João Magalhães
Dep. Computer Science, NOVA FCT
Universidade NOVA Lisboa

Guided project
StackOverflow Social-Graph Answers Search

Table of Contents

Introduction ... 2
Goals ... 2
Project grading ... 2
Report ... 2
Final submission ... 2
Lab 0: Libraries, luke, Lucene demo, first submission ... 3
Software ... 3
Lucene ... 3
Luke ... 3
Dataset ... 3
Baseline implementation ... 3
Offline evaluation ... 4
Exercise: Baseline submission ... 4
Lab 1: Analysers and tokenizers ... 5
Analyzers ... 5
Token Filters: punctuation, stop words, stemming, n-grams… ... 5
Exercise: Submission with improved text processing ... 5
Lab 2: Ranking and indexing multiple fields ... 6
Documents ranking with Lucene's retrieval models ... 6
Indexing multiple fields ... 6
Querying multiple fields ... 6
Exercise: Submission with search over multiple fields ... 6
Lab 3: PageRank for the StackOverflow Social-Graph ... 7
StackOverflow Social-Graph ... 7
Exercise: Submission with PageRank re-ranked results ... 7
Lab 4: Evaluation protocol ... 8
Search utility metrics ... 8
Search stability metrics ... 8
Exercise: Evaluate your search engine ... 8

João Magalhães
Dep. Computer Science, NOVA FCT
Universidade NOVA Lisboa

Introduction

Goals

In this project 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.

You will have to 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.

Project grading

The project grading is split between the final report+code (25%) and the weekly progress. At the end of each lab, each group must submit the search results produced by their system implementation done in the class. Groups that manage to show an improvement in precision metrics over the past class (on weeks 2, 3, and 4), are awarded 25% towards the final grade.

Report

The project report is limited to a maximum of 4 pages (no cover page, no annexes), it must follow the ACM template (Word or Latex, https://www.acm.org/publications/proceedings-template), and it must be structured as follows:

1. Introduction
2. Algorithms from classes used in the project
3. Implementation: What are your key ideas? What makes your project unique?
4. Evaluation
    a. Dataset description
    b. Baselines
    c. Results and discussion
5. Critical summary
6. References

Final submission

The project is considered submitted only when the group (1) submits the final report and their implementation through EasyChair, and (2) submits the final search results through Kaggle.

João Magalhães
Dep. Computer Science, NOVA FCT
Universidade NOVA Lisboa

Lab 0: Libraries, luke, Lucene demo, first submission

Software

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):

- core
- analyzers-common
- queries
- queryparser
- spatial
- benchmark

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.

Dataset

Download the dataset (https://www.kaggle.com/stackoverflow/statsquestions) of questions and answers from CrossValidated.

Understand the data fields: Each question and answer have their own Id, which are used internally by Kaggle to score your submission. There are other fields that you may wish to consider in your implementation.

You should try to find the best answer to each question using only search algorithms. In this approach, you should index only the answers. The questions will be your source for queries. Each answer is linked to a corresponding question and has its own score.

Baseline implementation

Download and **study the baseline** java implementation from CLIP.

Understand the data workflow: Draw a diagram depicting the data processing elements that you identify in the code.

Offline evaluation

Offline evaluation is done with the trec_eval command line application. Download it and compile it in your local machine. 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     745457437     2       11     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.

Download the relevance judgments from CLIP.

Exercise: Baseline submission

Your first submission requires you to register your group, not later than March 23. Please, indicate your student numbers in the name of the group.

Your submission will be evaluated with Mean Precision at 10, hence, each query can be answered with only 10 documents because only the top 10 documents will be assessed. Prepare your submission according to the *trec_eval* format and submit it through the indicated URL:

https://docs.google.com/a/fct.unl.pt/forms/d/e/1FAIpQLSfgQo77kw50sYHypB9NZYLw-RSEA7yrPEO_7YyjIXcwKBsvZA/viewform?usp=sf_link

Policy about cheating. You must be prepared to show the implementation used to compute the search results of every submission. Your group will have a penalty of 50% if you fail to reproduce your search results.

Lab 1: Analysers and tokenizers

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);
   }
}
```

Token Filters: punctuation, stop words, stemming, n-grams…

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 punctuation/characters/words and reduce the text 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

Exercise: Submission with improved text processing

> **Submit search results:** Build your own analyser and examine the impact that each choice has on the quality of the returned answers. Submit your best search results.

> **Report:** Discuss the impact of each text processing/tokenization method on the search results. Provide evidence in terms of numerical results.

[1] Support for complex document formats, such as PDF, Word, XML and ODF, is provided by the Apache Tika library, http://tika.apache.org/.
[2] Support for multiple languages, dictionaries, Wikipedia, HTML, etc. can be found on the analysis-common API https://lucene.apache.org/core/6_4_2/analyzers-common/index.html.

Lab 2: Ranking and indexing multiple fields

Lucene's search framework provides many software constructs to rank documents and it also implements convenient methods to better understand search results ranking. Based on the results of the first lab, inspect the search results using the explain method of the search framework.

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. The most common one is the *Vector Space Model* and two of the most successful ones are the *Probabilistic Model BM25* and the *Language Model with Dirichlet Smoothing*. Using Lucene's API, search the Answers' body field with these three retrieval models[3]:

- Index the collection with a similarity function: IndexWriterConfig.setSimilarity(Similarity)
- Search the index with the same similarity function: IndexSearcher.setSimilarity(Similarity)

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.

Lucene provides the AnalyserWrapper and 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);
```

Querying multiple fields

Lucene provides the MultiFieldQueryParser to search multiple fields[4]. 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);
```

Exercise: Submission with search over multiple fields

> Index the first sentence and the full body of answers in separate fields using different analysers.
> **Submit search results:** Build your own analyser and examine the impact that each choice has on the quality of the returned answers. Submit the search results with the your analyser.
> **Report:** Discuss the impact of each text processing/tokenization method on the search results. Provide evidence in terms of numerical results.

[3] Lucene trusts that you specify the same scoring function at indexing and searching time.
[4] It is also possible to have specific similarity functions for each field with the PerFieldSimilarityWrapper class.

Lab 3: 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?*"

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.

Thus, 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 Answers and Questions files and identify the set of users providing Questions and Answers.
- **Step 2a:** Parse the Answers and Questions files a second time and for each Q&A create a link between 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 weight of the link according to the answer/question score.
  - **Step 2d:** You should use a variable for storing the full set of links sorted 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.

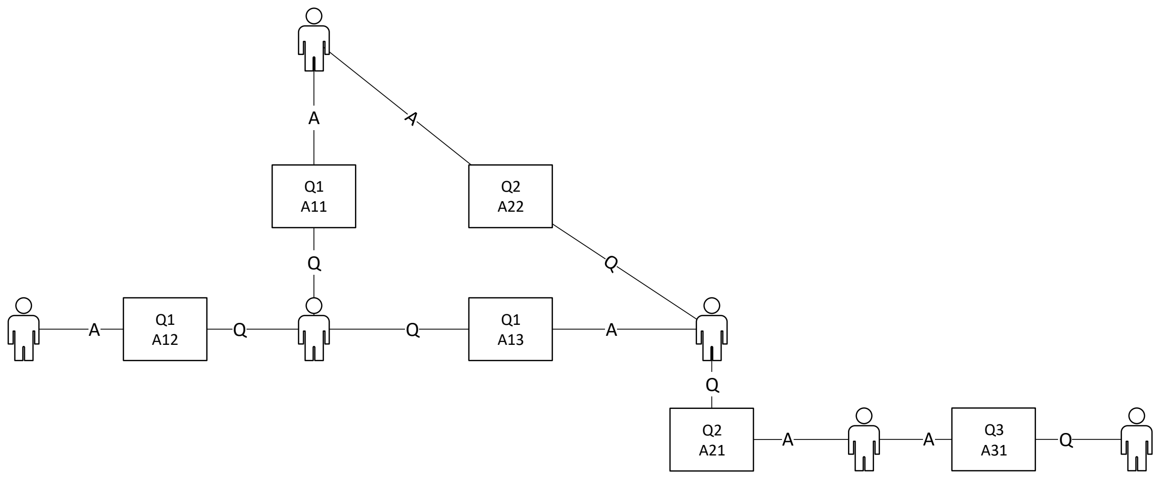

Figure 1. Sample example of a StackOverflow social-graph.

Exercise: Submission with PageRank re-ranked results

> **Submit search results:** Once you have computed the PageRank of each user, you must consider its value in the final ranking of the *answers search results*. Re-rank the top 50 results based on the PageRank information and submit your re-ranked search results.

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

Lab 4: Evaluation protocol

Evaluating the search results is an important step in the design and implementation of a search engine. In our evaluation you should consider the following relevance judgments (also known as ground-truth or oracle information):

- **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.

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

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:

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:

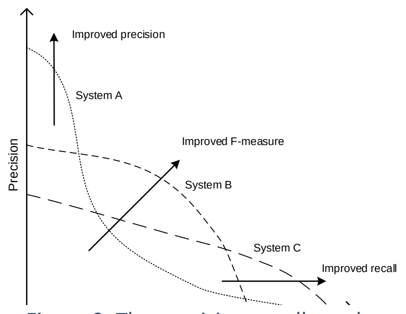

Figure 2. The precision-recall graph can be used to examine the behaviour of the search engine.

Exercise: Evaluate your search engine

> **Analyse all your past search submissions:** Once you have understood all the metrics, compile the results for all your past results.

> **Report:** Prepare your report with the above evaluation results. You can fix past runs if you wish to. Do a critical analysis of your submitted and non-submitted results.