DD2476 Search Engines and Information Retrieval Systems

Assignment 2: Ranked Retrieval

Johan Boye, Carl Eriksson, 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 with some mistakes that could be corrected at the review session.*
B: *C + Completed Task 2.5 without mistakes.*
A: *B + Completed Task 2.6.*

These grades are valid for review March 3, 2015. See the web pages www.kth.se/social/course/DD2476, VT 2015 ir15 - Computer assignments in the menu, for grading of delayed assignments.

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

Computing Framework

For Tasks 2.1-2.2, and 2.6, you will be further developing your code from Assignment 1.

For Tasks 2.4-2.5, you will be using a source code skeleton found in the course directory `/info/DD2476/ir15/lab/pagerank`. Copy this directory to your home directory.

The `pagerank` directory contains 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 linksDavis.txt
```

The link file `linksDavis.txt` and the file with the article titles `articleTitles.txt` are also found in the folder `pagerank`. Each line in `linksDavis.txt` has the following structure:

```
1;2,3,4,
```

meaning that the article with number `1` according to `articleTitles.txt` is linking to the articles in `2`, `3` and `4` according to `articleTitles.txt`. The `PageRank` program reads such link files and represents the link structure internally by hash tables.

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 data set `davisWiki`. Select the "Ranked retrieval" option in the "Search Options" menu, and try the search queries

zombie

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

Found 36 matching document(s)

0. davisWiki/JasonRifkind.f ...
1. davisWiki /EmilyMaas.f ...
2. davisWiki/AliciaEdelman.f ...
3. davisWiki/Kearney_Hall.f ...
4. davisWiki/Zombie_Walk.f ...
5. davisWiki/Spirit_Halloween.f ...
6. davisWiki/StevenWong.f ...
7. davisWiki/Measure_Z.f ...
8. davisWiki/Scream.f ...

9. davisWiki/Disasters.f ...
etc.

and

attack

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

Found 228 matching document(s)

0. davisWiki/TheWarrior.f ...
1. davisWiki/Measure_Z.f ...
2. davisWiki/Kearney_Hall.f ...
3. davisWiki/Muilop.f ...
4. davisWiki/bg-33p.f ...
5. davisWiki/s.martin.f ...
6. davisWiki/TrustInMe.f ...
7. davisWiki/Furly707.f ...
8. davisWiki/Thong_H._Huynh.f ...
9. davisWiki/PamAarkes.f ...
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:*

$$\text{tf\_idf}_{dt} = \text{tf}_{dt} * \text{idf}_t / \text{len}_d$$

$$\text{idf}_t = \ln(N/\text{df}_t)$$

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** (why we do not list them at all), and the **ordering of the documents above might differ 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.**

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 you have finished adding to the program, compile and run it, indexing the data set `davisWiki`. Select the "Ranked retrieval" option in the "Search Options" menu, and try the search queries

zombie attack

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

Found 249 matching document(s)

0. davisWiki/JasonRifkind.f ...
1. davisWiki/Kearney_Hall.f ...
2. davisWiki/Measure_Z.f ...
3. davisWiki/Spirit_Halloween.f ...
4. davisWiki/Zombie_Walk.f ...
5. davisWiki/EmilyMaas.f ...
6. davisWiki/AliciaEdelman.f ...
7. davisWiki/Disasters.f ...
8. davisWiki/Scream.f ...
9. davisWiki/Biological_Disasters.f ...
etc.

and

money transfer

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

Found 1602 matching document(s)

0. davisWiki/MattLM.f ...
1. davisWiki/Angelique_Tarazi.f ...
2. davisWiki/JordanJohnson.f ...
3. davisWiki/NicoleBush.f ...
4. davisWiki/Title_Companies.f ...
5. davisWiki/Transfer_Student_Services.f ...
6. davisWiki/Anthony_Swofford.f ...
7. davisWiki/NinadelRosario.f ...
8. davisWiki/My_Brother_Steve_Computer_Services_%26_Tutoring.f ...
9. davisWiki/Transfer_Student_Association.f ...
etc.

and

sleeping on campus

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

Found 9885 matching document(s)

0. davisWiki/CharlesSuh.f ...
1. davisWiki/MichaelReagan.f ...
2. davisWiki/Maxipoo.f ...
3. davisWiki/Sleeping_on_campus.f ...
4. davisWiki/Zinfandel_Lounge.f ...
5. davisWiki/DaveNguyen122.f ...
6. davisWiki/money.f ...

7. **davisWiki/Campus.f ...**
8. **davisWiki/198_Young.f ...**
9. **davisWiki/King_Lounge.f ...**

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.

Depending on exactly how you compute the tf_idf scores, the **numerical values of the cosine scores can differ significantly** from the above (why we do not list them), and the **ordering of the documents above might differ 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.**

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 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 questions 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 three representative queries.

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 data set `davisWiki`. 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 the same queries as in Task 1.4:

skiing trip

university rowing team

tourist attractions

For each of the above three queries, look only at the **20 highest ranked** documents (if there are less than 20, all the returned documents) on the web. Most documents called `Doc_Name.f` has the web address `https://daviswiki.org/Doc_Name`. (There are some documents that lie in subfolders, you can find these with the Wikipedia search function on the Davis Wiki homepage.) **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, 2, 3], `DOC_ID` = the name of the document, `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.

For each of the three 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 three 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 on `linksDavis.txt`.

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 `linksDavis.txt`:

```
1: 121 0.007980901794924564
2: 21 0.007731247234492249
3: 245 0.007359891426395264
4: 1531 0.00509405660492995
5: 1367 0.0028366893437372915
6: 31 0.0025369975190081186
7: 80 0.002216407671615069
8: 1040 0.0021825129548366097
9: 254 0.0020234985434726717
10: 452 0.0019454428874005798
11: 157 0.0016263649985622695
12: 392 0.0016195381427957791
13: 169 0.0016098408219562277
14: 100 0.001563095428378879
15: 561 0.0014602141893513304
16: 3870 0.0014439897797260072
17: 997 0.0013543509333874295
18: 884 0.001277762824007422
19: 202 0.001266144765920104
20: 8 0.0012575035891859777
21: 72 0.001230526929286469
22: 145 0.0011901429022755287
23: 27 0.0010921473662683632
24: 645 0.001083156870669533
25: 490 0.001062683563202588
26: 2883 0.0010501250673160228
27: 81 0.0010264697730446022
28: 942 0.0010101607543902813
29: 125 9.522527024813006E-4
30: 247 9.402995763037513E-4
31: 337 8.777449765518368E-4
32: 179 8.776636143175264E-4
33: 708 8.766749797539502E-4
34: 1403 8.697030112833079E-4
35: 484 8.57637060167815E-4
36: 26 8.540721077509235E-4
37: 152 8.526138239010308E-4
38: 321 8.277735810492957E-4
39: 242 8.118822019024427E-4
40: 15492 7.968234400580981E-4
41: 1964 7.869667098749676E-4
42: 1043 7.853311765722786E-4
43: 857 7.714146325915579E-4
44: 1755 7.504804360825602E-4
45: 1200 7.226875154669279E-4
46: 281 7.119962573160739E-4
47: 154 7.091595631764484E-4
48: 16 7.031042059743013E-4
49: 1365 7.023635516744133E-4
50: 15686 7.00519329506775E-4
```

The highest ranked document has the title "Davis", while the document ranked 50 has the title "Interpreting User Statistics". 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?

At the review

To pass Task 2.4, you should show that the method returns a very similar top-50 ranking for `linksDavis.txt` to the one shown above. The difference in rank for a certain document should not be larger than ±2 positions (preferrably smaller), and the difference in pagerank value for the documents should not be larger than ±0.001 (preferrably 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: Monte-Carlo PageRank Approximation (C or higher)

The task is now to implement the five Monte-Carlo methods for approximate pagerank computation mentioned in Lecture 5 and in the paper by Avrachenkov et al. listed as course literature.

Run all three variants on `linksDavis.txt`, using $c = 0.85$ and several different settings of N (the number of initiated walks). Compare the five method variants and settings of N in terms of how fast they converge and how similar the solution is to the exact solution. Implement two goodness measures:

1. The sum of squared differences between the exact pageranks (found in Task 2.4) and the MC-estimated pageranks for the **50 documents with highest exact pagerank** in `linksDavis.txt`.
2. The sum of squared differences between the exact pageranks (found in Task 2.4) and the MC-estimated pageranks for the **50 documents with lowest exact pagerank** in `linksDavis.txt`.

Plot these goodness measures for all five methods as a function of N.

Do your findings about the difference between the five method variants and the dependence of N support the claims made in the paper by Avrachenkov et al.?

With convergence we here mean that the goodness measure 1 is within the error limits stated in Task 2.3, At the review. You should find a settings of N for all five methods so that they converge.

Take a look at the curves for goodness measure 2 and compare to goodness measure 1.

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.

At the review

To pass Task 2.5, you should show a record of your experimentation with the five method variants and their N parameter settings.

You should be able to discuss the questions in italics, and be able to discuss the differences between the five variants, and relate to the claims in the paper.

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

Task 2.6: Combine tf-idf and PageRank (A)

Your final task is to integrate your results from Task 2.4 into the search engine we have been developing in Assignment 1 and Tasks 2.1-2.2. 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 `linksDavis.txt` in Task 2.4. You should pre-compute the pageranks and read them from file at the start of a search engine session.

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 2.6, 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 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.