = queryNorm
    10.92522 = (MATCH) fieldWeight(Text:black in 643277), product of:
      2 = tf(termFreq(Text:black)=4)
      5.462611 = idf(docFreq=7797, maxDocs=676193)
      1 = fieldNorm(field=Text, doc=643277)
```

Figure 12. Score details of the top ranked query for "are polar bears black" from the baseline system.

The query is parsed with the QueryParser object, then searched with the IndexSearcher object. It returns a TopDocs object. This is a list of Lucene.Net.Documents.Document objects. Each contains the passage, title, URL, passage ID, the set of matching fragments (see Figure 11), the score of the query and result, and an explanation of score.

The TopDocs object is passed to DisplayResults(). This method prints out the ranking, title, passage ID, score, URL and up to five fragments of the top results (the number of results was selected by the user during the program setup). Control passes back to SearchIndex(), which returns the TopDocs object and the (expanded) query object to the method which called it, as some methods have follow up uses for the results. One example is the option for a user to print out the entirety of a passage (see Figure 7).

```
            Original query: cats

            Parsed query: PassageContent:cats Title:cats

            Number of results is 2325
Time to search: 00.784 seconds


             Result Ranking: 1
Title:      Cat Breed Info Small Cat Breeds
Document ID:        462324
Score:      38.88665
38.88665 = (MATCH) sum of:
  38.88665 = (MATCH) weight(PassageContent:cats in 462323), product of:
    0.6478232 = queryWeight(PassageContent:cats), product of:
      6.775494 = idf(docFreq=2097, maxDocs=676193)
      0.09561269 = queryNorm
    60.02664 = (MATCH) fieldWeight(PassageContent:cats in 462323), product of:
      81 = tf(termFreq(PassageContent:cats)=9)
      6.775494 = idf(docFreq=2097, maxDocs=676193)
      0.109375 = fieldNorm(field=PassageContent, doc=462323)

URL:        http://www.cat-breed-info.com/small-cat-breeds.html
```

Figure 13. Explanation of score for query "cats".

The scoring function was altered in two ways. The term frequency exponent was changed from square root to square, and the query document coordination was doubled. Explanations of both these changes are in 3.1 Changes to Baseline.

1.6 User Interface

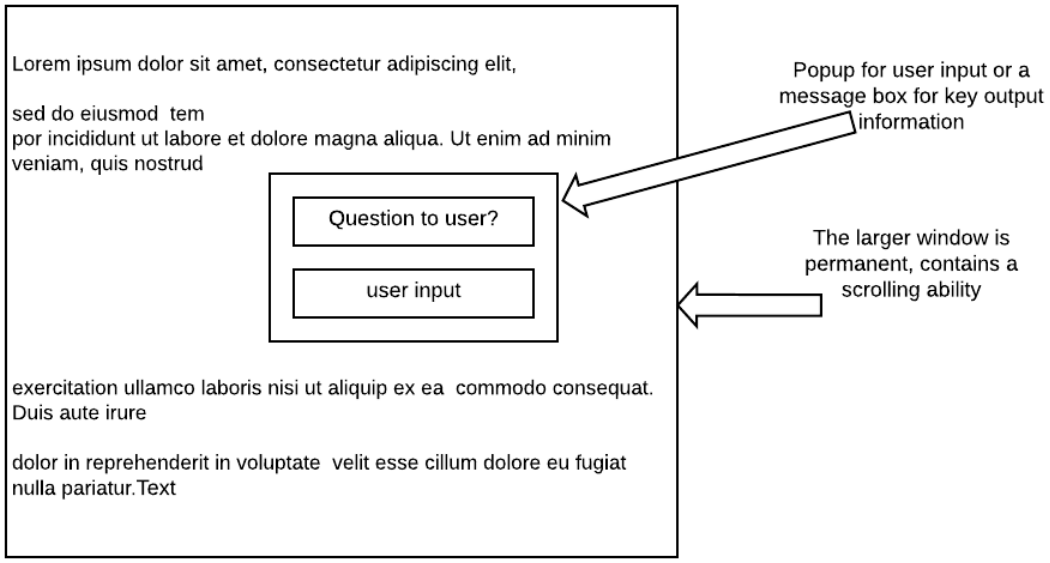


Figure 14. User interface

An unusual approach was taken to the interface. Instead of a window with multiple options within it (like Manuel's example in the lecture), an interface was designed so that there was a single point of focus for the user.

At any moment, they are either reading the main output in the larger window, reading key information in a MessageBox() popup (e.g. how much boost was selected, the number of results saved to a text file), entering text information into the InputBox() popup, or selecting a directory from a standard drop down folder hierarchy used during the setup (see Figure 15).

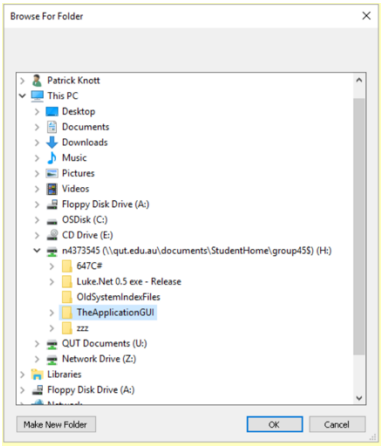
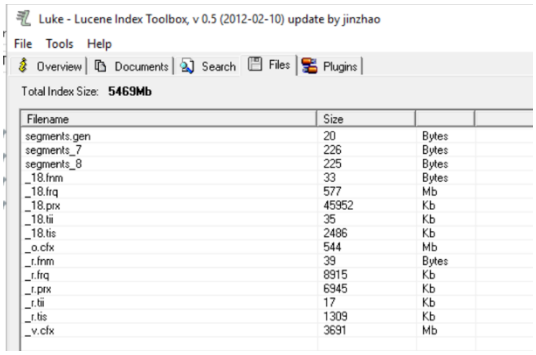


Figure 15. Select folder containing json source file

2. CHANGES TO BASELINE

2.1 List of Changes



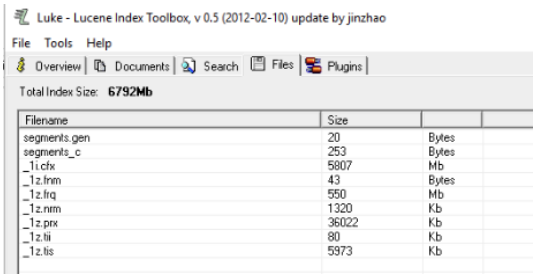


Figure 16 and 17. The size on disk of the Baseline system and The Application, respectively

Term vectors and document norms were added to the index. Increasing the size of the index from nearly 5.5 GB to nearly 6.8 GB would not overburden the KUT IT resources (see Figures 16, and 17). But, the reduction in searching from not calculating the lengths of the passages, and being able to move directly between term vectors would make an important impact on search time. Even after these efforts to quicken search times, when a query used a lot of synonym expansion, it could still take ten to fifteen seconds (see Figure 18).

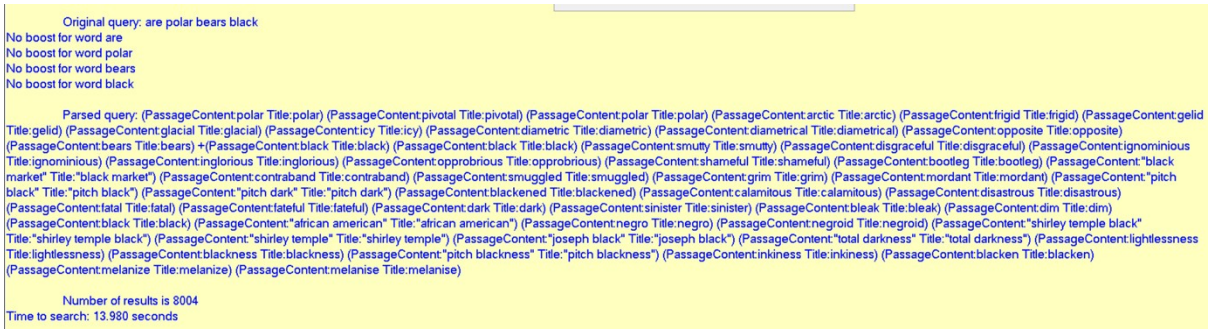


Figure 18. Runtime of an expanded query of nearly fourteen seconds.

The title field was indexed out of a belief that it could be used to extract key concepts related to passage. For example, in the search immediately above, it was thought that if 'bear' was in the title (from the URL) of a page, it would be a more significant match than if the word was in the content of the passage.

The analyzer was changed from Simple to Standard. It would remove stopwords, but keep some kinds of non-standard text (e.g. email). It was also chosen because it does not stem. It was thought that if WordNet synonyms were to be used, also stemming words would be inefficient.

A method was created to give the user the option to boost fields during indexing and query terms before searching.

In a similar vein, the ability to restrict results to those containing a particular word, or to those not containing a particular word.

The final query processing option was to add synonyms from WordNet. There were four part-of-speech options.

When returning results, the matching term and the text around it were printed out so the user could see the context of the match, not just which passage the match was from. It was supposed to be bolded, but I couldn't get that to work so it just prints out the HTML.

Finally, the parser was changed to a MultiFieldQueryParser object because there was now more than one field.

2.2 Default Similarity

```
public class NewSimilarity : DefaultSimilarity
{
    public override float Tf(float freq)
    {
        return (float)Math.Pow(freq, 2);
    }
    public override float Coord(int overlap, int maxOverlap)
    {
        return 2 * overlap / (float)maxOverlap;
    }

}
```

Figure 19. Altered scoring function.

The term frequency calculation was changed from square root to squared. The literature seemed to be binary, linear, log damped, or use a fractional exponent (Domeniconi, 2015; Zhu, 2011). So, an alternative was used for exploratory purposes. It was not a success (more detail in 4.1 System Evaluation).

During the research, it was found that scientific articles tend to focus on a specific subject (Ramos, 2003). The search index is intended for university students, so the scoring was altered to encourage more scientific results. The query document coordinating factor was doubled in an attempt to counteract this tendency of academic documents to have a smaller range of terms.

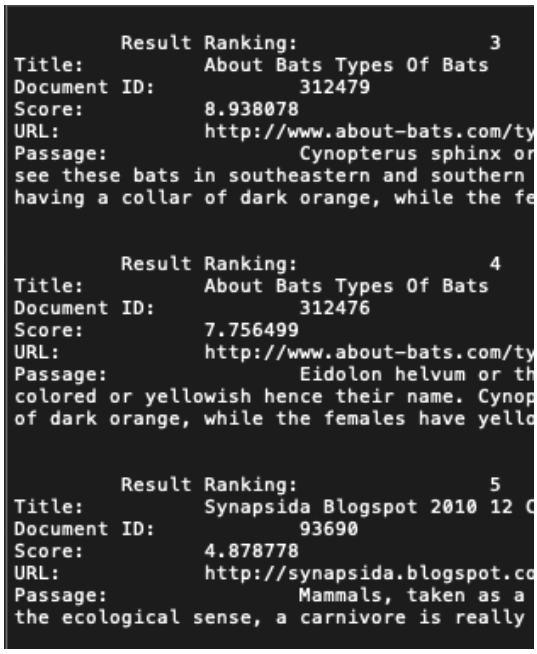
2.3 Detail of Two Changes

The ability to exclude words from results is a corollary to synonym expansion. With WordNet expansion to increase the recall, there needed to be an effort to reduce polysemy. Otherwise, the precision would shrink away. People tend not to look through a long list of results. So, if there was only synonym expansion there would be a risk that users would lose interest and stop using the system. There needs to be a way to refine the results. Many people are used to an iterative process of using web search engines. Increase the breadth of the query, look through some results, then use some of what they see to sharpen the query. It seemed important to include an option for a similar methodology for the KUT Question answering system. The alteration to the code was minor. In the ExpandQuery method, the user is asked to input a word to exclude from results. If they input a word, it is prepended with a negative sign (-), and then appended to the query string. This alteration worked well.

Another change from the Baseline system was to add an indexed field for the title and the option to boost it before indexation. It was believed that this would be a way to extract key words to represent the passage. The title is just the URL, with some HTTP nomenclature removed, and split at punctuation. So, it seemed that any words which remained would likely be indicative of the related passage. When coupled with a boost for the field, it was expected to match query terms with passages with a focus on that word. The code alterations for this were more involved than for word exclusion. The IndexText method already stored the title, so it only required a change from NO to YES for the Index parameter, and then the parameters for choice of analyzer and term vectors. The AskForBoost method was created to get user input on boosts for the indexation and for queries. It was called twice from the IndexCycle method to get the boosts for the two indexed fields, title and passage. For each call to IndexText, the field boost was passed in.

2.4 Improved Query Processing

Describe your approach of implementing the Task 7, "Improved Query Processing" options. You could describe your changes with a short description and include screen shots of the results obtained in each step.



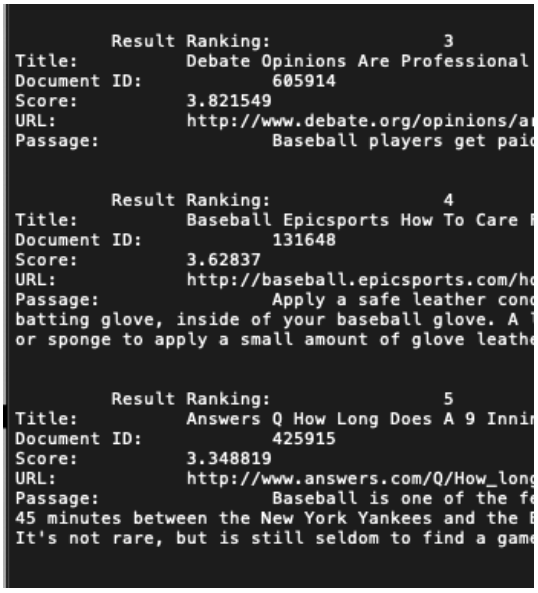


Figure 20 and 21. Search results for the phrase "saw baseball bats". In the first set of results, the third to fifth results are about "bats" the animal. In the second set of results, the word baseball was given a boost of 5 which changed the results to the required information away from the animal .

The improved query processing involved five extra options for the user. Boosting terms gave the option to add a multiplicative to the score for a particular word across all indexed fields. Excluding terms can reduce word ambiguity by reducing polysemous results (see Figure 22). Synonym expansion can broaden the results to include a concept rather than to just one of the words related to it. Terms which must be included can counteract the effect of other terms which tend to be repeated in a passage (see Figure 20)

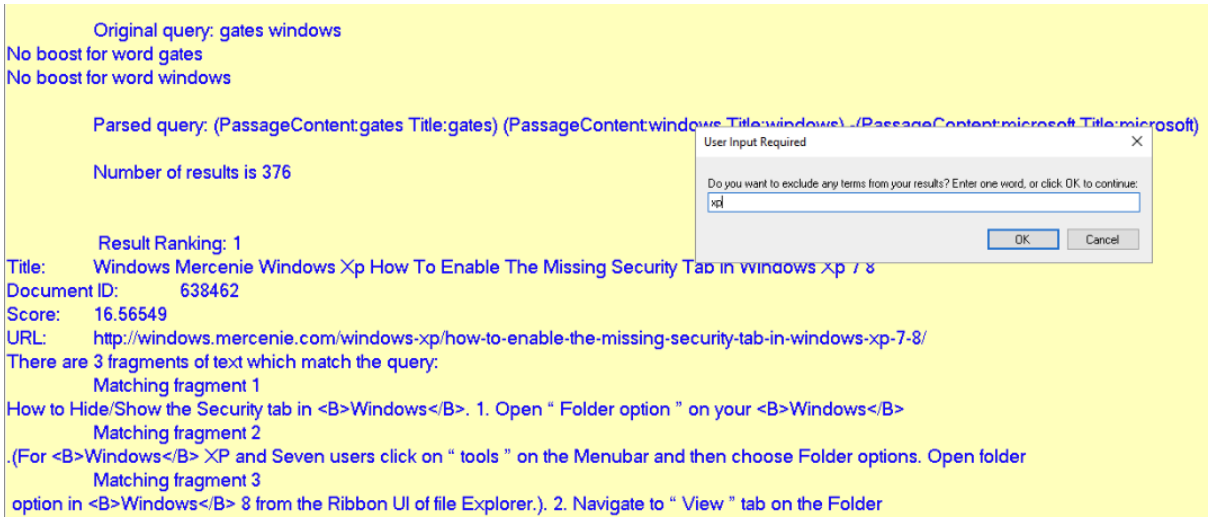


Figure 22. Results for "gates windows" returns Windows XP, not construction supplies.

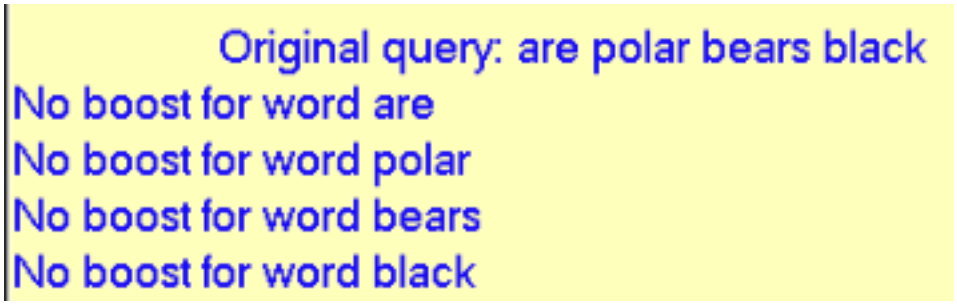


Figure 23. Boost decisions

Figure 24. Searching for "big" and "apple".

Figure 25. Searching for "big" and "apple" "big apple"

3. System Evaluation

Efficiency metrics

The index for The Application is about 6.8 GB (see Figure 16)

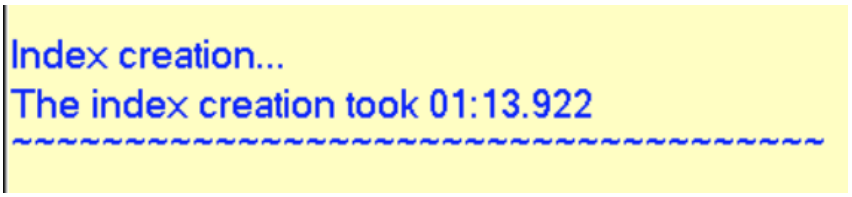


Figure 26. Times to build index for The Application.

```
Original query: are polar bears black

Parsed query: (PassageContent:polar Title:polar) (PassageContent:bears Title:bears) (PassageContent:black Title:black)

Number of results is 9837
Time to search: 00.838 seconds
```

Figure 27. Time to search "are polar bears black" in The Application.

Effectiveness Metrics:

These metrics were generated using the first 101 queries from the database. The Precision @ 10 is 0.1752 (using the linked passages).

Using the selected passages, the Mean Average Precision is 0.1984, and the Mean Reciprocal Rank is 0.190.

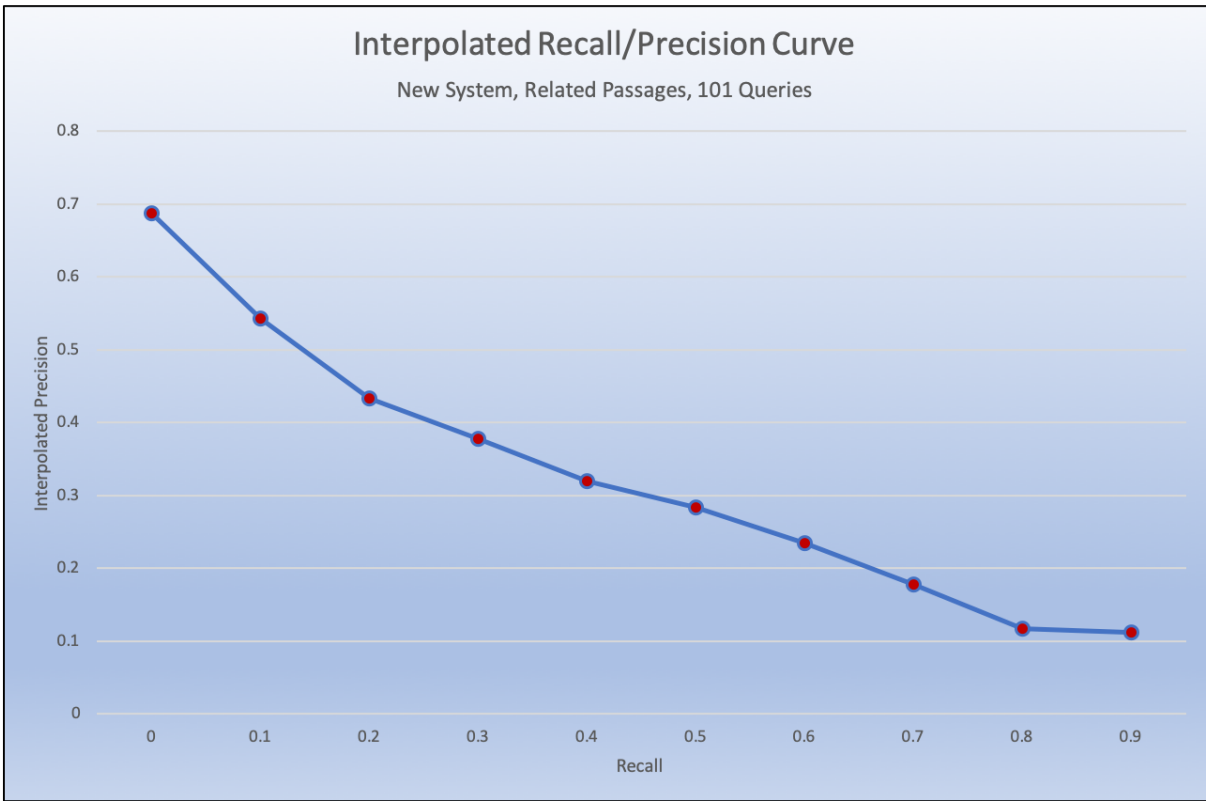


Figure 28. Interpolated Recall/Precision Curve. The Application, using all linked passages.

4. Comparison to Baseline

Efficiency metrics

The index for The Application is about 25% larger than for the Baseline System (see Figures 16, and 17). The new system takes six seconds longer to load(see Figures 26 and 29). The search time has increased by a third (see Figures 27 and 30). The differences in all efficiency metrics are small (though inconsistent, 9% longer to index but the index is 25% larger).

Figure 29. Time to build the index for the Baseline.

Figure 30. Time to search "are polar bears black" in Baseline. NB it returns all results containing the word "are", which is why the number of results is so large

Effectiveness metrics

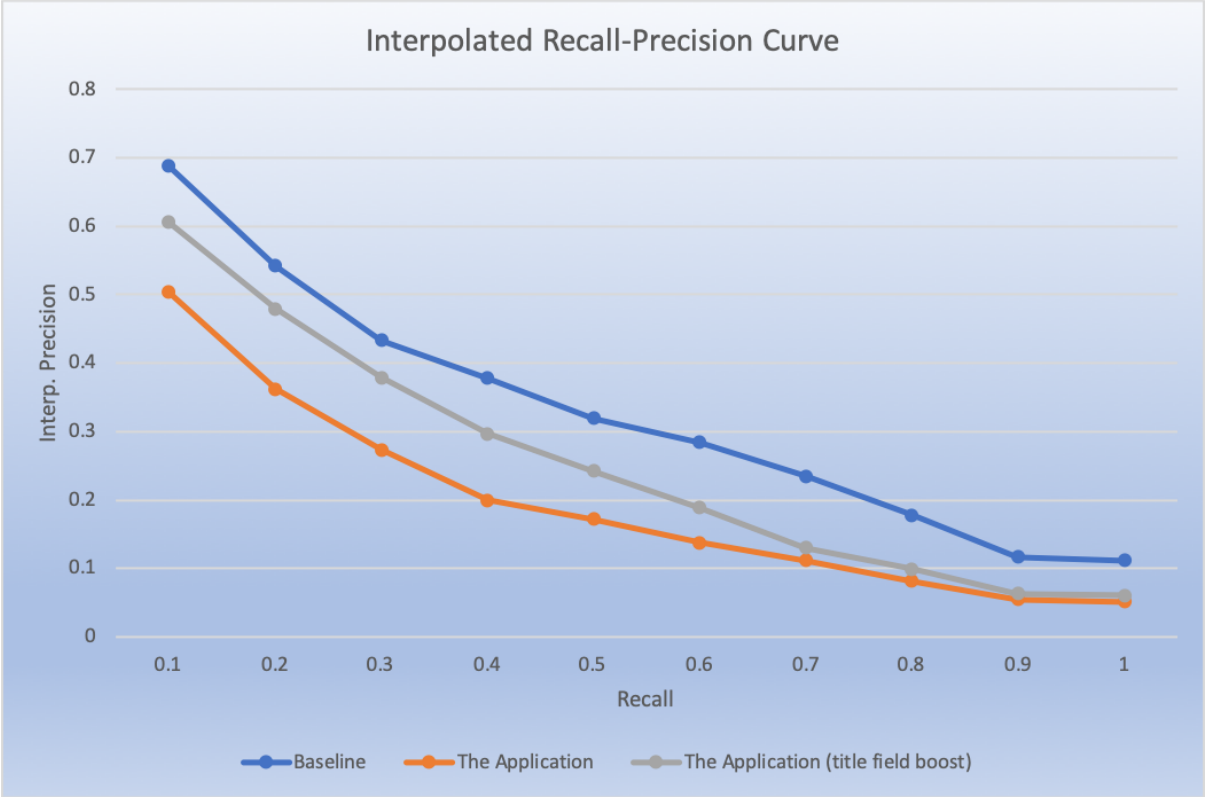


Figure 31. Interpolated Recall/Precision Curve. Includes the Baseline, The Application and The Application with a boost of five for the title field.

Table 1. Effectiveness metric comparison

	Baseline	The Application	The Application (title boosted)
Precision@10	0.2931	0.1752	0.2337
MAP	0.2333	0.1734	0.2198
MRR	0.2513	0.19	0.242

By these metrics, the new system was a failure. It may be in part because the titles were too inaccurate to be useful as an indexed field. It is surprising that removing stop words (the analyzer changed from Simple to Standard) did not improve precision. For Mean Average Precision and Mean Reciprocal Rank the difference between the Baseline and The Application with a title field boost of five was small, about 1% less for both.

I don't think this means that The Application is a failure. These results do not include the effect of synonym expansion, word inclusion or word exclusion. The changes to the scoring function were likely unnecessary. Perhaps this is the reason that term frequency is either linear or damped in the literature. With the DefaultSimilarity changes removed, and the query pre-processing available, The Application would be an improvement on the Baseline when used by KUT students (though not for trec competitions).

5. USER GUIDE

Indexing

When the system loads, the user is asked to select two folders from a folder hierarchy drop down menu. The first contains the json source file, the second will be used as an output folder. Then the user is asked for the name of the json file. The system will deserialize it, then ask if the user wants to boost the fields. Then it builds the index and notifies the user when completed, including the time it took.

Search and Retrieve Results

The home screen offers the user the option to enter natural language queries. They'll be asked if they want to query expand, and then to enter the query. If query expanding, for each term they're asked if they want to boost it; to add synonyms for nouns, verbs, adverbs and adjectives (this option is only shown to the user if there are synonyms for the term); and to only receive results containing the word. After these questions are answered for each query term (not including the new synonyms), the user is asked if they want to exclude any words from the results.

Then the results print out on screen. The user is asked if they want to see the full passage from any of the results (see Figure 7). After which, the user is asked if they want to save the results to dis. If they do want to save, they are asked how many results to include in the text file.

Save Results

There are two options for generating and saving trec results. If the user wants to select a continuous range of results, they're asked to input the first and last index of the range, and the system will search for each of the respective supplied queries. Or if the user wants to enter individual supplied queries, they're asked to input individual indices, which are searched for one at a time.

Whichever query selection method is used, the saving of the file proceeds the same way. The user is asked to enter a filename. If the file exists, the new queries are appended to it. Or, if the file does not exist, it is created. This happens in the output folder specified by the user during the Indexing process outlined above.

REFERENCES

Bajaj, P., Campos, D., Craswell, N., Deng, L., Gao, J., Liu, X., ... & Rosenberg, M. (2016). MS MARCO: *A human generated MAchine Reading COmprehension dataset. arXiv preprint arXiv:1611.09268.*

Domeniconi, G., Moro, G., Pasolini, R., & Sartori, C. (2015, July). *A Study on Term Weighting for Text Categorization: A Novel Supervised Variant of tf. idf.* In DATA (pp. 26-37).

Ramos, J. (2003, December). *Using tf-idf to determine word relevance in document queries.* In Proceedings of the first instructional conference on machine learning (Vol. 242, pp. 133-142).

Wadhwa, S., Chandu, K. R., & Nyberg, E. (2018). *Comparative Analysis of Neural QA models on SQuAD*. arXiv preprint arXiv:1806.06972.

Zhu, D., & Xiao, J. (2011, October). *R-tfidf, a Variety of tf-idf Term Weighting Strategy in Document Categorization*. In 2011 Seventh International Conference on Semantics, Knowledge and Grids (pp. 83-90). IEEE.