DD2476 Search Engines and Information Retrieval Systems

Assignment 3: Relevance Feedback and Language Models

Johan Boye, Jussi Karlgren, Hedvig Kjellström

The purpose of Laboration 3 is to learn about ways to get more powerful representations of query and documents. You will learn 1) how to use relevance feedback to improve the query representation; 2) how to use language models to improve the document and query representation; and 3) how to speed up the search in various ways.

The recommended reading for Assignment 3 is that of Lectures 2, 4, 6, and 7.

Assignment 3 is graded, with the requirements for different grades listed below. In the beginning of the oral review session, the assistant will ask you what grade you aim for, and ask questions related to that grade. All the tasks have to be presented at the same review session – you can not complete the assignment with additional tasks after it has been examined and given a grade. ***Come prepared to the review session!*** *The review will take 10 minutes or less, so have all papers in order.*

E: *Completed Tasks 3.1, 3.2 with some mistakes that could be corrected at the review session.*
D: *Completed Tasks 3.1, 3.2 without mistakes.*
C: *E + Completed Task 3.3 without mistakes.*
B: *C + Completed Task 3.4 without mistakes.*
A: *B + Completed Task 3.5 without mistakes.*

These grades are valid for review April 1, 2014. See the web pages www.kth.se/social/course/DD2476, ir14 - Computer assignments in the menu, for grading of delayed assignments.

Assignment 3 is intended to take around 50h to complete.

Computing Framework

For Assignment 3, you will be further developing your code from Task 2.2 or 2.7. Make sure that you **correct all errors in the code from Assignment 2**, that were pointed out at the examination, so that the ranked retrieval works without errors.

Task 3.1: Relevance Feedback

The first task is to **implement relevance feedback in your search engine**. You will need to add code to the method `relevanceFeedback` in `Query`, so that when this method is called with the `queryType` parameter set to `Index.RANKED_QUERY`, the system should expand the query using the Rocchio algorithm with parameter γ = 0, i.e., by multiplying the weights of the the original query terms with α, and adding query terms from the documents marked as relevant, setting the weight of these terms to β*<weight of term in doc>.

Note that you have to normalize both the query weights and the document weights, by dividing the document/query vector with the length of the same vector. This has to be done prior to the Roccio computation.

The query and the document vectors should be represented in the same way, with either tf or tf-idf weights. The length of the query and the documents should also be represented in the same way, as #terms, or as the Euclidean length of the document/query vector. The results below were generated with tf weights (i.e., no idf weighting of query terms) and length(doc) = #terms in doc.

Many terms will reoccur in different documents. Make also sure not to add terms twice to the query, but instead add to the weight of the existing term instance.

Play with different values of α and β. Make sure that your tf-idf score computation takes the query term weights into account.

When your implementation is ready, compile and run the search engine, indexing the 10 data sets `svwiki/files/1000`, `svwiki/files/2000`, …, `svwiki/files/10000`. Select the "Ranked retrieval" option in the "Search Options" menu, and try the search queries

mellan olika alternativ

den tyska huvudstaden

tillvarons yttersta grunder

which are the same as you applied to this dataset in Assignment 2. For each of the three queries, select two documents in the top ten list of retrieved documents, that you think are the most relevant. Mark these using the buttons at the bottom of the GUI, and press ”New search”.

> A fast way of seeing what articles are about, is to look in the file `articleTitles.txt`, as:

```
> egrep "^365;"
  /info/DD2476/ir14/lab/svwiki_links/articleTitles.txt
365;Danmark
```

The Rocchio algorithm will now be applied using the original query and these two documents.

What happens to the two documents that you selected?

What are the characteristics of the other documents in the new top ten list - what are they about? Are there any new ones that were not among the top ten before?

Play with the weights α and β: *How is the relevance feedback process affected by α and β?*

Ponder these questions: *Why is the search after feedback slower? Why is the number of returned documents larger? Why are there more highly ranked short documents?*

At the review

To pass Task 2.2, you should be able to start the search engine with a dataset specified by the teacher, and perform a search in ranked retrieval mode with a query specified by the teacher, and then perform relevance feedback, marking a set of documents specified by the teacher.

You should be able to explain the Rocchio algorithm using pen and paper, using the concepts and illustrations in the book and in the slides of Lecture 7. You should also be able to discuss the questions in italics above, and to explain all parts of the code that you edited.

Task 3.2: Designing an evaluation

In Tasks 1.4 and 2.3, you have learned how to evaluate a search engine by measuring precision and recall for a representative set of queries (the 10 queries listed in these tasks) and a representative data set (a subset of Swedish Wikipedia).

Now you will use this knowledge to design an experiment to evaluate how the performance of the search engine improves with the relevance feedback you introduced in Task 3.1. The experiment should use the same test queries and test data set as in Tasks 1.4 and 2.3. The concepts of *precision-recall curve* and *precision at 10* should be used.

NOTE: You do not have to carry out the experiment, but you must describe it in text, in such a degree of detail that another student in the course could carry out the experiment with this text as guidance.

At the review

To pass Task 3.1, you should show your written account of the experiment design, and be able to explain all parts of it to the teacher.

Task 3.3: Speeding Up the Search Engine (C or higher)

Implement **at least one of the following speedups** of the search engine:

1) Index Elimination – use only high-idf terms at query-time (remember to recompute word positions in documents after removing low-idf words)

2) Index Elimination – use only multi-term occurrences at query time (≥*n* terms from the query must appear in the document in order for it to be returned)
3) Champion Lists – at indexing time, save a list of the *n* top ranked documents for each term, and use for ranked retrieval

When your implementation is ready, select the "Ranked query" option in the "Search Options" menu, and perform the same search with and without the speedup, for a representative dataset and representative set of queries, e.g. the same as in Tasks 1.4 and 2.3. Measure the computation time with and without the speedup, using the function `System.nanoTime()` or one of the profilers available for Java.

What is the average speedup, and how is it affected by parameters in the approximation method?

At the review

To pass Task 3.3, you should be able to describe the approximation you implemented, how much is gained in terms of computation time, and what approximations to the search are made in order to obtain the speedup. You should also be able to explain all parts of the code that you edited.

Task 3.4: Ranked Bi-Gram Retrieval (B or higher)

The task is now to **implement ranked retrieval using a bi-word model**. You will need to construct a new type of index called `BiwordIndex`, implementing the `Index` interface, as an alternative to the `HashedIndex` class. The terms in this index should be bi-grams. (For example, the phrase **mellan olika alternativ** consists of two bi-grams, **mellan olika** and **olika alternativ**.)

You will need to add code to the `search` method, so that when this method is called with the `structureType` parameter set to `Index.BIGRAM`, the system should perform retrieval based on the bi-word index rather than the standard one-word index. You only have to make ranked retrieval possible with this setting.

Change the parsing of the query so that each bi-gram in the query is compared to the bi-word index in turn, and that a cosine score is computed as a combination of the tf-idf scores for each bi-gram in the query.

How can the tf-idf score of a bi-gram be defined?

When your implementation is ready, compile and run it, indexing the data set `svwiki/files/1000`. (For memory reasons we only use 1000 documents.) Select the "Ranked retrieval" option in the "Search Options" menu, the "Bigram" option in the "Text Structure" menu, and try the search queries

mellan olika alternativ

which should result in a list similar to

Found 24 matching document(s)

```
 0. ../../svwiki/files/1000/293.txt   0...
 1. ../../svwiki/files/1000/674.txt   0...
 2. ../../svwiki/files/1000/33.txt   0...
 3. ../../svwiki/files/1000/16.txt   0...
 4. ../../svwiki/files/1000/41.txt   0...
 5. ../../svwiki/files/1000/693.txt   0...
 6. ../../svwiki/files/1000/39.txt   0...
 7. ../../svwiki/files/1000/380.txt   0...
 8. ../../svwiki/files/1000/47.txt   0...
 9. ../../svwiki/files/1000/619.txt   0...
```

etc.

and

den tyska huvudstaden

which should result in

```
Found 0 matching document(s)
```

and

tillvarons yttersta grunder

which should result in a list similar to

```
Found 3 matching document(s)

 0. ../2014/svwiki/files/1000/23.txt   0...
 1. ../2014/svwiki/files/1000/47.txt   0...
 2. ../2014/svwiki/files/1000/199.txt   0...
```

The similarity scores will vary greatly, depending on how you compute the tf-idf score of a bi-gram. **To take your focus off the scores, we have not listed our scores above.** However, the ranking orders should be similar to the ones above, and the number of retrieved documents should be exactly the same.

Compare to the same queries on the same dataset `svwiki/files/1000` using a standard ranked retrieval with a one-word index, as described in Assignment 2:

mellan olika alternativ

which should result in a list similar to

```
Found 402 matching document(s)

 0. ../../svwiki/files/1000/293.txt   0...
 1. ../../svwiki/files/1000/706.txt   0...
 2. ../../svwiki/files/1000/880.txt   0...
 3. ../../svwiki/files/1000/859.txt   0...
 4. ../../svwiki/files/1000/520.txt   0...
 5. ../../svwiki/files/1000/397.txt   0...
 6. ../../svwiki/files/1000/862.txt   0...
 7. ../../svwiki/files/1000/62.txt   0...
 8. ../../svwiki/files/1000/350.txt   0...
 9. ../../svwiki/files/1000/100.txt   0...
```

etc.

and

den tyska huvudstaden

which should result in a list similar to

```
Found 600 matching document(s)

 0. ../../svwiki/files/1000/222.txt   0...
 1. ../../svwiki/files/1000/723.txt   0...
 2. ../../svwiki/files/1000/123.txt   0...
 3. ../../svwiki/files/1000/744.txt   0...
 4. ../../svwiki/files/1000/564.txt   0...
 5. ../../svwiki/files/1000/558.txt   0...
 6. ../../svwiki/files/1000/36.txt   0...
 7. ../../svwiki/files/1000/127.txt   0...
 8. ../../svwiki/files/1000/562.txt   0...
 9. ../../svwiki/files/1000/38.txt   0...
```

etc.

and

tillvarons yttersta grunder

which should result in a list similar to

```
Found 33 matching document(s)

 0. ../../svwiki/files/1000/23.txt   0...
 1. ../../svwiki/files/1000/478.txt   0...
 2. ../../svwiki/files/1000/47.txt   0...
 3. ../../svwiki/files/1000/529.txt   0...
 4. ../../svwiki/files/1000/19.txt   0...
 5. ../../svwiki/files/1000/424.txt   0...
 6. ../../svwiki/files/1000/126.txt   0...
 7. ../../svwiki/files/1000/199.txt   0...
 8. ../../svwiki/files/1000/525.txt   0...
 9. ../../svwiki/files/1000/334.txt   0...
```

etc.

Run each of the 6 tests above using your implementation, look at the title of each of the 10 highest ranked document and label it as relevant or non-relevant to the question. Assume the total number of relevant documents in the dataset to be 20 for all queries – this only affects the recall measure up to a scale factor.

Produce 6 precision-recall graphs (of length 0 or higher…) and note if applicable the precision at 3, recall at 3, precision at 10 and recall at 10.

Are the bi-gram search results generally more precise than the standard uni-gram results (higher precision at 3, 10)? Does the bi-gram ranking list miss important relevant documents, that were returned by the uni-gram search (lower recall at 3, 10)?

At the review

To pass Task 3.4, you should be able to start the search engine with a dataset specified by the teacher, and perform a search in ranked retrieval and bi-gram mode with a query specified by the teacher, that returns the correct number of documents in an order similar to the model solution used by the teachers.

You should also be able to discuss the questions in italics above, showing 6 precision-recall curves for the three queries above on the dataset `svwiki/files/1000`, using bigram and unigram search, and to explain all parts of the code that you edited.

Task 3.5: Ranked Sub-Phrase Retrieval (A)

In Task 3.4, you saw that a standard ranked retrieval sometimes has a low precision, while a bi-gram ranked retrieval sometimes has a low recall. **A realistic search engine combines the advantages of both methods!** Use both the `BiwordIndex` and the `HashedIndex` at the same time, performing both a bi-gram search and a standard uni-gram search. (If you had more computational resources, it would be good to add a `TriwordIndex`, a `TetrawordIndex` and so on; Google has indexes up to 6-gram.)

You will need to add code to the `search` method, so that when this method is called with the `structureType` parameter set to `Index.SUBPHRASE`, the system should perform sub-phrase. You only have to make ranked retrieval possible with this setting.

The sub-phrase retrieval commonly used in search engines works like this:

Let the maximum indexed phrase length be n words (n = 2 in your case). Let the query length be m.

First, an min(n,m)-gram ranked retrieval is performed. (As an example, a 3-gram retrieval in the `svwiki/files/1000` data set with the query **tillvarons yttersta grunder** returns two matches, documents 23 and 47.)

If less than k documents are returned, proceed to do an (n–1)-gram retrieval. (As an example, a 2-gram (bi-gram) retrieval in the `svwiki/files/1000` data set with the query **tillvarons yttersta grunder** returns three matches, documents 23, 47, and 199.)

If less than k documents are returned from the (n–1)-gram retrieval, and n > 1, proceed to do an (n–2)-gram retrieval. Repeat until k documents are found or until n = 1. (As an example, a uni-gram (single term) retrieval in the `svwiki/files/1000` data set with the query **tillvarons yttersta grunder** returns 33 matches, documents 19, 20, 23, 47, 61, 126, 149, 199, 334, 389, 424, 478, 525, 529, 542, 548, 576, 600, 641, 656, 664, 721, 741, 834, 837, 878, 902, 909, 933, 935, 975, 976, and 990.)

Weigh together the document scores so that an n-gram match is always worth more than just an (n–1)-gram match, i.e., that documents are ranked according to how long substrings from the query are found in the document. *How should the engine weigh together the n-gram, (n–1)-gram, etc, score for a document, to achieve this?*

At the review

To pass Task 3.5, you should be able to start the search engine with a dataset specified by the teacher, and perform a search in ranked retrieval and sub-phrase mode with a query specified by the teacher, that behaves in a manner that cohere with the search results in Task 3.4.

You should be able to reason about the search result in relation to the results in Tasks 3.1 and 3.4 and how cosine scores for different phrase lengths are weighed together, and to explain all parts of the code that you edited.