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 4-6.

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 cannot 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: *E + Completed Task 2.5 without mistakes.*
A: *C + Completed Task 2.6.*

These grades are valid for review March 6, 2017. See the web pages www.kth.se/social/course/DD2476, VT 2017 ir17 - 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/ir17/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`** is also found in the folder **`pagerank`**. Each line has the following structure:

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

meaning that webpage number **`1`** is linking to the articles in **`2,3`** and **`4.`** The **`PageRank`** program reads such link files and represents the link structure internally by hash tables. We are using numbers instead of the actual webpage names for the sake of brevity; however you can translate from numbers to filenames by using the table in **`articleTitles.txt`**.

Note that the numbers in `linksDavis.txt` and the internal docIDs that are produced on the fly when you run your search engine are NOT the same! Realizing this will save you many hours of debugging. Therefore I'm going to write it again:

Note that the numbers in `linksDavis.txt` and the internal docIDs that are produced on the fly when you run your search engine are NOT the same!

Task 2.1: Ranked Retrieval

Before starting the implementation of Tasks 2.1 and 2.2, go through the assigned reading for Lecture 4. It is not enough to look at the lecture notes.

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 e.g. could result in the list

```
Found 36 matching document(s)

 0. JasonRifkind.f   ...
 1. Zombie_Walk.f   ...
 2. EmilyMaas.f   ...
 3. AliciaEdelman.f   ...
 4. Kearney_Hall.f   ...
 5. Spirit_Halloween.f   ...
 6. Zombies_Reclaim_the_Streets.f   ...
 7. StevenWong.f   ...
 8. Measure_Z.f   ...
 9. Scream.f   ... etc.
```

and

```
    attack
```

which e.g. could result in the list

```
Found 228 matching document(s)

 0. TheWarrior.f   ...
 1. Measure_Z.f   ...
 2. Kearney_Hall.f   ...
 3. Muilop.f   ...
 4. bg-33p.f   ...
 5. Furly707.f   ...
 6. PamAarkes.f   ...
 7. s.martin.f   ...
 8. TrustInMe.f   ...
 9. stevenscott.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*].

The number of documents should be very similar to those listed above. Possible differences may depend on your regular expressions from assignment 1.

Depending on exactly how you compute the similarity scores, both the **numerical values of the scores** and the **ordering of the documents can differ significantly from those produced by your program** – this is fine. You can debug your results by manually computing the tf_{dt} and len_d scores for the top ranked documents *d* for a term *t*. (The idf score does not influence the ranking since there is only one term.)

We encourage you to try out different variants of the document length measurement len_d, such as the Euclidean length, the number of words, and the square root of the number of words (what would this correspond to?). What are the effect of different length estimates?

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

Found 249 matching document(s)

 0. JasonRifkind.f   ...
 1. Zombie_Walk.f   ...
 2. Kearney_Hall.f   ...
 3. Measure_Z.f   ...
 4. Spirit_Halloween.f   ...
 5. EmilyMaas.f   ...
 6. AliciaEdelman.f   ...
 7. TheWarrior.f   ...
 8. Scream.f   ...
 9. Zombies_Reclaim_the_Streets.f   ... etc.
```

and

```
     money transfer

Found 1600 matching document(s)

 0. MattLM.f   ...
 1. Angelique_Tarazi.f   ...
 2. JordanJohnson.f   ...
 3. Transfer_Student_Services.f   ...
 4. NicoleBush.f   ...
 5. Title_Companies.f   ...
 6. Anthony_Swofford.f   ...
 7. Transfer_Student_Association.f   ...
 8. Munch_Money.f   ...
 9. money.f   ...
 10. NinadelRosario.f   ... etc.
```

Furthermore, try the same query (or queries) with 3 words or more that you designed in Assignment 1 on the dataset.

Our lists above were computed with the same length-normalized tf_idf scores for each query term as in Task 2.1, weighed together using cosine similarity.

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

Look at the 10 highest ranked document returned by your search engine for each query. Why are these documents ranked highly? Be prepared to explain this using pen and paper.

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 1.5. 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 1.5.

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 1.5, but now with results from the ranked retrieval. Search the indexed data sets with the same query as in Task 1.5:

graduate program mathematics

Inspect the **50 highest ranked** documents. Most documents called `Doc_Name.f` have 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. Other documents are removed from the online wiki, but still present in our local copy. Then you can read the local copy – less nice format but still readable.) **If you already came across that document for the same query in Task 1.5, use the existing relevance label.** Otherwise, assess the relevance of the document for the query. As in Task 1.5, 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 1.5 using the following space-separated format, one line per assessed document:

`QUERY_ID DOC_ID RELEVANCE_SCORE`

where **`QUERY_ID = 1, 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 1.5 and 2.3 document relevance labels.** Like with Task 1.5, send the text file to **`jboye@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 judgment. For difficult cases, write a short note on why you chose the label you did. At the review, you will present three difficult cases.

Plot a **precision-recall graph** for the returned top-50 list, and compute the **precision at 10**, **20**, **30**, **40**, and **50** (relevant documents = documents with relevance > 0).

Assume the total number of relevant documents in the corpus to be 100, and estimate the **recall at 10**, **20**, **30**, **40**, and **50**.

Compare the precision at 10, 20, 30, 40, 50 for ranked retrieval to the precision for unranked retrieval. *Which precision is the highest? Are there any trends?*

Do the same comparison of recalls. *Which recall is the highest? Are there any correlation between precision at 10, 20, 30, 40 , 50, recall at 10, 20, 30, 40, 50?*

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 be able to show the text file with labeled documents, in the correct format. You should have emailed it before presenting. You should show the precision-recall graph for the 50 highest ranked documents, and be able to explain the concepts precision-recall graph and precision at K, and give account for these measures for the returned ranked top-50 document list.

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

Task 2.4: Computing PageRank with Power Iteration

Before starting the implementation of Tasks 2.4-2.6, please go through the assigned reading for Lecture 5.

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 the lecture notes and in the textbook (Section 21.2.2), and run your program on **`linksDavis.txt`**.

Note that the docIDs in `linksDavis.txt` and the internal docIDs that are produced on the fly in your index during indexing of the davisWiki corpus are NOT the same!

Make sure your program prints the pagerank of the 30 highest ranked pages. Use the array **`docName`** to translate from internal ID numbers to file names. Compare with the results in the file **`pagerank_top_30.txt`**.

The highest ranked document has the title ”Davis”. What is the title of the document ranked 30?. 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-30 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 6 and in the paper by Avrachenkov et al. listed as course literature.

Run all five 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 **30 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 **30 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 2.4, At the review. You should find 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 data set **`davisWiki`**. 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.