DD2476 Search Engines and Information Retrieval Systems

Assignment 2: Ranked Retrieval

Johan Boye, Jussi Karlgren, Hedvig Kjellström

The purpose of Assignment 2 is to learn how to do ranked retrieval. You will learn 1) how to include tf-idf scores in the inverted index; 2) how to handle ranked retrieval from multiword queries; 3) how to use PageRank to score documents; and 4) how to combine tf-idf and PageRank scoring.

The recommended reading for Assignment 2 is that of Lectures 3-6.

Assignment 2 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 15 minutes or less, so have all papers in order.*

E: *Completed Task 2.1-2.4 with some mistakes that could be corrected at the review session.*
D: *Completed Task 2.1-2.4 without mistakes.*
C: *E + Completed Task 2.5.*
B: *C + Completed Task 2.6.*
A: *B + Completed Task 2.7.*

These grades are valid for review March 18, 2014. See the web pages www.kth.se/social/course/DD2476, ir14 - Computer assignments in the menu, for grading of delayed assignments.

Assignment 2 is intended to take around 50h to complete.

Computing Framework

For Tasks 2.1-2.3, and 2.7, you will be further developing your code from Assignment 1.

For Tasks 2.4-2.6, you will be using a source code skeleton found in the course directory `/info/DD2476/ir14/lab/pagerank`. Copy this directory to your home directory.

> If you are using your own computer you will need to copy the sub-directory `svwiki_links` as well.

The `pagerank` directory contains only the file `PageRank.java`, which is compiled simply by

```
javac PageRank.java
```

The program is executed as follows:

`java PageRank` *linkfile*

for instance

```
java PageRank /info/DD2476/ir14/lab/svwiki_links/links1000.txt
```

The link files are found in the folder `svwiki_links` in the course directory. Each line has the following structure:

```
1;365,959,944,
```

meaning that the article in `1.txt` is linking to the articles in `365.txt`, `959.txt` and `944.txt`. There are three such link files, representing the link structure within the first 1000 articles, 10000 articles, and all articles, respectively. The `PageRank` program reads such link files and represents the link structure internally by hash tables. **Note that the program internally uses other ID numbers for files!** For instance, article `365.txt` will be internally represented by some number which is not 365. (This is because we want the program to work for file names that do not happen to be integers.)

A convenient way of seeing what articles are about, is to look in the file `articleTitles.txt`, for instance by:

```
> egrep "^365;"
  /info/DD2476/ir13/lab/svwiki_links/articleTitles.txt
365;Danmark
```

Task 2.1: Ranked Retrieval

Extend the `search` method in the `HashedIndex` class to implement ranked retrieval. For a given search query, compute the cosine similarity between the tf-idf vector of the query and the tf-idf-vectors of all matching documents. Then sort documents according to their similarity score.

You will need to add code to the `search` method, so that when this method is called with the `queryType` parameter set to `Index.RANKED_QUERY`, the system should perform ranked retrieval. You will furthermore need to add code to the `PostingsList`, `PostingsEntry`, and `HashedIndex` classes, to compute the cosine similarity scores of the matching documents. To sort the matching documents, assign the score of each document to the `score` variable in the corresponding `PostingsEntry` object in the postings list returned from the `search` method. If you do this, you can then use the `sort` method in the built-in `java.util.Collections` class.

When you have finished adding to the program, compile and run it, 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

antiken

which should result in a list **similar** to

```
Found 167 matching document(s)

 0. ../../svwiki/files/3000/2029.txt   0.05278
 1. ../../svwiki/files/1000/358.txt   0.04369
 2. ../../svwiki/files/7000/6683.txt   0.02740
 3. ../../svwiki/files/4000/3339.txt   0.02587
 4. ../../svwiki/files/2000/1658.txt   0.02520
 5. ../../svwiki/files/2000/1502.txt   0.02473
 6. ../../svwiki/files/9000/8360.txt   0.02137
 7. ../../svwiki/files/10000/9340.txt   0.02125
 8. ../../svwiki/files/10000/9634.txt   0.01680
 9. ../../svwiki/files/10000/9887.txt   0.01573
```

etc.

and

```
    fysik
```

which should result in a list **similar** to

```
Found 286 matching document(s)

 0. ../../svwiki/files/6000/5736.txt   0.05314
 1. ../../svwiki/files/4000/3802.txt   0.05002
 2. ../../svwiki/files/1000/574.txt   0.03784
 3. ../../svwiki/files/10000/9090.txt   0.03738
 4. ../../svwiki/files/10000/9919.txt   0.03657
 5. ../../svwiki/files/8000/7128.txt   0.02455
 6. ../../svwiki/files/1000/397.txt   0.02194
 7. ../../svwiki/files/4000/3528.txt   0.01966
 8. ../../svwiki/files/3000/2500.txt   0.01807
 9. ../../svwiki/files/9000/8910.txt   0.01800
```

etc.

With one-word queries, the numbers above are equal to the length-normalized tf_idf scores of each document with respect to the query term. Our lists above were computed with a tf_idf score for document *d* and query term *t:*

$$\mathrm{tf\_idf}_{dt} = \mathrm{tf}_{dt} * \mathrm{idf}_{t} / \mathrm{len}_{d}$$

$$\mathrm{idf}_{t} = \ln(N/\mathrm{df}_{t})$$

where tf_{dt} = [*# occurrences of t in d*], N = [*# documents in the corpus*], df_{t} = [*# documents in the corpus which contain t*], and len_{d} = [*# words in d*].

Depending on exactly how you compute the similarity scores, the **numerical values of the scores can differ significantly** from the above, and the **ordering of the documents above might differ somewhat from those produced by your program –** this is fine.

However, your lists should not be fundamentally different from the ones above, and you should get the **same number of matching documents. (There are problems with the character encoding on Mac, contact us if you have deviations using a Mac.)**

At the review

There will not be any examination of Task 2.1, it is merely a preparation for Task 2.2.

Task 2.2: Ranked Multiword Retrieval

Modify your program so that it can search for multiword queries, and present a list of ranked matching documents. All documents that include at least one of the search terms should appear in the list of search results.

When your implementation is ready, compile and run it, 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
```

which should result in a list similar to

```
Found 3567 matching document(s)

 0. ../../svwiki/files/2000/1243.txt   0.27465
 1. ../../svwiki/files/10000/9496.txt   0.27465
 2. ../../svwiki/files/4000/3903.txt   0.10219
 3. ../../svwiki/files/10000/9067.txt   0.09285
 4. ../../svwiki/files/1000/293.txt   0.07976
 5. ../../svwiki/files/10000/9909.txt   0.07324
 6. ../../svwiki/files/2000/1064.txt   0.06761
 7. ../../svwiki/files/4000/3914.txt   0.05316
 8. ../../svwiki/files/9000/8492.txt   0.04994
 9. ../../svwiki/files/6000/5681.txt   0.04225
```
etc.

and

```
    den tyska huvudstaden
```

which should result in a list similar to

```
Found 5586 matching document(s)

 0. ../../svwiki/files/8000/7604.txt   1.09861
 1. ../../svwiki/files/10000/9224.txt   0.10267
 2. ../../svwiki/files/9000/8193.txt   0.09553
 3. ../../svwiki/files/8000/7750.txt   0.05853
 4. ../../svwiki/files/8000/7942.txt   0.04484
 5. ../../svwiki/files/6000/5120.txt   0.03582
 6. ../../svwiki/files/9000/8982.txt   0.03577
 7. ../../svwiki/files/8000/7611.txt   0.03537
 8. ../../svwiki/files/4000/3268.txt   0.03139
 9. ../../svwiki/files/8000/7963.txt   0.02953
```
etc.

and

```
    tillvarons yttersta grunder
```

which should result in a list similar to

```
Found 170 matching document(s)

0. svwiki/files/5000/4438.txt   0,04855
1. svwiki/files/1000/23.txt   0,03878
2. svwiki/files/6000/5882.txt   0,02519
3. svwiki/files/4000/3339.txt   0,02337
4. svwiki/files/7000/6829.txt   0,02023
5. svwiki/files/1000/600.txt   0,02022
6. svwiki/files/9000/8074.txt   0,01836
7. svwiki/files/8000/7014.txt   0,01316
8. svwiki/files/2000/1498.txt   0,01310
9. svwiki/files/7000/6664.txt   0,01135
```

etc.

Our list above was computed with the same length-normalized tf_idf score for each query term as in Task 2.1, weighed together using cosine similarity. The numbers above are the cosine scores for each document with respect to the query.

Depending on exactly how you compute the tf_idf scores, the **numerical values of the cosine scores can differ significantly** from the above, and the **ordering of the documents above might differ somewhat from those produced by your program** – this is fine.

However, your lists should not be fundamentally different from the ones above, and you should get the **same number of matching documents. (There are problems with the character encoding on Mac, contact us if you have deviations using a Mac.)**

Why do we use a union query here, but an intersection query in Assignment 1?

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, 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 explain all parts of the code that you edited, and be able to discuss the question in italics above.

The central concept here is the vector model for query-document similarity. You should be able to explain this concept using pen and paper, and discuss how variations in tf representation (such as log(1+tf)) and document length representation (such as Euclidean length, or sqrt(#words) affect the cosine similarity measure.

Task 2.3: What is a good search result?

This task is a continuation of Task 1.4. The purpose is now to assess whether ranked retrieval gives answers with higher precision and recall than unranked intersection retrieval; this will be done on our set of 10 representative queries.

NOTE: Like in Task 1.4, you will not be failed in the review for selecting the ”wrong” labels. However, make a try to label each document in a reasonable manner, if you are not a native Swedish speaker, just based on the title of the document and the titles of hyperlinks in the document. The labeling fills two functions:

1) **you get a feeling for the data that one works with in Information Retrieval, and understand how search engine evaluations are carried out, so that you can design your own evaluation later in your career,**
2) **you get data for your own experiments so that you can evaluate the search engine that you built here in Assignment 1 and Tasks 2.1-2.2.**

We first need to learn **how the quality of ranked retrieval results can be measured** – slightly differently from the unranked retrieval in Task 1.4.

Run the program from Task 2.2, indexing the 10 data sets `svwiki/files/1000`, `svwiki/files/2000`, ..., `svwiki/files/10000`. Select the "Ranked query" option in the "Search options" menu.

You will continue to extend the text file `FirstnameLastname.txt` from Task 1.4, but now with results from the ranked retrieval. Search the indexed data sets with each of the following 10 queries:

antikens underverk (114 hits)

olympiska spel och fred (6553 hits)

europacupen (16 hits)

konflikten i palestina (6754 hits)

snowboard (9 hits)

den europeiska bilindustrin (4403 hits)

enhetlig europeisk valuta (421 hits)

sex i reklam (6694 hits)

lutande tornet i pisa (6135 hits)

genteknik (6 hits)

For each of the above ten queries, look only at the **20 highest ranked** documents (if there are less than 20, all the returned documents). If you already came across that document for the same query in Task 1.4, use the existing relevance label. Otherwise, assess the relevance of the document for the query. As in Task 1.4, use the following four-point scale:

(0) Irrelevant document. The document does not contain any information about the topic.
(1) Marginally relevant document. The document only points to the topic. It does not contain more or other information than the topic description.
(2) Fairly relevant document. The document contains more information than the topic description but the presentation is not exhaustive.
(3) Highly relevant document. The document discusses the themes of the topic exhaustively.

[E. Sormunen. Liberal relevance criteria of TREC—Counting on negligible documents? *ACM SIGIR*, 2002]

Add the results into the file from Task 1.4 using the following space-separated format, one line per assessed document:

```
QUERY_ID DOC_ID RELEVANCE_SCORE
```

where `QUERY_ID` = [1, ..., 10], `DOC_ID` = the name of the text file, e.g., 365 for `365.txt`, `RELEVANCE_SCORE` = [0, 1, 2, 3]. **Do not remove anything from the file, it should contain the union of Task 1.4 and 2.3 document relevance labels for each query.** Like with Task 1.4, send the text file to `hedvig@kth.se`.

It should again be noted that there is no objectively correct relevance label for a certain query-document combination! It is a matter of judgement. For difficult cases, write a short note on why you chose the label you did. At the review, you will present three difficult cases for each query.

> The documents are readable in a standard text editor. However, a first 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
> ```

For each of the ten queries, plot a **precision-recall graph** for the returned top-20 list, and compute the **precision at 10** and **precision at 20** (relevant documents = documents with relevance > 0).

The recall is not possible to estimate without further information. However, it only adds a scale factor to the horizontal axis of the precision-recall graph. To get a recall estimate up to an unknown scale factor, assume the total number of relevant documents in the corpus to be 1000 for each query. Estimate the recall at 10 and recall at 20. Using the same assumption about the total number of relevant documents = 1000, estimate the recall of the answer to the same query in Task 1.4.

Compare the precision at 10 and precision at 20 for ranked retrieval to the precision for unranked retrieval for each query. *Which precision is the highest? Is it different for different queries? Are there any trends?*

Do the same comparison of recalls. *Which recall is the highest? Is it different for different queries? Are there any correlation between precision at 10, precision at 20, recall at 10, and recall at 20 for the same query?*

Does ranked retrieval in general give a higher or lower precision, higher or lower recall than unranked retrieval? Why is that?

At the review

To pass Task 2.3, you should show the text file with labeled documents for the 10 queries, in the correct format. You should have emailed it before presenting. You should be able to explain the concepts precision-recall graph and precision at K, and give account for these measures for each of the returned ranked top-20 document lists.

You should also be able to discuss the questions in italics.

Task 2.4: Computing PageRank with Power Iteration

Your task is to **extend the class `PageRank.java` so that it computes the pagerank of a number of Wikipedia articles** given their link structure. Use the standard power iteration method, as described in Lecture 5 and in the textbook (Section 21.2.2), and run your program both on `links1000.txt` (containing the link structure of the 1000 first documents) and `links10000.txt` (containing the link structure of the 10000 first documents). (Running the program on `links.txt`, which contains the entire Wikipedia link structure, is likely to take a very long time.)

Make sure your program prints the pagerank of the 50 highest ranked pages. Use the array `docName` to translate from internal ID numbers to file names.

A correctly implemented power iteration with c = 0.85 gives the following top-50 ranking for `links10000.txt`:

```
1.  1081 0,00393   11. 7031 0,00235   21. 5115 0,00170   31. 3105 0,00148   41. 837  0,00136
2.  522  0,00382   12. 3094 0,00228   22. 5621 0,00169   32. 723  0,00143   42. 6039 0,00135
3.  454  0,00357   13. 2381 0,00221   23. 425  0,00160   33. 6074 0,00142   43. 3743 0,00134
4.  2634 0,00354   14. 1306 0,00214   24. 6070 0,00156   34. 2635 0,00142   44. 2    0,00132
5.  365  0,00286   15. 9765 0,00192   25. 838  0,00155   35. 8071 0,00141   45. 4919 0,00132
6.  36   0,00277   16. 6287 0,00192   26. 6722 0,00154   36. 8098 0,00140   46. 664  0,00131
7.  526  0,00269   17. 1432 0,00188   27. 8184 0,00152   37. 2343 0,00138   47. 6451 0,00131
8.  3930 0,00264   18. 4762 0,00186   28. 1584 0,00150   38. 2136 0,00137   48. 8813 0,00129
9.  1324 0,00246   19. 2353 0,00186   29. 3931 0,00149   39. 21   0,00137   49. 5559 0,00127
10. 483  0,00239   20. 5608 0,00176   30. 6907 0,00149   40. 1524 0,00137   50. 2134 0,00127
```

The highest ranked document, 1081, has the title Latin, while the lowest ranked document, 669, has the title Gunnebo IP (a sports field in a tiny town in Sweden). Does this pagerank ordering seem reasonable? Why?

Look up the titles of some other documents with high rank. What is the trend with decreasing pagerank? What does the pagerank measure for this dataset, and what does it measure for documents on the internet in general?

At the review

To pass Task 2.4, you should show that the method returns a very similar top-50 ranking for `links10000.txt` to the one shown above. The difference in rank for a certain document should not be larger than ±2 positions, and the difference in pagerank value for the documents should not be larger than ±0.0005. However, the differences should preferrably be much smaller.

You should also be able to explain all parts of the code that you edited, and be able to discuss the questions in italics above.

Task 2.5: No-Sinks PageRank Approximation (C or higher)

The power iteration method is very time-consuming, but it is possible to compute approximate pagerank values in far less time.

A specific property of the Wikipedia dataset we are using here is that it contains no sinks: All documents contain links to at least one other document. This is the underlying assumption of the first method for approximate pagerank computation mentioned in Lecture 5.

Implement this algorithm and run it on `links10000.txt` to verify that it gives **almost the same solution** (within the error limits stated in Task 2.4, At the review) as the exact power iteration method.

Now use the no-sinks approximation to calculate the approximate pagerank of all Wikipedia articles, listed in `links.txt`. **This can be expected to take some time – it is normal, since there are in total ~10^6 documents in the entire dataset.** Save the list of the 50 highest ranked documents in a file so that you can show it at the review without running the algorithm.

What is the highest ranked document in the whole structure? Does it make sense, given your conclusions from Task 2.4 of what pagerank represents in this dataset?

At the review

To pass Task 2.5, you should run the no-sinks method on `links10000.txt`, returning a top-50 ranking very similar to the power iteration solution shown in Task 2.4.

You should show a record of the top-50 ranked documents from `links.txt` and be able to discuss the questions in italics.

You should also be able to explain all parts of the code that you edited.

Task 2.6: Monte-Carlo PageRank Approximation (B or higher)

The task is now to implement the five Monte-Carlo methods for approximate pagerank computation mentioned in Lecture 5.

Run all five variants on `links10000.txt`., using several different settings of all parameters (T, N, etc). Compare method variants and parameter settings in terms of how fast they converge and how similar the solution is to the exact solution. Arrange your results in tables so that you easily can explain your conclusions to the teacher.

Like in Task 2.5, convergence means that the method gives **almost the same solution** (within the error limits stated in Task 2.4, At the review) as the exact power iteration method.

Take a look at the 50 **lowest-ranked** documents for the best method variant with the best parameter setting. Compare to the 50 lowest-ranked documents returned from the exact power iteration method.

What do you see? Why do you get this result? Explain and relate to the properties of the (probabilistic) Monte-Carlo methods in contrast to the (deterministic) power iteration method.

Then, run the best method variant with the best parameter setting on `links.txt`. You should get almost the same top-50 ranking order as in Task 2.5.

At the review

To pass Task 2.6, you should show a record of your experimentation with the five method variants and their parameter settings. You should be able to discuss the differences between the variants, and in what situations (number of documents, average number of links from each document, etc) one variant is better than the other.

You should show a record of the bottom-50 ranked documents from the best Monte-Carlo method and the exact power iteration method on `links10000.txt` and be able to discuss the questions in italics.

You should also be able to explain all parts of the code that you edited.

Task 2.7: Combine tf-idf and PageRank (A)

Your final task is to integrate your results from Task 2.2 and 2.5 into the search engine we have been developing in Assignment 1 and Task 2.1 and 2.2. When doing a ranked query, make sure that the **score is computed as a function of the tf-idf similarity score and the pagerank** of each article in the result set. Design the combined score function so that you can vary the relative effect of tf-idf and pagerank in the scoring.

Use the pageranks you computed from `links.txt` in Task 2.5. You should pre-compute the pageranks and read them from file at the start of a search engine session. **Note that you should NOT use the pageranks from `links10000.txt`.**

You will need to add code to the `search` method, so that when this method is called with the `rankingType` parameter set to `Index.TF_IDF`, the system should perform ranked retrieval based on tf-idf score only, with the `rankingType` parameter set to `Index.PAGERANK`., only pagerank should be regarded, and with the `rankingType` parameter set to `Index.COMBINATION`, your combined score function is used to rank the documents.

When your implementation is ready, compile and run it, 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 the "Combination" option in the "Ranking Score" menu, and try the search queries listed in Task 2.2.

Each query should return the same number of matching documents as in Task 2.2. However, the ranking will vary depending on how you use the document pageranks in the score.

What is the effect of letting the tf-idf score dominate this ranking? What is the effect of letting the pagerank dominate? What would be a good strategy for selecting an ”optimal” combination? (Remember the quality measures you studied in Task 2.3.)

At the review

To pass Task 2.7, you should present a function for combining tf-idf and pagerank scores where the influence of the two factors can be varied.

You should be able to start the search engine with a dataset specified by the teacher, and perform a search in combination, ranked retrieval mode with a query specified by the teacher, that returns the correct number of documents, and be able to discuss the effect of tf-idf and pagerank on the subsequent ranking.

You should also be able to explain all parts of the code that you edited, and be able to discuss the question in italics above.