Web Search

Hands-on guides

João Magalhães
jmag@fct.unl.pt

Dep. Computer Science
NOVA FCT
Universidade NOVA Lisboa

Table of Contents

Introduction

The goal of this hands-on course is to provide students with an understanding of all aspects of the design and implementation of Web search engines. Students will master the fundamental concepts of Information Retrieval, i.e., text representation, indexing, querying, and ranking by relevance.

Learning outcomes

Knowledge:

- Learn the concept of information relevance.
- Analyse Web and multimedia data.
- Learn how to rank information by relevance.
- Understand evaluation protocols.

Know-how:

- Implement information retrieval models.
- Ability to adapt and improve components of a search engine.
- Deploy search engines with large-scale datasets.
- Design evaluation protocols and evaluate search engines.

Soft-Skills:

- Select the right IR techniques for particular problems.
- Design information retrieval systems.
- Ability to do critical thinking about retrieval results.

Organization

During the course lectures, we will discuss key concepts and introduce well-established information retrieval techniques and algorithms: the vector space model, the BM25 retrieval model, information relevance, PageRank, indexing, language-models and learning to rank.

Students are further exposed to these key information retrieval concepts on the laboratory lectures. The weekly laboratories, aim to provide students with a hands-on experience to allow the consolidation of the concepts discussed in the lectures.

The second part of the laboratory lectures, consists of a project, where the software constructs developed in the hands-on lectures will be applied to solve a real-world problem.

Pre-requisites:

- Good programming skills (Java and some scripting language)
- Critical analysis skills

Course grading:

- Project checkpoints: StackOverflow Answers Search (20%)
- Project: Summarization of information streams (40%)
- Final exam (40%)

Syllabus / planning

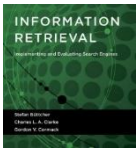
The weekly plan is presented on the following table.

#	Week	Lectures	#	In-class labs
1	11-Sep-17	Introduction	1	Setup environment (Lucene, Luke, …)
2	18-Sep-17	Basic techniques (P, R, P@10)	2	Text pre-processing, VSM
3	25-Sep-17	Query processing (P-R, MAP)	3	Evaluation protocols
4	02-Oct-17	PRP and Language models	4	Checkpoint 1: script + visualization
5	09-Oct-17	Probabilistic retrieval models	5	Query expansion
6	16-Oct-17	Rank fusion (NDCG)	6	Retrieval models
7	23-Oct-17	First test	7	Checkpoint 2: RM comparison
8	30-Oct-17	Linked data	8	Rank fusion
9	06-Nov-17	Crawling information sources	9	PageRank
10	13-Nov-17	Information diversity	10	Project
11	20-Nov-17	Static and dynamic indexing	11	Project
	27-Nov-17	-		-
12	04-Dec-17	Efficient query processing	12	Project
13	11-Dec-17	Second test	13	Project

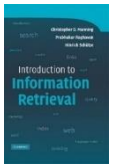
Classes:

- 2 hours lectures per week
- 2 hours hands-on classes per week

Bibliography



S. Büttcher. C. L. A. Clarke, G. V. Cormack, "Information Retrieval: Implementing and Evaluating Search Engines", The MIT Press, 2010. http://www.ir.uwaterloo.ca/book/



C. D. Manning, P. Raghavan and H. Schütze, "Introduction to Information Retrieval", Cambridge University Press, 2008. http://www-nlp.stanford.edu/IR-book/

Hands-on Web search

Goals

You will understand the following aspects of a search engine:

- Text pre-processing: tokenization, stop words, stemming, n-grams.
- Indexing fields and document ranking by relevance.
- Query-independent ranking: Ranking by document authority (PageRank).
- Search engine evaluation methods.

Case study project: Searching StackOverflow Q&A

In this case study, you will be guided through a number of steps to implement a search system for the StackOverflow CrossValidated Answers dataset. Your system should be able to:

1. Accept search queries from the command line;
2. Parse the search queries correctly;
3. Submit the search queries to the search engine;
4. Produce the search results in a specified file format.

The search system will be implemented incrementally, each week adding a new component.

Software requisites

The search index must be supported by Lucene (although, there are other more research oriented search engines). The project implementation can be in either Java (better search engine support) or Python (better text pre-processing support).

Lucene

Lucene is a search engine library that provides fundamental search algorithms and text processing methods to build text search engines. Download the Lucene library (https://lucene.apache.org/) and create an Eclipse project with the following Lucene jars (you can use Maven):

Libraries: core.jar analyzers-common.jar queries.jar queryparser.jar

There are other libraries external to Lucene that you may wish to use. However, it is strongly advisable to focus your efforts in studying and understanding the fundamentals of the search and indexing algorithms provided by Lucene.

Luke

To test and inspect the index you can use Luke. Luke is a "development and diagnostic tool, which accesses already existing Lucene indexes and allows you to display and modify their content in several ways". Download the jar file (https://github.com/DmitryKey/luke/releases) and run it in your local machine to inspect Lucene indexes and test search queries. Please, make sure you download a version of Luke that is compatible with your version of Lucene.

RankLib

RankLib is a library part of the LEMUR search engine, that implements several learning-to-rank algorithms.

Online discussion forum

Support will be provided on Google Groups.

Lab WS.1: Introduction to Lucene

In this first laboratory, you will get familiar with the basics of the text search framework Apache Lucene. A baseline java implementation of a search engine to index text documents and search the text index is available on CLIP. You must create a project with this sample code, correct existing errors and adapt it to solve this guide.

The goal of the search system, to be implemented throughout these hands-on labs, is to find the best answer to each query using only search algorithms. In this approach, you should index only the answers.

Dataset

From the lab materials Web page (http://ctp.di.fct.unl.pt/~jmag/ws/materials.html) download the dataset of questions and answers from The StackOverflow CrossValidated forum. Each question and answer have their own Id, which will be used to evaluate your ssearch results. There are other fields that you may wish to consider in your implementation.

Building and searching the index

When you create an index with Lucene with the IndexWriter class, you need to specify the Analyzer class that will analyse your text documents and the Similarity class that will compare your query to the documents in the index.

To search the index, you also need to specify an Analyzer class that will now analyse the query text and the Similarity class to compare queries to the documents in the index. The IndexSearh class provides several useful methods:

- The search method receives a query and returns a ranked list of documents from the index. Each document has a similarity score that quantifies the similarity between the document and the query.
- The searcher method explain also allows you understand numerical calculation of the similarity between a query and a document.

Data workflow

Understand the data workflow and draw a diagram depicting the data processing elements that you identify in the code.

Understanding search results

Based on the search results, inspect the search results using the explain method of the search framework. Describe the general structure of this output.

Based on the TF-IDF with cosine distance retrieval mode, discuss the output of the explain method of the Lucene framework.

Evaluation

Offline evaluation is done with the trec_eval command line application. From the lab materials Web page (http://ctp.di.fct.unl.pt/~jmag/WS/lab-materials.html) you can download its Windows binary or the Linux source code and build it on local machine. To print the evaluation report of your system, you need to run trec_eval from the command line as follows:

```
./trec_eval qrels.txt myresults.txt
```

Your software must write search results in a file format that enables *trec_eval* to produce evaluation reports. *trec_eval* expects its input to be in the format described below.

```
QueryID       Q0     DocID          Rank   Score  RunID
10            Q0     43254353       1      16     run-1
10            Q0     0987687462     3      9      run-1
:             :      :              :      :      :
11            Q0     2652542        1      18     run-1
```

The QueryID should correspond to the query ID of the query you are evaluating. Q0 is a required constant that you can ignore. The DocID should be the external document ID. *trec*_eval does not use the rank field to sort results, therefore the scores should be in descending order, to indicate that your results are ranked. The Run ID is an experiment identifier which can be set to anything.

Lab WS.2: Text pre-processing

Lucene is a full-text index system[1]. As such, it implements an extensive text analysis API with several algorithms to analyse text from different languages and domains[2]. The goal of the text pre-processing and analysis steps is to generate tokens, the minimal set of characters extracted from sentences that will be indexed internally.

Analyzers

Natural language is a rich form of representing knowledge. Most search engines take a pragmatic approach to textual information and seek for the most low-level, but relevant, patterns existing in text. Lucene follows this approach and processes text in a pipeline fashion, processing each word at a time. Lucene Analyzer classes are used to analyse text and produce the tokens to be used by the indexing or search tasks. When a document is indexed or a query is parsed, an Analyser is invoked through the *createComponents* method that returns a chain of TokenFilters that generate the final tokens from the original text.

```
Analyzer analyzer = new Analyzer() {
  @Override
   protected TokenStreamComponents createComponents(String fieldName) {
     Tokenizer source = new FooTokenizer(reader);
     TokenStream filter = new FooFilter(source);
     filter = new BarFilter(filter);
     return new TokenStreamComponents(source, filter);
   }
}
```

Text based Token filters

Study the code made available on CLIP and the course lecture about fundamental text pre-processing techniques. These techniques are implemented as TokenFilters and can be used in chain to strip text out of its irrelevant elements and reduce it to its canonical linguistic patterns.

Using the provided code understand how the different token filters generate different tokens: *punctuation removal, stop words, word-grams, n-grams, stemming*.

HTML Token filters

Pure text indexes, are designed to handle text only data. To handle structured documents, they must be decomposed into text segments, that are then indexed separately or without the structural information (e.g. html tags).

- Lucene provides a simple HTML parser that removes tags from a stream. The HTMLCharFilter class wraps a reader class and strips the HTML tags from that stream, hence, losing the structure information of the document.
- Other more complete HTML parsers exist such as Boilerpipe and Jsoup. You can use one of these libraries if you are not happy with the result of Lucene's HTML pre-processing result.

[1] Support for complex document formats, such as PDF, Word, XML and ODF, is provided by the Apache Tika library.
[2] Support for multiple languages, dictionaries, Wikipedia, HTML, etc. can be found on the analysis-common API.

```
Analyzer analyzer = new Analyzer() {
  @Override
   protected TokenStreamComponents createComponents(String fieldName) {
     Tokenizer source = new FooTokenizer(reader);
     TokenStream filter = new FooFilter(source);
     return new TokenStreamComponents(source, filter) {
                  @Override
                  protected void setReader(final Reader reader) {
                    super.setReader(new HTMLStripCharFilter(reader));
                  }};
   }
}
```

Fields extraction with regular expressions

In some cases, your text documents contain specific information that follow a pattern. For example, most of the times, the age of a person is expressed as "75 years old". In such situations, Lucene's class PatternReplaceCharFilter enables the implementation regular expressions to extract information that are exist in the documents according to a known text pattern.

Experiments automation

This homework requires a large number of experiments. If you use scripts in your favourite language to write parameter files with adjusted parameter settings, most of the experiments will require minimal manual effort, thus most of your effort will be in analysing experimental results. If you try to run all of the experiments manually, it will be very tedious and time-consuming.

We suggest that your script write results in a tabular format (e.g., .csv) similar to what you will need for your report to facilitate analysis.

Discussion

Discuss the impact of each text processing/tokenization method on the search results. Provide evidence in terms of numerical results. Fill the table bellow:

Analyser	**Input Sentence:** *A web search engine is a software system that is designed to search for information on the WWW.*
StopFilter	
WhitespaceTokenizer	
LowerCase	
StandardFilter	
CommonGramsFilter	
NGramTokenFilter	
EdgeNGramCommonFilter	
ShingleFilter	
SnowballFilter	
SynonymFilter	

Lab WS.3: Evaluation protocols

Evaluating the search results is an important step in the design and implementation of a search engine. The dataset is a key evaluation instrument: it requires query-document relevance judgments, indicating if a document is valid for a given query. In every evaluation, you must consider the types of relevance judgments that you require:

- **Binary relevance judgments:** For each question, an answer is considered to be relevant if it has a score greater than 1/3 of the question score.
- **Multi-level relevance judgments:** For each question, i) an answer is non-relevant (=0) if it has a score lower than 1/3 of the question score; ii) an answer relevant (=1) if it has a score between 1/3 and 2/3 of the question score; and iii) an answer is highly-relevant (=2) if it has a score higher than 2/3 of the question score.

Follow the script provided on CLIP to evaluate your search results. **You can only search for answers using the text information and the user PageRank information.**

Search utility metrics

Precision measures the number of relevant retrieved documents over a total of retrieved documents. A popular measure is Precision at 10 or 30 retrieved documents (first page or first three pages of results.

$$Precision = \frac{truePos}{truePos + falsePos}$$

Recall measures the number of relevant retrieved documents over the total number of relevant documents (including the ones that are not retrieved). This metric requires that the full set of documents is annotated with relevant or not relevant for every test query.

$$Recall = \frac{truePos}{truePos + falseNeg}$$

Normalized Discount Gain is useful when some documents are more relevant than others. Documents need to have ground-truth with different levels of relevance:

$$DCG_m = \sum_{i=1}^{m} \frac{2^{rel_i} - 1}{\log_2(1+i)} \qquad rel_i = \{0,1,2,3,\dots\} \qquad nDCG_m = \frac{DCG_m}{bestDCG_m}$$

Search stability metrics

Precision-recall graphs provide a detailed view of the complete search results (not just the top). It is important to assess the system stability across the entire rank and a wide range of queries.

Average precision is the area under the P-R curve:

$$AP = \frac{1}{\#relevant} \cdot \sum_{k \in \left\{ \substack{set\ of\ positions \\ of\ the\ relevant\ docs} \right\}} p@k$$

Mean average precision evaluates the system for a given range of queries. It summarizes the global system performance in one single value. It is the mean of the average precision of a set of n queries:

$$MAP = \frac{AP(q_1) + AP(q_2) + AP(q_3) + \dots + AP(q_n)}{n}$$

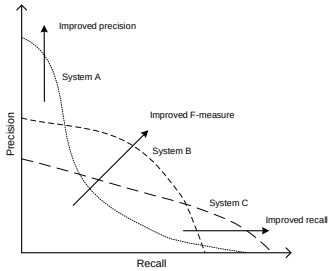


Figure 1. The precision-recall graph can be used to examine the behaviour of the search engine.

Systematic evaluation

Implement a script that generates and compiles all evaluation results into a single file. A second script is then required to parse the evaluation results file and generate all plots and the tables. This will allow you to easily pick the ones that best represent phenomena that you may wish to single-out and analyse.

Discussion

1. Once you have understood all the metrics, merge the results of all your past results. On a table, list the retrieval precision for all queries. Include the average retrieval precision.

2. Plot the precision recall curves for all queries in one single graph. Overlay the average of that curve. Discuss the variation.

3. Analyse and discuss the evaluation results from the perspective of retrieval recall.

4. Compare the MAP metric, the average of P@10 metric and the average of queries precision. Discuss the relation in terms of stability and utility. Use the precision-recall graphs to support your points.

Lab WS.4: Query expansion[3]

The purpose of this lab is to gain experience with query expansion algorithms. You must conduct experiments with initial document rankings created by i) your implemented algorithms, and ii) a reference baseline. You must conduct five experiments that investigate the sensitivity of pseudo relevance feedback algorithm to parameter settings and its effectiveness in different situations.

Linguistic query expansion

The intuitive way of expanding a query is to add terms to the query that synonyms of the original query terms. To implement this expansion method, one needs a dictionary to find word synonyms. WordNet is lexical database of English and provides sets of synonyms.

In Lucene, WordNet is available as WordnetSynonymParser and can be used to expand the query (expanding words at indexing time significantly increases the computational complexity and the require space to store the index).

Corpus-based query expansion

Pure linguistic terms expansion can be too generic to be useful in specific domains. For example, in programming, the technical jargon can be too specific to be expanded by a language standard synonyms. A better way to expand domain specific jargon with synonyms is to look at the co-occurrence of terms in the date.

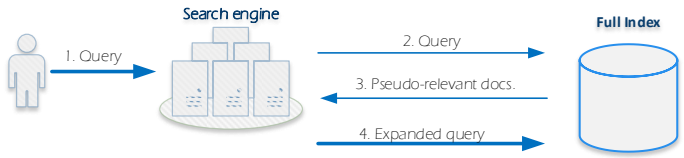
A corpus based expansion can be implemented by searching a single query term at a time and counting the most common words in the top returned documents. Repeating this process for all query terms will give you a list of terms that co-occur frequently with your query terms. Thus, they can be used as to expand the query in the domain of that corpus.

Implement a corpus-based query expansion. For each query term, use the top 5 documents and the 3 most frequent words.

Pseudo-relevance feedback

Relevance feedback is a manual procedure to expand the original query with terms from documents that the user mark as relevant. It builds on the possibility of doing interactive search where on each iteration the user adds more information to the original query.

Pseudo-relevance feedback simulates the user and automatically expands the original query with the most frequent terms of the top retrieved documents. The expanded query is resubmitted to the search engine and new results.



Implement a pseudo-relevance feedback query expansion algorithm. Use the top 10 documents as pseudo-relevant documents and consider the 10 most frequent words as expansion terms. Assume a weight of 1.0 on the original query terms and a weight of 0.5 on the expanded terms. Compare

[3] This lab is an excerpt from: http://boston.lti.cs.cmu.edu/classes/11-642/HW/HW3/

the retrieval results of (1) a baseline retrieval model without expansion and (2) a retrieval model using pseudo-relevance query expansion.

Discussion

1. Compare the effects of setting the number of feedback documents to 10, 20, 30, 40, 50, and 100 on PRF query expansion accuracy and stability using the reference baseline results for the initial retrieval.

2. Use the best setting that you discovered in your previous experiment. Compare the effects of setting the number of feedback terms to 5, 10, 20, 30, 40, and 50 on PRF query expansion accuracy and stability using the reference baseline results for the initial retrieval.

3. Use the best settings that you discovered in your previous experiments. Compare the effects of setting the weight of the original terms to 0.0 (only expansion terms), 0.2, 0.4, 0.6, 0.8, and 1.0 (no query expansion) on query expansion accuracy and stability using the reference baseline results for the initial retrieval.

4. Investigate the effects of PRF μ smoothing parameter on the retrieval performance of expanded queries. Generate three sets of expanded queries: fbTerms=10, fbTerms=20, and fbTerms=30. (You may use whatever values of fbDocs you think is best, based on your previous experiments.) Compare the effects of the default μ=2500 setting and five other values of μ on retrieval accuracy. Do the longer expanded queries require different smoothing than the shorter unexpanded queries?

Lab WS.5: Retrieval models

The purpose of this lab is to get familiar with the most popular retrieval models. Lucene's search framework implements several retrieval models to compute the collection statistics and rank documents based on different fundaments. You should be able to relate the experimental results to the different foundations of each retrieval model.

Documents ranking with Lucene's retrieval models

A central challenge to search engines is the retrieval model that computes a rank of documents based on a free text query provided by the end-user. Over the years many retrieval models have been researched, each one showing particular advantages in specific domains (e.g., medical domain, long-documents, short-documents, etc).

Lucene implements several retrieval models. Lucene's API, allow changing the retrieval model both at indexing and search time:

- At indexing time, use the method: IndexWriterConfig.setSimilarity(Similarity)
- At search time, use the method: IndexSearcher.setSimilarity(Similarity)

By default, Lucene uses the BM25 retrieval model (however, this depends on the version).

Vector Space Model (Cosine TF-IDF)

The *Vector Space Model* (implemented as the ClassicSimilarity) is one of the most popular retrieval models. It weights the terms with the term frequency and with the inverse document frequency. The ranking is obtained with the cosine distance.

Best Model 25 (BM25)

The family of probabilistic retrieval models has a long history of successive improvements, until the mid 90's when the *Probabilistic Model BM25* was invented (implemented as BM25Similarity). There are many variations of the BM25 for different situations, however, the BM25 retrieval model is still the best model for a wide range of collections

$$RSV = \sum q_t \cdot \frac{f_{t,d}(k_1+1)}{k_1\left((1-b)+b\left(\frac{l_{avg}}{l_d}\right)\right)+f_{t,d}} \cdot IDF_t$$

The variable k_1 controls term frequency scaling ($k_1 = 0$ is binary model and $k_1 = 1$ large is raw term frequency. The variable *b* controls document length normalization (*b = 0* is no length normalization and *b = 1* is fully scaled by document length).

Language Model with Dirichlet Smoothing (LMD)

Language models estimate the term distribution of the collection and the terms distribution of each document. Retrieval models based on language models, combine the two distributions in different ways. The *Language Model with Dirichlet Smoothing* retrieval model (implemented in LMDirichletSimilarity), smooths the document terms distribution with a prior distribution corresponding to the collection's terms distribution:

$$p(q|d) = \prod_{t \in q}\left(\frac{f_{t,d}+\mu \cdot M_c(t)}{|d|+\mu}\right)^{q_t}$$

The μ variable controls the smoothing effect, i.e., the deviation of the document's terms distribuion from a prior distribution (the collection terms distribution).

Discussion

1. Present the results of the studied retrieval models in a tabular format (ClassicSimilarity, BM25Similarity and LMDirichletSimilarity). Use the main retrieval metrics (P@10, MAP and NDCG). Plot the precision recall curves for all retrieval models. Discuss the results.

2. Using the setup of the previous question, compare the effects of different values of mu in the LMDirichletSimilarity (consider 10, 100, 500, 1000 and and 5000). Discuss the results.

3. Using the setup of the previous question, compare the effects of setting different values to b and k_1 in the BM25Similarity. Use b={0.0, 0.25, 0.5, 0.75, 1.0} and k_1={0, 0.5, 1.0, 1.25, 1.5, 2.0 }. Discuss the results.

4. Examine the document length per query and relate it to the performance of each retrieval model. Examine the collection statistics and relate it the performance of the retrieval models. Discuss the ideal conditions for each retrieval model.

Lab WS.6: Rank fusion

In this lab you will study different methods to merge the outputs of different retrieval models and different statistics related to individual documents. Merging different ranks, allows taking advantage of the different properties of each retrieval model. We will examine methods from two different areas: unsupervised rank fusion and supervised rank fusion methods.

Indexing multiple fields

Indexing documents across multiple fields is a fundamental functionality to index information according to their semantics (e.g., title, body, prices, names, locations). A problem arising from multi-field indexes is that you may wish to analyse the text of each field with different Analyzers.

Taking advantage of Lucene's **analysers** and **retrieval models**, create multiple fields each one with the configurations listed below and with the analysers of your choice. Based on your observations of the previous labs, decide the configuration of the three last fields.

Once the index is created, you can query just one field, a few of them or even all fields and merge them onto a single rank. Lucene also supports querying multiple fields and computing a weighted average of the documents.

Index the first paragraph and the full body of answers in separate fields using different analysers.

#	Statistic	Target	Data	Description
1	VSM	Q,D	Full body (no HTML)	
2	BM25	Q,D	Full body (no HTML)	
3	LMD	Q,D	Full body (no HTML)	
4	TF	Q,D	Full body (no HTML)	
5	IDF	Q,D	Full body (no HTML)	
6	UserRank	D	Without manual scores	PageRank of users with manual scores
7	UserRank	D	With manual scores	PageRank of users without manual scores
8	VSM	Q,D	Code elements	
9	BM25	Q,D	Code elements	
10	LMD	Q,D	Code elements	
11	TF	Q,D	Code elements	
12	IDF	Q,D	Code elements	
13	VSM	Q,D	First paragraph	
14	BM25	Q,D	First paragraph	
15	LMD	Q,D	First paragraph	
16	TF	Q,D	First paragraph	
17	IDF	Q,D	First paragraph	
18	Length	D	Length	
23				

PerField processing

Lucene provides the PerFieldAnalyserWrapper classes to select the text analysis algorithms according to the field name. This example shows how we can build an analyser, *aWrapper*, that can later be passed to the *IndexWriter* or *IndexWriterConfig*:

```
Map<String,Analyzer> analyzerPerField = new HashMap<>();
analyzerPerField.put("firstname", new KeywordAnalyzer());
analyzerPerField.put("lastname", new KeywordAnalyzer());

PerFieldAnalyzerWrapper aWrapper =
        new PerFieldAnalyzerWrapper(new StandardAnalyzer(), analyzerPerField);
```

Similarly, Lucene also provides the PerFieldSimilarityWrapper abstract class to select the retrieval model according to the field name. In this case, you must implement the actual class:

```
class Lab7_RetrievalModels extends PerFieldSimilarityWrapper {

        Map<String, Similarity> similarityPerField = new HashMap<>();

        @SuppressWarnings("deprecation")
        public Lab7_RetrievalModels() {
                similarityPerField.put("firstsentence", new LMDirichletSimilarity());
                similarityPerField.put("body", new BM25Similarity());
        }

        @Override
        public Similarity get(String s) {
                return similarityPerField.get(s);
        }
}
```

Querying multiple fields

There are two ways to search multiple fields with Lucene. The first method is with unsupervised rank fusion methods and the second method is with learning to rank methods.

The unsupervised fusion methods, require you to query all fields you need, collect all the documents and weighted them according to one of the methods (see the following section).

The learning to rank methods is supported by the MultiFieldQueryParser class to search multiple fields. The learning to rank algorithms are responsible to learn the weight of each field in a weighted average fashion. Then, the constructor receives the set of boost that weight fields according to their importance:

```
MultiFieldQueryParser(String[] fields, Analyzer analyzer, Map<String,Float> boosts)
```

Individual fields have their own analyser and possibly retrieval model. To generate the *Query object,* the query text must be processed by the same analyser used to index that field, and then passed to the *Similarity* model. Using the same PerFieldAnalyser that was used at indexing time, you can query multiple fields as follows:

```
String[] query = {"query1", "query2", "query3"};
String[] fields = {"filename", "contents", "description"};
BooleanClause.Occur[] flags = {BooleanClause.Occur.SHOULD, BooleanClause.Occur.MUST,
                BooleanClause.Occur.MUST_NOT};
MultiFieldQueryParser.parse(query, fields, flags, analyzer);
```

Unsupervised rank fusion

Unsupervised rank fusion methods are an inexpensive way of merging different ranks. When merging ranks, it is possible to consider the positions of a document on the multiple ranks or to consider the scores of documents on those same multiple ranks.

The reciprocal rank fusion weights each document with the inverse of its position on the rank, favouring documents at the "top" of the rank and penalizing documents below the "top" of the rank. This rank fusion method is expressed as

$$RRFscore(d) = \sum_i \frac{1}{k + r_i(d)}.$$

Instead of using just the position of the document on the list, other methods use the score of the document on the different lists. These include three rank fusion methods that use the maximum (CombMAX), minimum (CombMIN) or sum (CombSUM) of a document scores. The CombMNZ multiplies the number of ranks where the document occurs by the sum of the scores obtained across all lists. Despite normalization issues common in score-based methods, CombMNZ is competitive with rank-based approaches.

$$CombMAX(d) = \max\{s_0(d), \dots, s_n(d)\} \qquad CombMIN(d) = \min\{s_0(d), \dots, s_n(d)\}$$

$$CombSUM(d) = \sum_i s_i(d) \qquad CombMNZ(d) = |\{i | d \in Rank_i\}| \cdot \sum_i s_i(d)$$

Using the indexes generated in the previous lab, implement a rank fusion class that queries the different fields and merges the outputs with the five above methods.

Learning-to-rank

The above methods ignore the performance of each individual rank, becoming vulnerable to noise that may exist in some rank. LETOR methods[4] are a family of supervised methods that have an ordinal output. A key challenge in this task is that the objective function in information retrieval is not convex, such as the MAP or the P@10 metrics, resulting on a non-trivial optimization problem.

Despite the simplicity of linear models, when the objective function is considered on a multinomial manifold. The Coordinate Ascent algorithm can then be successfully applied to learn the weighted linear combination that maximizes MAP or another retrieval metric,

$$RSV(q, d) = \lambda_1 \cdot f_1(q, d) + \lambda_2 \cdot f_2(q, d) + \cdots + \lambda_n \cdot f_n(q, d),$$

where for each query *q*, $f_i(q, d)$ corresponds to a position of the document *d* on the rank produced by the field *i*. There may be an arbitrary number of fields/ranks.

Model development. Read the RankLib documentation and understand the format of the training files. Read your *qrels.txt* file and for each query-document pair, generate the scores for all features/fields of the indexes of the previous lab. Use this file to train a coordinate ascent model, optimising the MAP metric and store the model into a (human readable) file.

[4] SVMRank is a LETOR model that makes use of maximum-margin theory to learn the ranking function. GBDT is one of the best models to solve LETOR problems.

Discussion

1. Explain how Lucene combines the search results of the different fields. Suggest methods to improve it. Compare the performance of the individual ranks (individual fields) to the performance of the combined ranks.

2. Lucene allows to change the weight of some fields with the boost method. Discuss the boosting method at indexing and search time.

3. Discuss the analysers used on each field. Explain the semantics and the rationale of each field.

4. Change the regularization parameter (0.1, 0.2, 0.5, 1.0, 2.0) of the coordinate-ascent and present the new results.

5. List the field weights computed by the coordinate-ascent learning-to-rank method. Analyse the field weights on a per-query basis and discuss the performance differences. Compare the field weights to the performance of the individual ranks.

Lab WS.7: PageRank for the StackOverflow Social-Graph

Ranking with text only can lead to tied results, moreover, it ignores many other sources of evidence that can further improve the search results. On the Web (and in particular social-media), it is widely known that user feedback provides some of the best sources of evidence.

Google's page rank is one of the first algorithms to use human relevance in the ranking of Web information. It relied on the links created by humans as a proxy to the question: "*how relevant is this page on the whole Web?*" The PageRank of a Web page *i*, is defined as

$$p_i = \frac{(1-d)}{N} + d \cdot \sum_{j=1}^{N} \left(\frac{L_{ij}}{c_j} \right) p_j$$

Where d is the teleporting factor, p_j is the PageRank of Web page *j*, c_j is the number of out links of that page, and L_{ij} indicates if there is a valid link from page p_j to page p_i.

StackOverflow social-graph

In the StackOverflow dataset, human relevance is explicitly provided by the score given to an answer. We obviously don't want to score the answers but the users instead, to predict the relevance of future answers published by that user, and thus, influence the ranking of the answers.

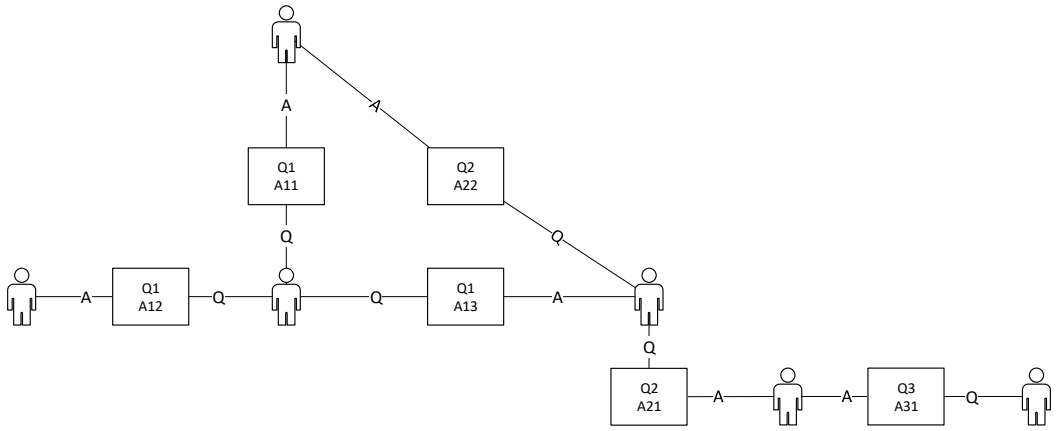


Figure 2. Sample example of a StackOverflow social-graph.

PageRank

In today's lab, you will implement the PageRank algorithm based on the users' feedback. You should create a graph of users where a set of Q&A will link sets of users. The steps to compute the PageRank for users are as follows:

- **Step 1:** Parse the *questions.csv* file, load the questions and identify the set of users.
- **Step 2a:** Parse the Answers file and link the Q-user and the A-user:
  - **Step 2b:** Decide which should be the direction of the link (Q->A or A->Q).
  - **Step 2c:** Store the full set of links grouped by user.
- **Step 3:** Initialize the PageRank of each user with a seed value of *1/(#total users)*.
- **Step 4:** Iterate over the full set of links and update the PageRank of each user.

PageRank with weighted links

The link between a Question and an Answer is further qualified by the manual scores. Adapt your PageRank implementation to consider the manual score of Questions and Answers in the propagation of the PageRank value.

Re-rank search results

Run your PageRank implementation for 10 iterations and output the PageRank value of each user into a file. Implement a re-ranking method that computes the text-based search results and update its order with the users' PageRank value. For each document, its PageRank and Lucene score should be weighted differently according to a parameter passed in your reranking method. This weight must be calibrated offline.

Discussion

1. Test the re-ranking method with different weights and plot the change in P@10. What is the best weight? Discuss the curve.

2. Plot the users PageRank versus the manual score of the user's answers. Discuss the relation.

3. Plot the user's PageRank versus the user's number of answers/questions the user. Discuss the relation.

4. Discuss how PageRank, or other graph based methods, can be applied to other problems in Web and Social-media search.

Exercises

1. Consider a retrieval system with TF-IDF weighting and cosine ranking. The repository has a total of 1000 documents.

   a. Compute the similarity between the two documents.

   > *A: "This update is designed to reduce rankings for low-quality sites—sites which are low-value add for users, copy content from other websites or sites that are just not very useful."*
   >
   > *B: "We can't make a major improvement without affecting rankings for many sites. It has to be that some sites will go up and some will go down."*
   >
   > Source: http://googleblog.blogspot.com/2011/02/finding-more-high-quality-sites-in.html

   b. Which document is the most relevant for the query "ranking sites"?

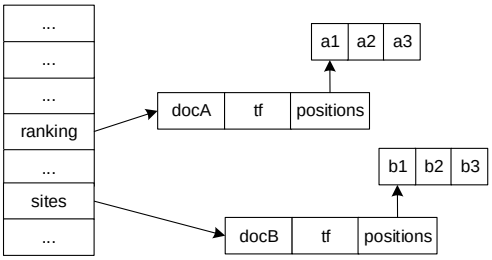
   c. The TF-IDF weighting is composed by two parts. Explain the motivation for each part and detail how they are combined.

2. Suppose that a user's initial query is "ranking sites". The user examines two documents, A and B (the same from the previous question). She judges A, relevant and B nonrelevant. Assume that we are using direct term frequency (with no scaling and no document frequency). There is no need to length-normalize vectors.

   a. Using Rocchio relevance feedback what would the revised query vector be after relevance feedback? Assume $\alpha = 1$, $\beta = 0.75$, $\gamma = 0.25$.

   b. Discuss the limitations of the Rocchio algorithm.

3. Consider an indexing system implementing a specific postings structure including, the weight and the occurrence positions within the document.



   a. Propose a term weighting scheme and a ranking algorithm considering the occurrence position information.

   b. Modify the Block Sort-Based Indexing method for this method. Consider the proposed method.

   c. Discuss how the two methods would work together.

4. Consider two ranking algorithms that for the same query produced the two following ranks:

   S1: d4 d3 d5 d8 d2 *and* *S2: d3 d9 d5 d6 d1*

   a. Assuming that the relevant documents are d9, d1, d3 and d4, compute the precision and recall values of each system.

   b. Assuming that the multi-value relevance judgments of documents are d9=1, d1=1, d3=3 and d4=2, assess and compare the two ranks with the appropriate metric.

   c. Assume no relevance judgments and compare the two systems.

5. Consider a search engine implementing a query categorization module and a topic-specific page rank algorithm. The query categories are: sports, news, health, navigational.

   a. Propose a method to categorize queries. Consider the DMOZ directory.

   b. Discuss in detail how the final rank is computed from the query categorization, the topic-specific PageRank and the document ranking modules.

6. Consider the two following documents:

   d1 : Jackson was one of the most talented entertainers of all time

   d2: Michael Jackson anointed himself King of Pop

   a. Using a BM25 retrieval model determine which document is more relevant to the query q= “Michael Jackson” (consider b = 0.75 and k = 1.5).

   b. Using a language model with Jelinek-Mercer smoothing determine which document is more relevant to the query q= “Michael Jackson” (consider $\lambda = ½$)

   c. Using a language model with Dirichlet smoothing determine which document is more relevant to the query q= “Michael Jackson” (consider $\mu = 100$)

7. Considering the Probability Ranking Principle as a starting point, discuss the rationale behind the BM-25 model and the Language Model with Dirichlet Smoothing.

8. Relate the Binary Independence Model to the Inverted Document Frequency.

9. What are the differences between standard vector space TF-IDF weighting and the BIM probabilistic retrieval model (in the case where no document relevance information is available)?

10. Information Retrieval systems can implement several different weighting schemes and ranking functions.
    a. Discuss the differences between the following term weighting functions:
        i. Binary
        ii. Frequency
        iii. tf-idf
    b. Discuss the differences between the following ranking functions:
        i. Euclidean distance
        ii. Cosine
        iii. BM-25

11. Show that models resulting from Dirichlet smoothing can be treated as probability distributions. That is, show that $\sum_t M_d^u(t) = 1$.

12. You have discovered that documents in a certain collection have a "half-life" of 30 days. After any 30-day period a document's prior probability of relevance p(r|D) is half of what it was at the start of the period. Incorporate this information into LMD. Simplify the equation into a rank-equivalent form, making any assumptions you believe reasonable.

13. Let X_t be a random variable indicating whether the term t appears in a document. Suppose we have $|R|$ relevant documents in the document collection and that $X_t = 1$ in s of the documents. Take the observed data to be just these observations of X_t for each document in R. Show that the MLE for the parameter $p_t = P(X_t = 1|R = 1, \sim q)$, that is, the value for p_t which maximizes the probability of the observed data, is $p_t = \frac{s}{|R|}$.

14. Consider making a language model from the following training text:

    the martian has landed on the latin pop sensation ricky martin

    a. Under a MLE-estimated unigram probability model, what are P(the) and P(martian)?
    b. Under a MLE-estimated bigram model, what are P(sensation|pop) and P(pop|the)?

15. Consider a search engine implementing a query categorization module and a topic-specific page rank algorithm. The query categories are: sports, news, health, navigational.
    a. Propose a method to categorize queries. Consider the HITS system.
    b. Discuss how the topic-specific PageRank is implemented.

16. Suppose we have a collection that consists of the 4 documents given in the below table.

docID	Document text
1	click go the shears boys click click click
2	click click
3	metal here
4	metal shears click here

Build a query likelihood language model for this document collection. Assume a mixture model between the documents and the collection, with both weighted at 0.5. Maximum likelihood estimation (mle) is used to estimate both as unigram models.

a. Work out the model probabilities of the queries "click", "shears", and hence "click shears" for each document, and use those probabilities to rank the documents returned by each query.

b. What is the final ranking of the documents for the query click shears?

17. Using the calculations of the previous exercise as inspiration or as examples where appropriate, write one sentence each describing the treatment that the LM with Jelinek-Mercer smoothing gives to each of the following quantities. Include whether it is present in the model or not and whether the effect is raw or scaled.

a. Term frequency in a document

b. Collection frequency of a term

c. Document frequency of a term

d. Length normalization of a term

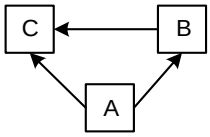
18. In the mixture model approach to the query likelihood model,

$$p(q|d,C) \approx \prod_{t\in\{q\cap d\}}\big(\lambda \cdot p(t|M_d) + (1-\lambda)\cdot p(t|M_c)\big),$$

the probability estimate of a term is based on the term frequency of a word in a document, and the collection frequency of the word. Doing this certainly guarantees that each term of a query (in the vocabulary) has a non-zero chance of being generated by each document. But it has a more subtle but important effect of implementing a form of term weighting, related to TF-IDF that was discuss in (Manning et al., Chapter 6). Explain how this works. In particular, include in your answer a concrete numeric example showing this term weighting at work.

19. Consider the PageRank algorithm applied to a term graph.
    a. Propose an algorithm to compute the term graph from a set of documents (consider LSI and simple term co-occurrence).
    b. Discuss possible ways in which the PageRank algorithm can be used to compute the importance of a term.

20. Consider the PageRank ranking algorithm.



    a. Compute the PageRank of each page on the following Web graph. Consider PR(A) = 0.2 and a teleporting factor of 0.3.

    b. Explain how PageRank can be adapted to user preferences. (Assume preferences are expressed by categories of content).

    c. In the PageRank algorithm what is the teleporting factor and why is it needed?

    d. Compute the HITS algorithm for the above Web graph.

21. Rank fusion methods combine ranks in different manners. Compute the fused ranks for the following three lists with the CombSUM, CombMNZ, BordaFuse and RRF.

Rank 1 (id/score)	**Rank 2 (id/score)**	**Rank 3 (id/score)**
D3 / 0.5	D3 / 0.8	D9 / 0.9
D4 / 0.2	D8 / 0.8	D3 / 0.8
D2 / 0.19	D2 / 0.8	D1 / 0.7
D5 / 0.18	D1 / 0.5	D8 / 0.6
D6 / 0.07	D5 / 0.4	D2 / 0.5
D1 / 0.05	D6 / 0.32	D5 / 0.4
D7 / 0.01	D9 / 0.31	D6 / 0.3
D9 / 0.01	D7 / 0.30	D7 / 0.2