MAC LABO 1

Indexing and Search with Apache Lucene

Claude-André Alves & Jérémy Corbaz

2 Familiarizing with Lucene

2.1 Understanding the Lucene API

1. **What are the types of the fields in the index?**

   LongPoint, StringField, TextField

2. **What are the characteristics of each type of field in terms of indexing, storage and tokenization?**

LongPoint : indexes long values for efficient range filtering ans sorting
StringField : this field is indexed but not tokenized, the entire string value is indexed as a single token
TextField : indexed and tokenized, without term vectors
Field : this one is for general purpose allows to specify name, value and type. to be used instead of TextField to be able to access the Term Vector

3. **Does the command line demo use stopword removal? Explain how you find out the answer.**

It doesn't. I searched for "usa" and "the usa", the results are different and if it used stopword removal we would have exactly same result.

4. **Does the command line demo use stemming? Explain how you find out the answer.**

It doesn't. We tried with the queries *long* and *longitude*, both queries gave us different responses.

5. **Is the search of the command line demo case insensitive? How did you find out the answer?**

yes it's case insensitive. When you type *USA* it searches for *usa* so it makes the input lower case.

6. **Does it matter whether stemming occurs before or after stopword removal? Consider this as a general question.**

Yes it different if we make it before because we could stemm a stopword for example *was* into *wa* and it would be suppress by the stopword removal. And additionally it will have the same root as other words and would confused the search.

3 Indexing and Searching the CACM collection

3.1 Indexing

1. **Find out what is a “term vector” in Lucene vocabulary?**

   Term vector is a list of that tells us the frequency, the offset and the position of the word it is attached to.

2. **What should be added to the code to have access to the “term vector” in the index? Have a look at the different methods of the class FieldType . Use Luke to check that the “term vector” is included in the index.**

when declaring a field put Field.Store.YES as the "Store" argument.

3. **Compare the size of the index before and after enabling “term vector”, discuss the results.**

652.7 ko vs *1.4 Mo* so *1400ko* so the size did more than doubled with the term vector on

3.2 Using different Analyzers

1. **Explain the difference of these five analyzers.**

   Whitespace: This analyzer with simply split the document based on whitespace and nothing more. It won't lower case and it will keep all punctuations.

   English: It will keep the English terms and will apply some filter. It will lower case, stem and remove stop word like 'of', 'for', 'a' .

   Shingle 1-2: It will tokenize the text and make terms with single or 2 words in it. It lower case the terms.

   Shingle 1-3: It will tokenize the text and make terms with single, 2 or 3 words in it. It lower case the terms.

   StopAnalyzer: It will lower case the text, remove the stopwords that are in a list.

2. **Look at the index using Luke and for each created index find out the following information:**

a. The number of indexed documents and indexed terms.

b. The number of indexed terms in the summary field.

c. The top 10 frequent terms of the summary field in the index.

d. The size of the index on disk.

e. The required time for indexing (e.g. using System.currentTimeMillis() before and after the indexing).

white space

a. number of documents 3203 number of terms 34827
b. number of terms in summary 26821
c. the top 10 frequent terms in summary
  1 .of
  2 .the

3 .is
4 .a
5 .and
6 .to
7 .in
8 .for
9 .The
10 .are

d. 1.5Mo
e. 1258 milliseconds

English

a. number of documents 3203 number of terms 23010
b. number of terms in summary 16724
c. the top 10 frequent terms in summary

1 .us
2 .which
3 .comput
4 .program
5 .system
6 .present
7 .describ
8 .paper
9 .method
10.can

d. 1.3Mo
e. 1526 milliseconds

Shingle 1 and 2

a. number of documents 3203 number of terms 119846
b. number of terms in summary 100768
c. the top 10 frequent terms in summary

1 .the
2 .of
3 .a
4 .is
5 .and
6 .to
7 .in
8 .for
9 .are
10.of the

d. 2.8Mo
e. 2853 milliseconds

Shingle 1 and 3

a. number of documents 3203 number of terms 251565
b. number of terms in summary 218853
c. the top 10 frequent terms in summary
   1 .the
   2 .of
   3 .a
   4 .is
   5 .and
   6 .to
   7 .in
   8 .for
   9 .are
   10.of the
d. 5.0Mo
e. 4067 milliseconds

stop

a. number of documents 3203 number of terms 24663
b. number of terms in summary 18342
c. the top 10 frequent terms in summary
   1 .system
   2 .computer
   3 .paper
   4 .presented
   5 .time
   6 .method
   7 .program
   8 .data
   9 .algorithm
   10.discussed
d. 1.3Mo
e. 1841 milliseconds

4. **Make 3 concluding statements bases on the above observations.**
    1. the number of documents isn't analyzer dependent.
    2. the proportion of summary terms is more or less the same every time
    3. the top 10 terms in summary varies depending of the analyzer

3.3 Reading Index

1. **What is the author with the highest number of publications? How many publications does he/she have?**

   This the author **Thacher Jr., H. C.** and he has 38 publications.

2. **List the top 10 terms in the title field with their frequency.**

Place	Title	Doc Frequency	Total frequency
1	algorithm	1008	1035
2	comput	392	401
3	program	307	311
4	system	280	281
5	gener	169	173
6	method	156	157
7	languag	144	147
8	function	142	147
9	us	123	123
10	data	110	112

```
public void printTopRankingTerms(String field, int numTerms) {
        // This methods print the top ranking term for a field.
        // See "Reading Index".

        TermStats[] termsTop;
        try {
            termsTop = HighFreqTerms.getHighFreqTerms(indexReader,
numTerms, field, new HighFreqTerms.TotalTermFreqComparator());
        } catch (Exception e) {
            e.printStackTrace();
            return;
        }

        System.out.println("Top ranking terms for field [" + field + "]
are: ");
        for (TermStats ts : termsTop) {
            System.out.println("\t" + ts);
        }
    }
```

3.4 Searching

```
public void query(String q) {

        // See "Searching" section
        QueryParser queryParser = new QueryParser("summary", analyzer);
        Query query = null;
        try {
            query = queryParser.parse(q);
        } catch (ParseException e) {
            e.printStackTrace();
        }

        System.out.println("Searching for [" + q + "]");

        try {
            // search query
            ScoreDoc[] hits = indexSearcher.search(query, 1000).scoreDocs;
            // retrieve results
            for (ScoreDoc hit : hits) {
                Document doc = indexSearcher.doc(hit.doc);
                System.out.println(doc.get("id") + ": " + doc.get("title") + "
(" +
                        hit.score + ")");
            }
            System.out.println("Results found: " + hits.length);
            System.out.println();
        } catch (Exception e) {
            e.printStackTrace();
        }


    }
```

1. Publications containing the term "Information Retrieval"

Searching for ["Information Retrieval"]
891: Everyman's Information Retrieval System (3.5482824)
1457: Data Manipulation and Programming Problemsin Automatic Information Retrieval (3.5482824)
1935: Randomized Binary Search Technique (3.334264)
1699: Experimental Evaluation of InformationRetrieval Through a Teletypewriter (3.286451)
2519: On the Problem of Communicating Complex Information (2.830918)
2516: Hierarchical Storage in Information Retrieval (2.511626)
2307: Dynamic Document Processing (2.377547)
2795: Sentence Paraphrasing from a Conceptual Base (2.315736)
2990: Effective Information Retrieval Using Term Accuracy (2.257058)
2451: Design of Tree Structures for Efficient Querying (2.0976045)

Results found: 11

2. Publications containing both "Information" and "Retrieval".

Searching for [Information AND Retrieval]
1457: Data Manipulation and Programming Problemsin Automatic Information Retrieval (4.0519514)
891: Everyman's Information Retrieval System (3.845176)
3134: The Use of Normal Multiplication Tablesfor Information Storage and Retrieval

(3.6613934)
1032: Theoretical Considerations in Information Retrieval Systems (3.613603)
1699: Experimental Evaluation of InformationRetrieval Through a Teletypewriter (3.6011014)
2307: Dynamic Document Processing (3.5647495)
1935: Randomized Binary Search Technique (3.3342643)
1681: Easy English,a Language for InformationRetrieval Through a Remote Typewriter Console (3.2606215)
2519: On the Problem of Communicating Complex Information (3.217997)
2990: Effective Information Retrieval Using Term Accuracy (3.146757)

Results found: 23

3. Publications containing at least the term “Retrieval” and, possibly “Information” but not “Database”.

Searching for [(+Information Retrieval) NOT Database]
1457: Data Manipulation and Programming Problemsin Automatic Information Retrieval (4.0519514)
891: Everyman's Information Retrieval System (3.845176)
3134: The Use of Normal Multiplication Tablesfor Information Storage and Retrieval (3.6613934)
1032: Theoretical Considerations in Information Retrieval Systems (3.613603)
1699: Experimental Evaluation of InformationRetrieval Through a Teletypewriter (3.6011014)
2307: Dynamic Document Processing (3.5647495)
1935: Randomized Binary Search Technique (3.3342643)
1681: Easy English,a Language for InformationRetrieval Through a Remote Typewriter Console (3.2606215)
2519: On the Problem of Communicating Complex Information (3.217997)
2990: Effective Information Retrieval Using Term Accuracy (3.146757)

Results found: 149

4. Publications containing a term starting with “Info”.

Searching for [Info]*
222: Coding Isomorphisms (1.0)
272: A Storage Allocation Scheme for ALGOL 60 (1.0)
396: Automation of Program Debugging (1.0)
397: A Card Format for Reference Files in Information Processing (1.0)
409: CL-1, An Environment for a Compiler (1.0)
440: Record Linkage (1.0)
483: On the Nonexistence of a Phrase Structure Grammar for ALGOL 60 (1.0)
616: An Information Algebra - Phase I Report-LanguageStructure Group of the CODASYL Development Committee (1.0)
644: A String Language for Symbol Manipulation Based on ALGOL 60 (1.0)
655: COMIT as an IR Language (1.0)

Results found: 193

5. Publications containing the term “Information” close to “Retrieval” (max distance 5).

```
Searching for ["Information Retrieval"~5]
891: Everyman's Information Retrieval System (3.5482824)
1457: Data Manipulation and Programming Problemsin Automatic Information Retrieval
(3.5482824)
1935: Randomized Binary Search Technique (3.334264)
1699: Experimental Evaluation of InformationRetrieval Through a Teletypewriter
(3.286451)
2307: Dynamic Document Processing (2.9471455)
2519: On the Problem of Communicating Complex Information (2.830918)
2516: Hierarchical Storage in Information Retrieval (2.511626)
2795: Sentence Paraphrasing from a Conceptual Base (2.315736)
2990: Effective Information Retrieval Using Term Accuracy (2.257058)
2451: Design of Tree Structures for Efficient Querying (2.0976045)
```

Results found: 15

3.5 Tuning the Lucene Score

Show the top 10 results (with scores) of both similiraty functions. Describe the effect of using the new parameters.

ClassicSimilarity

```
Searching for [compiler program]
3189: An Algebraic Compiler for the FORTRAN Assembly Program (1.4853004)
1215: Some Techniques Used in the ALCOR ILLINOIS 7090 (1.40438)
1183: A Note on the Use of a Digital Computerfor Doing Tedious Algebra and Programming
(1.3361712)
1459: Requirements for Real-Time Languages (1.3162413)
718: An Experiment in Automatic Verification of Programs (1.3136772)
1122: A Note on Some Compiling Algorithms (1.3136772)
1465: Program Translation Viewed as a General Data Processing Problem (1.2863079)
2652: Reduction of Compilation Costs Through Language Contraction (1.2732332)
1988: A Formalism for Translator Interactions (1.2580339)
46: Multiprogramming STRETCH: Feasibility Considerations (1.2391106)
```

Results found: 578

MySimilarity

```
Searching for [compiler program]
2923: High-Level Data Flow Analysis (13.265991)
2534: Design and Implementation of a Diagnostic Compiler for PL/I (13.243084)
637: A NELIAC-Generated 7090-1401 Compiler (12.414103)
1647: WATFOR-The University of Waterloo FORTRAN IV Compiler (12.195253)
2652: Reduction of Compilation Costs Through Language Contraction (10.961201)
1465: Program Translation Viewed as a General Data Processing Problem (9.89246)
1988: A Formalism for Translator Interactions (9.89246)
```

3189: An Algebraic Compiler for the FORTRAN Assembly Program (9.89246)
1135: A General Business-Oriented Language Based on Decision Expressions* (9.839054)
1237: Conversion of Decision Tables To Computer Programs (9.839054)

Results found: 578

```java
package ch.heigvd.iict.dmg.labo1.similarities;

import org.apache.lucene.search.similarities.ClassicSimilarity;

public class MySimilarity extends ClassicSimilarity {
    // Implement the functions described in section "Tuning the Lucene Score"

    @Override
    public float tf(float freq) {
        return (float) (1 + Math.log(freq));
    }

    @Override
    public float idf(long docFreq, long docCount) {
        return (float) (Math.log((double) docCount / (docFreq + 1)) + 1);
    }

    @Override
    public float lengthNorm(int numTerms) {
        return 1;
    }



}
```

We can clearly see that we have different order by using another class of comparison. the score is smaller with the classical than with MySimilarity.