MAC Labo01

Maurice Lehmann & Alexandre Simik

2.1

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

- The path of the file : Stringfield
- The last modified date of the file : Longpoint
- The contents of the file : TextField

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

- Path of the file : is searchable, but no tokenization, term frenquency or positional information
- Last modified date of the file : Indexed
- The contents of the file : Tokenized & indexed

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

Yes, given to the Apache documentation (https://lucene.apache.org/core/4_1_0/demo/) :
"The Analyzer we are using is StandardAnalyzer, which creates tokens using the Word Break rules from the Unicode Text Segmentation algorithm specified in Unicode Standard Annex #29; converts tokens to lowercase; and **then filters out stopwords**. Stopwords are common language words such as articles (a, an, the, etc.) and other tokens that may have less value for searching"

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

No, the stemming process depend of the language of the text. Given to the same Apache documentation : "It should be noted that there are different rules for every language, and you should use the proper analyzer for each.".
In the demo, we use the StandardAnalyzer wich fit for a basic usage of indexation.

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

Yes, it is not case sensitive, as it is said in the same documentation :
"The query parser is constructed with an analyzer used to interpret your query text in the same way the documents are interpreted: finding word boundaries, **downcasing**, and removing useless words like 'a', 'an' and 'the'."

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

It depend of what you want. Say you have "stemmed" words in your stopwords list, if you remove the stopwords after the stemming, it will remove a lot of words.
According to Jordan Boyd-Graber (https://www.quora.com/What-should-be-done-first-stopwords-removal-or-stemming-Why-In-weka-should-I-perform-stemming-to-stopwords-list-so-the-word-abl-can-be-removed) :
"Given the choice, I think it would be better to have unstemmed stopwords and apply it before stemming. This is more of an issue in highly inflected languages, where it's hard to create an unstemmed list of stopwords."

3

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

From the Lucene doc (https://lucene.apache.org/core/3_6_0/api/core/org/apache/lucene/document/Field.TermVector.html)
A term vector is a list of the document's terms and their number of occurrences in that document.

2. What should be added to the code to have access to the “term vector” in the index?

We can build our field with the class Field, wich use an FieldType object. (We actually use subclasses, eg. NumericDocValuesField) The FieldType can turn true or false the storage of the terms vector with the setStoreTermVectors(true || false) method.

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

So with Luke, we can see that the index is composed of 3203 documents and 26'602 terms.
We can also get the number of document with the IndexReader class, by the method indexReader.numDocs()
If we enable term vector in our field, we get the same amount of documents wich is normal because the amount of terms didn't changed, we just added additional informations in the vectors.

3.2

2. Explain the difference of these five analyzers

WhitespaceAnalyzer

The WhitespaceAnalyzer use the WhitespaceTokenizer.It is a tokenizer that *divides text at whitespace. Adjacent sequences of non-Whitespace characters form tokens.* (https://tool.oschina.net/uploads/apidocs/lucene-3.6.0/org/apache/lucene/analysis/WhitespaceAnalyzer.html) If we look with Luke, we can see that the WhitespaceAnalyzer give us the same amount of documents , but 38'125 terms (vs 26'602 terms for the standard analyser).

EnglishAnalyzer

Same as the StandardAnalyser, but will use the default stop words (by the method getDefaultStopSet()). Or with a given CharArraySet of stop words.
With Luke, it give us 25'716 terms.

ShingleAnalyzerWrapper 1 (using shingle size 1 and 2)

This analyser wrap another analyser. By default, it wrap a StandardAnalyser. It will generates more terms because in addition generating a single terms for a word for example, it will generate another term for all the couple of 2 terms.
Luke give us a total of 126'862 terms.

ShingleAnalyzerWrapper (using shingle size 1 and 3, but not 2)

Same as the previous one, but will generate single word terms plus three words terms for a total of 165'716 terms.

StopAnalyzer

The StopAnalyzer use a stop words list for the analysis. It will not tokenize the word wich are present in the list.
In our case, we used the "common_words.txt" for the stop words.
The results are for this analyzer : 24'025 terms

3. Analyzer informations

	Indexed documents	Indexed terms	Indexed terms in summary field	Top 10 frequent terms of summary field	Size of the index on disk	Time for indexing
StandardAnalyzer	3203	29'901	20'005	1.the 2.of 3.a 4.is 5.and 6.to 7.in 8.for 9.are 10.this	1.12 Mo	4581 ms
WhitespaceAnalyzer	3203	38'125	26'821	1.of 2.the 3.is 4.a 5.and 6.to 7.in 8.for 9.The 10.are	1.28 Mo	6032 ms
EnglishAnalyzer	3203	25'716	16'724	1.us 2.which 3. comput 4. program 5.system 6.present 7.describ 8.paper 9.method 10.can	924 Ko	4279 ms
ShingleAnalyzerWrapper 1 & 2	3203	126'862	100'768	1.the 2.of 3.a 4.is 5.and 6.to 7.in 8.for 9.are 10.of the	3.04 Mo	7696 ms
ShingleAnalyzerWrapper 1 & 3	3203	165'716	138'090	1.the 2.of 3.a 4.is 5.and 6.to 7.in 8.for 9.are 10.this	4.02 Mo	8579 ms
StopAnalyzer	3203	24'025	18'342	1.system 2.comput 3.paper 4.presented 5.time 6.method 7.program 8.data 9.algorithm 10.discussed	912 Ko	4308 ms

4. Make 3 concluding statements bases on the above observations

First we can see that the number of document never change, which is pretty normal when we are always using the same files with the same amount of line. On the other hand, the indexed terms can vary a lot. When we use a stop words list, it reduce a lot the number of terms. And when we use the shingle wrapper, it generate a lot more combinaison of term wich increase the number of terms.
We can see that the more of indexed terms we have, to more disk space and process time it take.
In conclusion we can assume that the more pertinent "top frequent terms" result is when we use the stop word list.

3.3 Reading Index

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

We choose to use the stop-words analyzer because otherwise we'll get only firstname initials in the top rank. With that configuration, we get :

```
Value : jr
Freq. : 125
```

```
Value : smith
Freq. : 43

Value : thacher
Freq. : 41
```

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

```
Top ranking terms for field [title] are:
Value : algorithm
Freq. : 963

Value : computer
Freq. : 260

Value : system
Freq. : 172

Value : programming
Freq. : 154

Value : method
Freq. : 125

Value : data
Freq. : 110

Value : systems
Freq. : 108

Value : language
Freq. : 99

Value : program
Freq. : 93

Value : matrix
Freq. : 82
```

3.4 Search

```
public void query(String q) {
    try {
        // 2.1. create query parser
        QueryParser parser = new QueryParser("title", analyzer);
        Query query = parser.parse(q);

        TopDocs topdocs = indexSearcher.search(query, 10);
        ScoreDoc[] hits = topdocs.scoreDocs;
        System.out.println("Searching for [" + q +"]");
        // 3.4. retrieve results
        System.out.println("Results found: " + topdocs.totalHits);

        for (ScoreDoc hit : hits) {
            Document doc = indexSearcher.doc(hit.doc);
            System.out.println(doc.get("id") + ": " + doc.get("title") + " (" + hit.score + ")");
        }
    } catch (ParseException | IOException e) {
        e.printStackTrace();
    }
}
```

1. Publications containing the term “Information Retrieval”.

```
Searching for [Information Retrieval]
Results found: 86 hits
634: Manipulation of Trees in Information Retrieval* (4.2585773)
657: Information Structures for Processing and Retrieving (4.2585773)
891: Everyman's Information Retrieval System (4.2585773)
2516: Hierarchical Storage in Information Retrieval (4.2585773)
292: An Information Retrieval Language for Legal Studies (3.930503)
651: A Survey of Languages and Systems for Information Retrieval (3.930503)
1032: Theoretical Considerations in Information Retrieval Systems (3.930503)
275: Dynamic Storage Allocation for an Information Retrieval System (3.6493618)
2070: A Formal System for Information Retrieval from Files (3.6493618)
2114: A Formal System for Information Retrieval from Files (3.6493618)
```

2. Publications containing both “Information” and “Retrieval”.

```
Searching for [Information AND Retrieval]
Results found: 17 hits
634: Manipulation of Trees in Information Retrieval* (4.2585773)
657: Information Structures for Processing and Retrieving (4.2585773)
891: Everyman's Information Retrieval System (4.2585773)
2516: Hierarchical Storage in Information Retrieval (4.2585773)
292: An Information Retrieval Language for Legal Studies (3.930503)
651: A Survey of Languages and Systems for Information Retrieval (3.930503)
1032: Theoretical Considerations in Information Retrieval Systems (3.930503)
275: Dynamic Storage Allocation for an Information Retrieval System (3.6493618)
2070: A Formal System for Information Retrieval from Files (3.6493618)
2114: A Formal System for Information Retrieval from Files (3.6493618)
```

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

```
Searching for [Retrieval AND (Information NOT Database)]
Results found: 17 hits
634: Manipulation of Trees in Information Retrieval* (4.2585773)
657: Information Structures for Processing and Retrieving (4.2585773)
891: Everyman's Information Retrieval System (4.2585773)
2516: Hierarchical Storage in Information Retrieval (4.2585773)
292: An Information Retrieval Language for Legal Studies (3.930503)
651: A Survey of Languages and Systems for Information Retrieval (3.930503)
1032: Theoretical Considerations in Information Retrieval Systems (3.930503)
275: Dynamic Storage Allocation for an Information Retrieval System (3.6493618)
2070: A Formal System for Information Retrieval from Files (3.6493618)
2114: A Formal System for Information Retrieval from Files (3.6493618)
```

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

```
Searching for [Info*]
Results found: 78 hits
208: An Introduction to Information Processing Language V (1.0)
239: Inefficiency of the Use of Boolean Functionsfor Information Retrieval Systems (1.0)
275: Dynamic Storage Allocation for an Information Retrieval System (1.0)
292: An Information Retrieval Language for Legal Studies (1.0)
397: A Card Format for Reference Files in Information Processing (1.0)
616: An Information Algebra - Phase I Report-LanguageStructure Group of the CODASYL Development Committee (1.0)
634: Manipulation of Trees in Information Retrieval* (1.0)
651: A Survey of Languages and Systems for Information Retrieval (1.0)
652: Use of Semantic Structure in Information Systems (1.0)
656: An Information System With The Ability To Extract Intelligence From Data (1.0)
```

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

```
Searching for [Information Retrieval~5]
Results found: 87 hits
634: Manipulation of Trees in Information Retrieval* (3.6113107)
```

```
657: Information Structures for Processing and Retrieving (3.6113107)
891: Everyman's Information Retrieval System (3.6113107)
2516: Hierarchical Storage in Information Retrieval (3.6113107)
292: An Information Retrieval Language for Legal Studies (3.3331008)
651: A Survey of Languages and Systems for Information Retrieval (3.3331008)
1032: Theoretical Considerations in Information Retrieval Systems (3.3331008)
275: Dynamic Storage Allocation for an Information Retrieval System (3.0946908)
2070: A Formal System for Information Retrieval from Files (3.0946908)
2114: A Formal System for Information Retrieval from Files (3.0946908)
```

3.5 Tuning the Lucene Score

With ClassicSimilarity class

```
Analyze took : 2261 ms
Searching for [compiler program]
Results found: 354 hits
3189: An Algebraic Compiler for the FORTRAN Assembly Program (3.0708408)
187: Compiling Connectives (2.5525208)
1676: The LRLTRAN Compiler (2.5525208)
1173: The Performance of a System for Automatic Segmentationof Programs Within an ALGOL Compiler (GIERALGOL)
(2.3474622)
364: On the Compilation of Subscripted Variables (2.3200622)
437: Compiling Matrix Operations (2.3200622)
1454: A Simple User-Oriented Compiler Source Languagefor Programming Automatic Test Equipment (2.216906)
123: Compilation for Two Computers with NELIAC (2.1264093)
409: CL-1, An Environment for a Compiler (2.1264093)
413: A Basic Compiler for Arithmetic Expressions (2.1264093)
```

With MySimilarity class

```
Analyze took : 3496 ms
Searching for [compiler program]
Results found: 354 hits
1173: The Performance of a System for Automatic Segmentationof Programs Within an ALGOL Compiler (GIERALGOL) (8.481136)
1454: A Simple User-Oriented Compiler Source Languagefor Programming Automatic Test Equipment (8.481136)
3189: An Algebraic Compiler for the FORTRAN Assembly Program (8.481136)
61: IBM 709 Tape Matrix Compiler (5.139705)
98: The Arithmetic Translator-Compiler ofthe IBM FORTRAN Automatic Coding System (5.139705)
100: Recursive Subscripting Compilers and List-Types Memories (5.139705)
123: Compilation for Two Computers with NELIAC (5.139705)
187: Compiling Connectives (5.139705)
205: Macro Instruction Extensions of Compiler Languages (5.139705)
280: A Preplanned Approach to a Storage Allocating Compiler (5.139705)
```

5. Describe the effect of using the new parameters.

Since we change the lengthNorm() method in our custom Similarity class to always return one, the computation won't take the number of terms in consideration.
We also can see that we have increase a lot the final score. In the ClassicSimilarity , the tf() method is squaring the frequency, while in our custom class, we apply a log on it.